Software Engineering in Games

Balazs Lichtl and
Gabriel Wurzer

Institute of Computer Graphics
Technical University Vienna

ABSTRACT

Our seminar lecture focuses on the ***process of making games***. As with any other piece of software, we can break up this development process into different *development phases*, which (in the simplest case for a software project) would be: *Analysis, Design, Implementation and Testing*.
Thus, our goal is to compare general software projects to game projects using these four phases, including one another phase, present only in game development. Starting with the definition of every development phase, we will use case studies and statistics to further stress the difference between a '*general*' software project and a *game*. Furthermore, we will deal with the important subject of *game engine licensing*, looking at the performance statistics, advantages, constraints and industry feedback. We will then conclude our talk discussing the *latest developments* in the game industry, where an increasing amount of players take over level and character design. In this context, we will also have a look at player communities and try to guess at future developments in this field.

Our intention with this lecture is to emphasize that game development is more than the writing of code. It is a creative process which must nevertheless be properly organized for the game to be successful.

Balazs Lichtl and Gabriel Wurzer
June ‘2001

INTRODUCTION

There are many answers to the question: "*What is a software project ?*". Some of them are [TJO1.1]:

- a project is a complex effort using different techniques and methods
- a project makes use of collaboration of people from different fields, with different knowledge and different forms of speaking and thinking
- Projects solve new and unknown Problems
- Projects stand under an extraordinary risk

Although these statements are not true for every game project, some of them *do* apply. The question is, now that we know that a game project is a sort of software project, can it also be described in the same terms used for general projects ? We will show during this lecture that in addition to this being true, new techniques of software engineering have evolved for games through the years (see Chapter 1, The Four (Five) Design Phases).

For us, a general *software project* is a project focusing on the *creation of software*. Consequently, success can be measured by taking a look at the resulting software.

In a *game project*, the product is a *game.* But, and here comes the point: *A game is much more than just its software*. It has to provide *content* to become enjoyable. Just like a web server: without content the server is useless, and the quality cannot be measured.
This has an important effect on the game project as a whole. The software part of the project is not the only one, and it must be considered in connection to all other parts: The environment of the game, the story, characters, gameplay, the artwork, and so on.

The difference between a 'general' software project and a game becomes even more elaborate when looking at some of the people involved in a game project (see Fig. I.1).
One major consequence of the rich diversity of the fields involved in the production of a game is that risk tends to be fairly high. We will further stress this point when looking at the current state of the game industry, in Chapter 2: Although the market for computer and console games is growing rapidly, there are only a handful of commercially successful companies. As a consequence, instead of writing the whole game themselves, game companies tend to use third-party game engines to shorten their development cycle. We will take a look at the most prominent two, Quake III and Unreal Tournament, in Chapter 2.1.

Project Manager
Marketing & Sales Manager
Game Designer
Lead Architect
Software Planner
Programmer
Quality Assurance Technician
Artist
Motion Capture Technician
Musician
Sound F/X Technician
Playtester
Technical Support Technician

Fig. I.1: people involved in the production of a game

In the fast-growing field of software development, and in the even more rapidly growing sector of game development, the future is hard to predict. After giving you a short summary of the preceeding chapters, we will try to extrapolate some of the latest developments concerning games in Chapter 3, The Future of the Game Industry. We will base our prognosis on written reports by the leading game companies concerning their latest games (Epic and Digital Extremes, this is). This will (with your appreciated participation) lead us directly to public discussion, with which we conclude our seminar lecture.

1. THE FOUR (FIVE) DEVELOPMENT PHASES

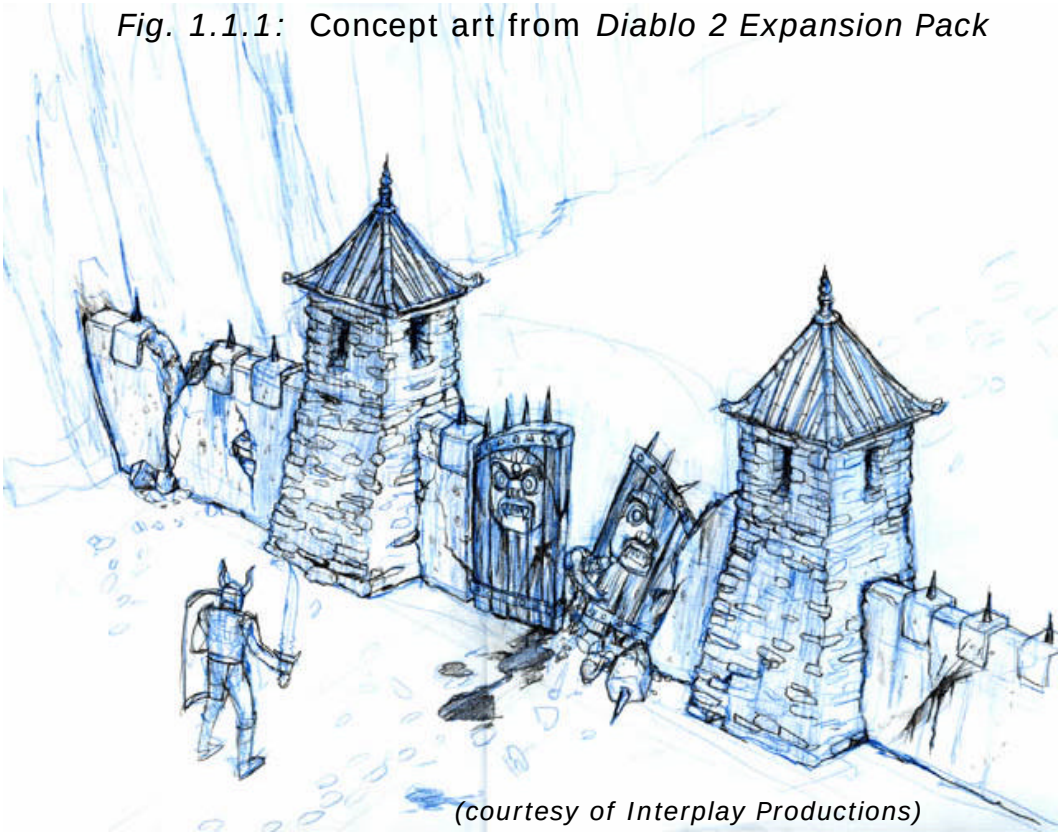
As you would assume, a normal software engineering cycle [30] starts with the analysis phase.
In game engineering, however, there is an additional phase before the analysis: the game concept phase. The main task here is to make the decision of what sort of game to produce. This means: Finding a main game idea, and determining how the game will look like. The game concept phase is described in detail in the next section.

1.1 The Game Concept Phase

Finding the initial concept

This is the creative work that eventually will make a game company start a game project. This means: Finding the idea, setting the scenario, drawing some sketches, drawing some posters, followed by presenting and selling all these to the people who will be involved in the project (see Fig. 1.1.1).
Of course, this phase is not managable, planable or improvable by means of software project management. Thus, game companies tend to give their creative people some free time away from their desk, so they can concentrate on finding new and exciting ideas. Origin Software, for example, sends their main game programmers on vocation after a game has completed. This ensures that they will return fresh and relaxed when a new project starts, and with new ideas in their minds.

Fig. 1.1.1: Concept art from *Diablo 2 Expansion Pack*



(courtesy of Interplay Productions)

Finding a good idea for a game can be a tough job, and every designer tends to have his own way of doing it. What game designers have in common are the two basic recommendations to newbies [8, 9], which are considered to be mandatory in the industry:
The first thing is that you *should play games*. It sounds simple, but this really helps figuring out what is on the market, what you can sell, and how good you must to be in order to be successful.

Second, an idea that is 'brand new' sells much better than something that has been produced many times. Products that don't offer something new are considered as "me-too", and will be much less successful than the original game they try to reproduce.

Designing the gameplay concept

After the designer has found his 'main idea' for the game, he must work out the details, which can then serve as a communication platform for the whole team in the later software design process (see Fig. 1.1.2).
This task (generally referred to as concept design) is often made by a team that is different from the software engineering team, and thus concentrates only on the contents of the gameplay, and not *how* it can be realized.

Fig. 1.1.2: Designing the gameplay concept means working out the details of the game

There are many rules [9] that should be followed when designing a game: First, you should *learn from your errors.* Moreover, and that's an even more important point in the game biz, you should also learn *from other's errors* ! Never make the same mistakes as others, learn what you could improve.
The third point is that you should *design the game for the player*. This certainly is the most challenging goal of all, and if you miss it, the game will not become enjoyable for the player, and it will not sell as good as you want it to sell.
Another important rule is that *you cannot substitute technological highlights in place of fun-making gameplay*[3]. Your game can have special effects, lighting or anything else you can imagine, but if it has a boring gamplay, no one will play it longer than ten minutes. This is similar to the film industry, and is discussed in greater detail in [10].

Finding the treatment

The treatment is a document which defines the major features of the game and attempts to paint a picture of the game that sets the mood for the team to follow. It is composed of the storyline, the look-and-feel guidelines and a description of the basic mechanics which will be later used in the game, as evolved in the gameplay concept design. Many games nowadays use pre-defined settings (such as, for example, the world of Star Trek or the dark middle ages). On there other hand, there is a growing interest in carefully thought out storylines which (in the case of games like 'Half-Life') give the players the feel of actually being there and 'taking hold' of the situation. Moreover, there are some games on the market that have a non-linear gameplay, which means that the story is either set each time a new game is started (like in Indiana Jones 4) or evolves as the player progresses through the game (as seen in Blue Steel). Nevertheless,

Fig. 1.1.3: The treatment adds substance and character to the game

there still is much space for development in this field (with today's rapid game development cycles leaving not much space for such gimicks). On the other hand, *modularisation* takes place also in the treatment design. Unlike in the beginnings of the gaming industry, stories are now told *in episodes* (as opposed to whole *quests*). This certainly has the advantage that new *expansions* can easily be provided, and games generally have a *faster pace***.**

Before moving on to the next subphase, the treatment is thoroughly reviewed (as you know, finding errors early does significantly reduce the risks of a project).

Making the Overall Design

The Overall Design document is the first draft which describes the game in detail. This is, how is the game played, how does it work (semi-technical), how does it look, what are the rules, how do all the units in the game work in detail, what is the plot and so on...
One major point in this context is to set the type of game, or (as it is often referred to) the *game genre:*

Action games typically rely more on hand/eye coordination than on story or strategy. They are fast-paced and reflex-oriented, the most prominent example being the classic first-person perspective 3D shooter.

Fig. 1.1.4: Action Games such as Duke Nukem 3D (left), and Unreal Tournament (middle and right) feature fast, reflex-oriented gameplay

Strategy games emphasize logical thinking and planning. They often stress resource and time management, which usually takes precedence over fast action and character involvement. Tactical organization and execution are necessary, and the game creators usually place the decision-making skills and delivery of commands in the player's hands (see Fig. 1.1.5, next page).

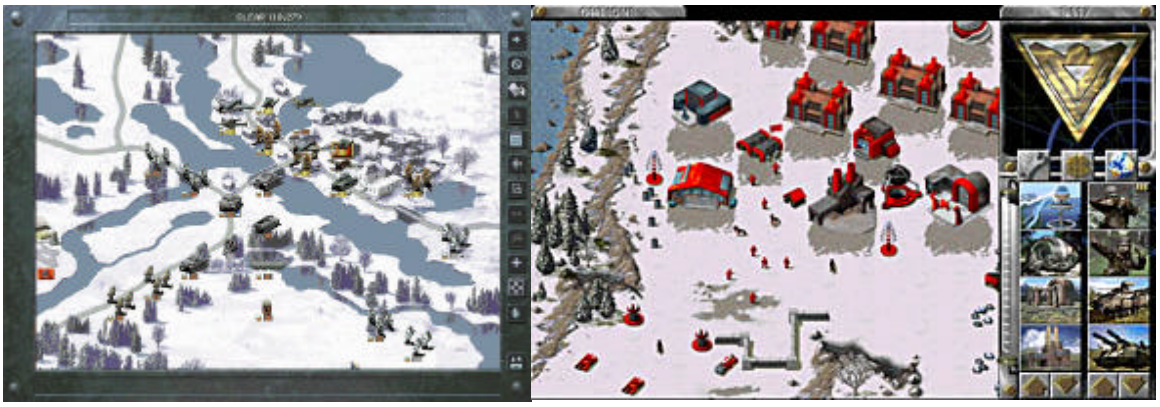


Fig. 1.1.5: Strategy Games concentrate on resource management and tactical options. The two examples presented herein are Panzer General II (left) and Command and Conquer: Red Alert

Adventure games involve the player in a journey of exploration and puzzle-solving. These games usually have a linear storyline in which you, the protagonist, set out to accomplish a main goal through character interaction and inventory manipulation.

Fig. 1.1.6: Adventure games pose several puzzles which require solving. As an example, we here present two screenshots from Leisure Suit Larry (a would-be playboy who always terribly fails in getting the girl of his dreams, to our delightful pleasure)

Roleplaying Games (RPGs) are similar to adventure games, but rely more on character growth and development (as the game advances, the player character becomes more powerful concerning his abilities). Conversation and strategic combat are also a major part of RPGs, as well as a non-linear storyline (adapting to the actions taken by the player character). Graphics are less important than a catching story and awarding gameplay (this is, you get points for killing monsters). See Fig. 1.1.8 (next page) for further details.

Fig. 1.1.7: Roleplaying games usually involve a quest, which the character has to master. Story is generally more important than graphics with RPG's.

Sports titles simulate a game from an instructional or player perspective. Realism is important, as are fast action and tactical strategy.

Fig. 1.1.8: FIFA 2001

Sims simulate a given animate or inanimate object or process. Most often, the gamer is placed in a 3D first-person perspective of a re-created machinery such as planes, tanks, helicopters or submarines. But also the evolvement of cities over time can be an issue with Sims.

Fig. 1.1.9: SimCity 3000

Puzzle games include older, more historic games of leisure such as cards, tile games, trivia, word, or board games (chess being a perfect example).

Fig. 1.1.10: Battle Chess

Fig. 1.1.11: Microsoft Combat Flight Simulator

Simulators are games where the exact behavior of a vehicle are tried to simulate, trying to give the player exact the same experience as he was really "driving" that vehicle. This game genre was in the last years not that popular as before from 1992 till 1996, there are still some titles available. Examples are the Microsoft Flight Simulator (over many versions), other military flight simulators, and more Auto races, F1 Auto-Simulations. The problem with this type of games is, that many of them are simply action games, they do not even try to simulate the reality, and only a little piece of them are "real" simulators, where then the handling and gameplay are near that in reality.

After dealing with the game type, the *perspective of the game* is set:

First-Person Perspective games are the equivalent of "seeing it through your character's eyes". In the game industry, this seems to be the approved choice for action games and simulations. In some games (example: Auto races) the player can choose between more first and third person perspectives.

Fig. 1.1.12: Did you see Mr. Burns ?

Third-Person Perspective, generally known as "over the shoulder"-view, is another popular choice among game designers. The advantage here is that you can observe what your player character 'does', while at the same time having full control over his movements.

Fig. 1.1.13: Lara Croft

Top-Down Perspective games give the feeling that the camera is hovering top-down over the game itself. Strategy games often rely on this type of perspective.

Fig. 1.1.14: Top Down

Isometric games offer a slightly tilted "three quarter" view of the game, which gives a sort of impression of 3D. Adventure game designers as well as strategy game producers seem to like this view, because they can sell their product as being "3D" (which, technically, is simply not true). For an example, see Fig. 1.1.9.

Flat, Side-View games give a two-dimensional "side view" of the game, which scrolls by around the players. These types of games were especially popular with console developers (such as the Game Boy).

Fig. 1.1.15: Commander Keen

Text-Based games don't use graphics at all, or very sparingly. Aside from classic text adventures from the early '80s, trivia games also fall into this type of game.

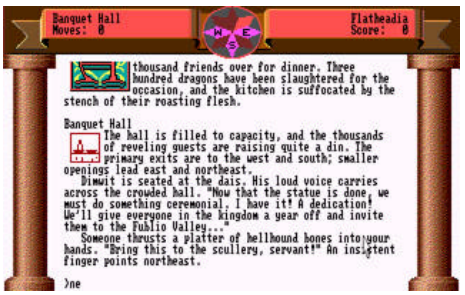


Fig. 1.1.16: A text Adventure

Coming back to the Overall Design Document, we note that the **target platform** is also set here. As discussed in [5], PCs have many advantages against consoles: They are expandable in hardware, the monitors give a much better quality (with the player sitting close to it, giving a better immersion), and, last but not least, the networking capabilities of the PC are much better than that of the consoles. The only disadvantage of PCs is the relatively high price (consoles only cost around $300).
However, it is important to say that both platforms are *not* competing with each other. This is mainly because of hardware differences, with each platform having its own strengths and drawbacks concerning certain types of games. The competition between them is only important in driving the evolution of the hardware: Today, the most powerful driving force for the console market is that they must keep up with the fast development of PC graphics hardware (or vice versa, as you prefer). Furthermore, a brand new trend in the console world is the movement of games onto mobile handheld devices [34]. This is either in the form of game-only handheld consoles (e.g. "*Game Boy Advance*"), or on java-enabled mobile devices (such as cell phones). New mobile networks, such as GPRS and UMTS will eventually bring network gaming onto mobile devices.
Summing up, there is a whole lot of target platforms to choose from. As we will see in Chapter 2, third-party game engines nowadays are cross-platform-compatible, so that both PC and console versions can be developed without additional work. What a relief for game programmers !

1.2 The Analysis Phase

After the phase of designing the game concept, the project turns in a more or less normal software engineering project, where analysers, designers and programmers (referred to in game development as *coders*) try to produce a piece of software that should eventually become a good game when combined with the content coming from the graphics designers.

In software engineering, the goal of the analysis phase is *to produce a complete description of the problem domain and problems to be solved*. This covers documenting every functionality the customer wants, and specifying the requirements of the system (e.g. who will use it, what are the hardware requirements). The product of the analysis phase is the requirements definition. These requirements define, what the software must can do, and what it must not to be usable for the original problem.

In game engineering, this phase differs from "normal" software engineering in the meaning that the requirements are not defined by some type of customer, but by the game concept, especially by the "Overall Design Document".
This makes not much difference, besides that the requirements are more specific in meanings of performance, user interface, and are much more hard to fulfill. This is because if you want to make a game that sells well, you must produce something excellent: It is *not* enough to make something that works, something average, as in the case of a normal application software.

1.3 The Design Phase

The goal of the design phase is to produce a detailed written model of the system, which must meet all the specified requirements.
The main idea for the design phase is thus to model ***WHAT** is needed* for the software (or game) to become usable, *not HOW it can be implemented.* In games, this phase can be further broken up into four sub-phases:

Making the Technical Architecture

The **technical architecture** is the initial technical document of the project. It breaks work down into modules, which can later be implemented. Furthermore, a reuse plan is generated to make optimum use of components that have already been developed. Briefly speaking, this step is more or less the same as in general software engineering.
At this level, reuse[4] is an important task to work on. In this phase the decision must finally be made what to implement in-house, what to reuse and what to buy from others. Furthermore, the module interfaces have to be designed for maximum reusability in the future, that means, they should be kept minimal and simple, ensuring minimal cohesion between the modules.

Module Design

As a handover document between game designers and programmers, each *module gets its own design document*. This document is highly technical, describing how a particular module works and specifying the interfaces for the modules (so that programmers precisely know what *their* job is, and what is not). Note that there are many types of modules, for example code modules and artwork modules. The module design document is also important for reuse in other projects, especially for the decisions in the architecture design of the usability of a module for a specific purpose. This is generally done in a way that the module interfaces are kept compatible with existing modules (so they can be integrated into the new software, and must not be redeveloped with similar functionality).

1.4 The Development Phase

The goal of the development phase, in general terms, is the realization of the specification into a running program. For games, this is only to a small extend the writing of code. Artwork, music, sounds and other many more things need to be produced to form the program. Furthermore, extensive testing is required for the game to work properly. Remember that even though there is a later testing phase, errors do happen during the development of the modules and need to be cleared right away.
The development phase is further broke up into:

Detailed Technical Design

Each module is written in tandem with a **detailed technical design document**. This document tries to explain to other developers exactly how the module functions, including the reasons behind different design choices and test scripts. It can be seen as a *journal* of the development of the module, and is as important as the module itself. Developers tend to forget about this piece of work, which results in many problems in reuse and maintentance of the module. The reason for this is mainly that stress and irrealistic deadlines prevent developers from spending some extra time on documentation.

Development

Every module is produced from some sort of design document (for artwork, this would be a style guide of some form, for software it is the module design document). The *result* of the development phase *is*, in all cases, a module *(artwork or code).*
At this level individual code reuse can be done by each programmer. This is hidden from other team members and the management, and depends mainly on the individual experience of the programmer. This is the lowest level of reuse, and results in the less productivity improvement.

Unit Testing

This is the *testing of the code written by the developer* (usually by himself). It follows a script written by the developer as a part of the detailed technical design document.
Testing can be made with two different techniques: **black box** test means, that the unit is triggered at the external interfaces, and the result values are compared to the defined ones. If there is no difference, the module is assumed to be correct.
White box testing, on the other hands, sees in the internals of the module, and test the internal behavior. With this test method, the performance bottlenecks, and other non-functional errors can be detected more efficiently, but it is in most cases more time-consuming, and requires the understanding of the module internals.

Fig. 1.4.1: Unit testing is usually handled by the programmers themselves

Sign Off

When all unit tests are successfully completed, the developer packs his module up and checks it in to source control[1]. This concludes the development of that module.

[1] This is sort of a database which holds all code of the programming team

Integration

The finished modules are integrated incrementally into the whole software. This is done in a controlled manner, because it is usually not possible to integrate the modules in any desired sequence. The interfaces declarations decides about this problem. It can be even necessary to write stub modules to integrate the software, because if it would be integrating with the real modules, errors can't be found, or only with much higher effort.

Integration Testing

This time, the developer *tests the completed module* to see *if it fits in with the whole game* software. Again, test scripts that were defined in the technical design document are used.

1.5 The Testing Phase

The goal of the testing phase is the detection of errors in the complete program. If this phase fails, the program might be shipped (causing the project to fail). Also, the efficiency of the system is measured in this phase (which might lead to optimization in some critical modules). The product of the testing phase is an error-free, shipable program.
For games, we distinguish between:

System Testing

The whole system is tested and produces errors that wouldn't have been found by unit and integration test. During this phase, errors in the specifications eventually tend to show up . And this in turn means, that massive efforts have to be put in correcting that errors (which is, of course, very costly, and causes much risk, and eventually the failure for the project).
In a game project the correctness of the resulting program is even more important, as in a normal software project. If a game hangs often, or has other bugs, that prevent normal operation, the fun at gameplay is fast gone, and the player will never play the game again, nor buy other games of the same producer.

Quality Assurance

The purpose of QA is to make sure that the aesthetics (game atmosphere, menu screens, manual, etc.) are user friendly and self-consinstant, while conforming to the game style section of the game design document. It is not meant to find defects in the program, as they should have all been already cleared out by this stage.
After this test is passed, it is considered the game to be "game complete" [13]. This means, that the game software and content are in the quality for that they where designed, and its "free" of bugs.

Playtesting

This is the final stage of testing, in which the total game experience is tested. This is: How does it play? Is the manual any good? How is the learning curve? Are there any gameplay issues ?

Fig. 1.5.1: Playtesting is essential for fine-tuning the game

Playtesting differs from all other tests in the software domain: The testers explore every possible location and every possible puzzle in the game. They try to outsmart the system, or, if possible, to crash it. Cheat codes were mainly invented for this reason, so that playtesters could position themselves everywhere in the level or get a close look at the chief monster without getting killed. In fact, cheat modes are rarely implemented as "a feature", since that would undermine the goal of the game.

If the test phase is completed, the game production is closed, and the success of selling the game is up to the marketing. The marketing process has to be started even before finishing the game, by making playable beta releases available for download for the gaming community. This way the development team become valuable feedback about the game, and can correct design errors to make the game more accepted at the time of shipping.

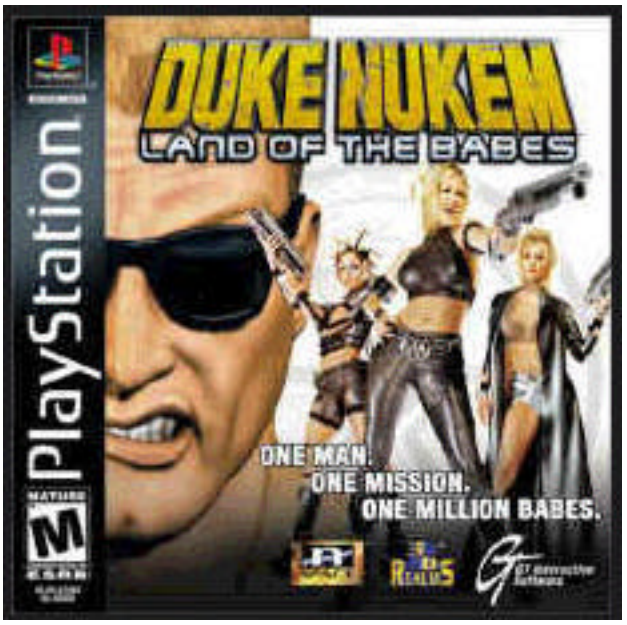


Fig. 1.5.2: Ready for shipping

2. STATE OF THE ART IN GAME ENGINEERING

The game industry's total sales nowadays overcome those of the movie business. Although there is much money involved in this business, only a handful game companies are successful. One reason for this is the high competition, as well as the low prices players are willing to pay. As a matter of fact, publishers had to come up with a solution that both decreases production time and manages to stay in touch with today's rapid changing technology: The idea of Game Engine Licensing was born. We will herein present the two most prominent ones, Quake III Arena and Unreal Tournament.

2.1 Third-Party Game Engines

Once again, the idea behind licensing a game engine is

- *to decrease the length of a game project*

  The time that would have been invested into developing a proprietry engine could be used for a new project

- *to reduce the risk of a game project*

  Newly developed code tends to take a lot of time to test and debug, whereas third-party engines have already been used in a number of games are thoroughly tested. Furthermore, maintenance and development of the engine is handed off to a third party, which keeps it up-to-date (e.g. integrating new graphics hardware).

- *to reduce the number of people working on a game*

  Most engines nowadays are object-oriented or provide some sort of scripting functionality. However, this means that less specialized people can be used, resulting in lower project costs.
  It is notable that when we talk about a game programming team, we refer to around 10 people. This is a big difference to normal software projects, where a lot more people can be involved in a single project.
  You can see from the fact that the number of people working on a game needs to be reduced (with only a handful people really involved in production), that profit spans are very thin in the game domain. That's another difference to normal software projects.

There are, in fact, many game engines that are free of charge (e.g. using the GNU General Public License). Some were made publicly available when the corresponding game title ceased to sell (most prominent examples being DOOM and QUAKE). Others simply were developed for the open source society (e.g. FLTK: www.fltk.org, Parsec: www.parsec.org), and continue being developed. For a comprehensive list of 3D Engines (free and commercial), we recommend visiting Karsten Isakovic's 3D Engines List (www.cs.tu-berlin.de/~ki/engines.html).

The first problem with free engines is that most licenses prohibit selling the games that were made with them. This can be either by forbidding charges for the game (except distribution costs), or by stating that all source code has to be made publicly available.
A second problem of the free software movement is that most engines can't keep up with the fast-growing game industry requirements. Furthermore, graphics hardware manufacturers don't give away

development kits for their newest graphic cards for free. These two facts also seem to threaten the future of games under Linux [26].

The solution is thus to license an up-to-date game engine, of which we want to present the two most common:

2.1.1 The QUAKE III Arena Engine

Released by id software in '2000, this engine provides all of the latest 3D Technology standards developers could possibly dream off (shaders, curved surfaces, 32-bit color, special effects, networking and fast hardware rendering). The engine is restricted to a number of 'big' QUAKEIII Arena licensees, in order to protect the market from a massive flooding with QUAKEIII-engine-games.
The QUAKEIII Arena License furthermore states that if additions to the engine are made, the technology can be kept and re-licensed by the licensee. According to id technologies, the engine costs 8% in royality of the price per title. For that price, the costumer also gets ports of the QUAKEIII Arena engine for PC, Mac, Linux and Dreamcast .

Fig. 2.2.1.1: The QuakeIII Arena Engine

2.2 The UNREAL Tournament Engine

Unreal Tournament, released in November 1999, was, in a way, an accident. After the original Unreal was completed, Epic Megagames wanted to follow up the project with some sort of add-on pack. Unreal multiplayer code was very poor, so the team felt that an expansion that improved multiplayer would be ideal. As feature lists grew and patches to *Unreal* were released, the add-on turned into a complete and independent game.
The engine itself features state-of-the art graphics, networking support and Sound. In addition to the object-oriented C++ code, there is a JAVA-like scripting language named UnrealScript. The engine runs both on Windows, Linux and MacOS.

Fig. 2.2.1.2: The Unreal Tournament Engine

A few facts about the Unreal Tournament Engine:
Project budget: $2 million
Project length: 18 months
Team size: approximately 16 developers
Code Length: 350,000 lines of C++ and UnrealScript
Critical development hardware: Pentium II 400s with 256MB RAM and Voodoo 2 or TNT-based cards
Critical development software: Microsoft Visual Studio, 3D Studio Max, UnrealEd

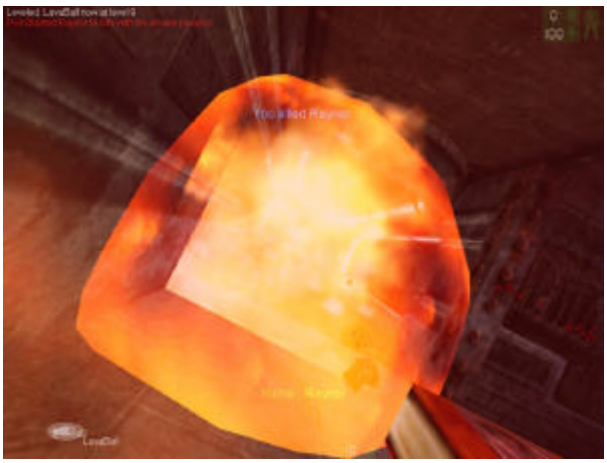


Fig. 2.2.1.3: The Unreal Engine features a variety of amazing effects, such as explosions or fire trails

A Brief Summary

Game Concept Phase

- Finding the idea
- Working it out
- Setting the Story

Analysis Phase

produce a complete description of the problems to be solved

Design Phase

produce a detailed written model of the game

Development Phase

realization of the specification into a running program

Testing Phase

fine-tuning through playtesting

A **software project** is a project focusing on the *creation of software.* A **game project** is more than a software project. In addition to its software, the game has to provide *content (story, environment, artwork,…)* to become enjoyable. This makes it a difficult and risky enterprise, which must be managed in order to be successful.

The development of the game can be broken down into five **development phases**. Each phase focuses on a different aspect of the game.

Beginning with the game idea, the story is slowly expanded into a mental model model of the game, the **game concept**. This model is then reviewed in the **Analysis Phase** in order to find what will be subject to the game project. These findings are then written down during the **Design Phase** to form a formal specification for the game. The specification is realized into a game during the **Development Phase**. The **Testing Phase** then concludes the software development cycle.

Today, most game projects license **third-party game engines** in order to reduce risk, decrease production time and to stay in touch with today's technology. Furthermore, the game engine vendor takes care of the development and support of the game engine, which relieves the game programmers from doing less productive work. Another advantage is that third-party game engines tend to be thoroughly tested.
It must be noted that third-party game engines do have their price, and the developing firms are very restrictive about whom to give a licensing contract.

We will now move on to a more interactive part of our lecture, in which we want to present some recent trends and developments in the game industry. We have compiled some interesting articles from the web for you which should form the basis for a concluding public discussion about the future of the game industry. **Please feel free to interrupt us any time and contribute your ideas**. This will make the lecture much more lively and interesting.

3. THE FUTURE OF THE GAMING INDUSTRY

There are some interesting new trends in the gaming industry that will dramatically change the way games are made. In this chapter we will try to give you an overview of these developments, which then will serve as a basis for a public discussion at the end of the lecture:

Decentralization of development, user communities

> *Nearly every Epic and Digital Extremes employee frequented message boards dedicated to the subject of Unreal and Unreal Tournament. The majority of Epic employees were drawn directly from the gaming community, either through mod projects or independent game work. Keeping in contact with the gaming community allowed Epic to focus on the target audience during the design process.*
> *This community-mindedness greatly contributed to the quality and completeness of Unreal Tournament. We had a very good idea of what players wanted. As I mentioned earlier, we often posted controversial design questions on public message boards to gauge public reaction. The results of these polls were taken into consideration when the feature in question was implemented.*
>
> Source: GT interactive

As mentioned in the above article, the process of making a game has already become a decentral effort. In fact, the development team of Unreal Tournament consisted of people both in the US and Canada, the communication being held over the Internet. Furthermore, the development takes place both inside and outside the development team:

> *In December, I downloaded a sample of a new Unreal mod under development by an Australian named Jack Porter. The mod, UBrowser, was a server browser using a Windows-like GUI. It was impressive, so I showed it to James Schmalz, lead designer at Digital Extremes, who said, "We need that, we need to hire this guy." A few weeks later Jack was a part of the team, expanding his UWindow GUI and reworking Unreal Tournament's menus to use the system. Jack fit into the team perfectly, bringing a complete solution for the interface and menus as well as his own independent programming initiative.*

There is a lively and creative community evolving around level editors and mesh modeler programs of various game engines (see www.planetquake.org/polycount/ and www.planetunreal.org). Here, new levels, sounds, meshes and rumours are discussed in detail: Players share experiences with eachother, download maps and initiate network plays. See Fig. 3.1 (next page) for a screenshot of the polygount homepage, where the newest gossip about new quake modules (levels which were created by users and can be loaded into the game) is exchanged.

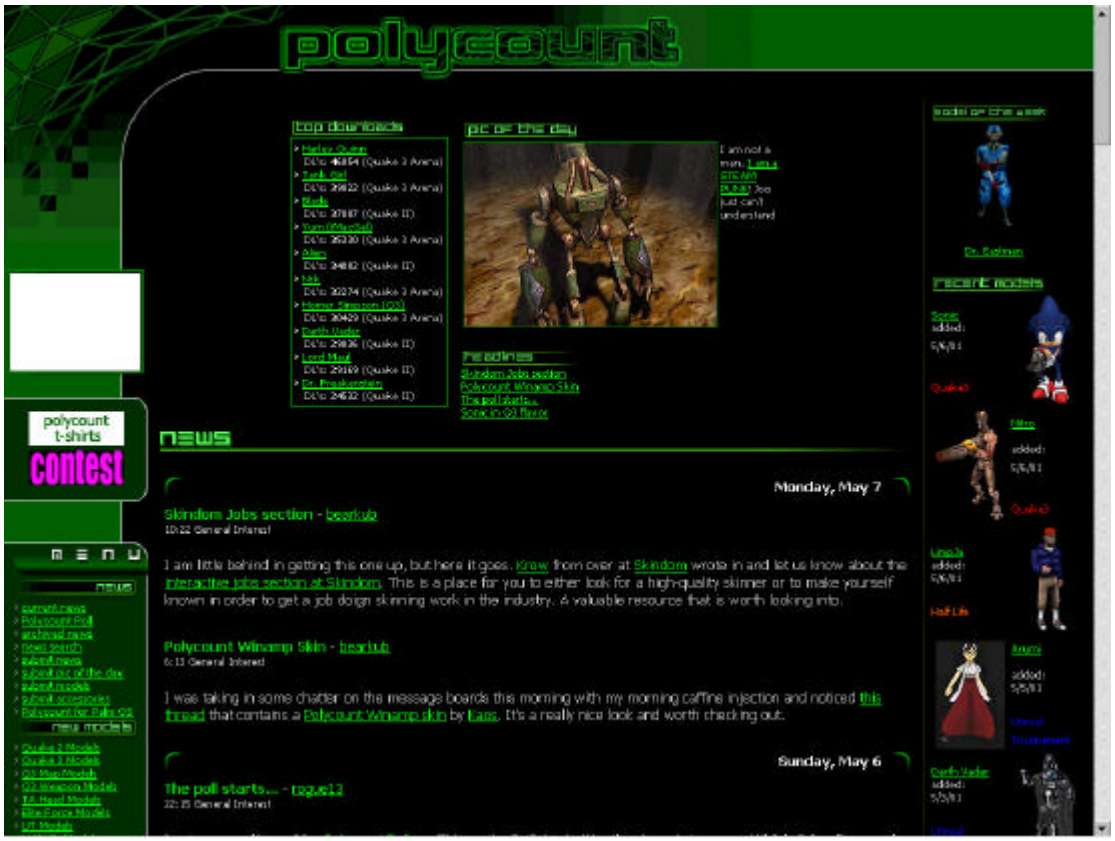


Fig. 3.1: "Polycount" is featuring artwork and models for Quake and Unreal. This is one of the places where users can contribute to their game, and are sometimes recruited by professional game companies

It is the strong belief of the authors that in future, game developers will increasingly take their game ideas directly from the gamers community. This would in turn mean that the game development process will be subject to further decentralization. New mesh modelers and map editors will evolve, eventually putting level and character design into the hands of the players.

This development doesn't always have advantages, as is explained on the home page of 3D Realms (the makers of Duke Nukem 3D):

> *It happens every time a mod or other project gets foxed: the message boards light up with the collective rage of a community that feels it's been wronged, and the developers or publishers are accused of squashing the little guy in favor of more money. The mod community begins calling for the developers' heads on platters.*
> *The most common accusation hurled is that the company is only doing it because of greed. While money is almost certainly tied into it, the predominant issue is one of intellectual property law.*
> *Recently I had to tell a modeler that several models he'd done violated copyright law--he'd converted Quake III Arena models for use in Unreal Tournament. The issue was similar to the recent axing of the Duke It Out in Quake mod--both groups had taken one company's intellectual property from one game and ported it into another. This is the most common and obvious violation of copyright law in community development and it should be common sense not to violate it, but apparently it isn't when it's the biggest puddle the community steps in. But despite the ubiquitous outrage that inevitably follows with each fall of the axe, the companies can and must put a stop to that development […]*

The author goes on that if every piece of artwork could be freely ported to other games, the games themselves would become dispensable. Hence, the game industry wants stronger legislation for intellectual property (see next), which is exactly the opposite of what many game fanatics want.

About the stronger protection of intellectual properties

> We have already stated that cloning a game (making a game with similar or identical gameplay and design) violates the copyright laws for intellectual properties. Hasbro Interactive, for example, won a lawsuit against a publisher who sold clones of old Atari games, for which Hasbro had bought the copyright [19, 20]. As outcome of the lawsuit, the publisher had to stop selling these titles, as well as recall the ones already sold. Some of the titles included very famous games, such as Tetris and Asteroids.
> Hence, copyright legislation can have a big influence on the gaming industry: You must ask the publisher of the original game for permission for making a clone, and you must eventually pay for this permission. On the other side, the publishers the original game can stop others from selling clones. That means that, because the freedom of game developers is more limited than before, the value of new ideas is increased.
>
> Why does this development come that late? The answer is, now that the game industry has grown big enough, publishers are eager to protect their own intellectual properties. In a way, as this is quite normal in other commercial sectors, we see that the game industry has over time evolved into a full-grown business sector, with much elbow-bullying and competition.
>
> It is interesting to mention that a game idea alone is no intellectual property[7]: It can't be protected by laws, it can't be sold, and the "idea" itself has no physical value. However, after it is realized in some form (e.g. published in a game), this property belongs to its publisher, and is then fully protected under copyright laws.

Where we go from here (conclusion, public discussion)

We have seen that the process of making games has already become a decentral effort, with development taking place both inside and outside the development team. Large player communities have evolved on the internet which concentrate on providing new levels, meshes and plugins (modules) for various game engines.
We think that in the near future, the development of ***content*** *(levels, art, meshes)* will be laid into the hands of the gamers community. As we have mentioned earlier, there are certain jurisdictional problems arising because of copyright laws.
In the long term, it is even possible that "game studio" software will be reinvented, so that the players could build their own games with minimal effort. That in turn would also place the ***software*** part of the game project into the hands of the player. The role of the game companies would then be to supply the player with *access technology*, with the effect that competition would dramatically increase. In a way, this trend would be similar to the developments concerning the use third-party engines, which we have discussed in Chapter 2. Game studio providers would take over the technological aspects of a game, while players concentrate on defining the gameplay. Needless to say that *commercial* game development would then get even tougher.

- How do these trends sound to **YOU** (is decentralisation good ?)
- Have **YOU** seen similar trends of decentralisation in the game industry ?
- What do **YOU** think the next step will be ?

References

1. Game Engine Design, Theory and Practice (unser Einstiegspunkt und spätere Grundlage für den Vergleich klassischen Software-Engineerings und Game Engineerings)

2. +Software Engineering is Not Computer Science, *By Steve McConnell* Gamasutra, *December 16, 1999,* URL: http://www.gamasutra.com/features/19991216/mcconnell_01.htm (es diskutiert im allgemeinen wie softwareentwicklung jetzt ausschaut: computer sience, und wie es ausschauen könnte, wenn es richtig betrieben sein würde.)

3. The Focus Of Gameplay; by Geoff Howland (www.gamedev.net) (es geht hauptsächlich um die spielbarkeit der spiele, und wieso spiele spielbar oder unspielbar sind)

4. Whatever Happened to Reuse? *By Steven Adolph,* Gamasutra, *December 13, 1999,* URL: http://www.gamasutra.com/features/19991213/adolph_01.htm (diskutiert, wieso es kein reuse betrieben wird, und was die wirtschafliche gründe sind.)

5. Is PC Gaming Dying? The PC market is changing. Will gamers be left out in the cold? March 12, 2001 (http://pc.ign.com) (diskutiert das spielemarkt, udn vergleicht pc und konsolemarket: ergebniss pc gaming is not dying)

6. Irreconcilable Differences: Game vs. Story by Mark Barrett (www.gamedev.net) (diskutiert, dass ein spiel und sein story korreliert sein sollten)

7. Can I Sell My Game Ideas? by Geoff Howland, (www.gdcentral.com) (eine kurze gedankengang über eigentumsrecht von ideen)

8. A Primer for the Design Process, Part 1: What to Do, *By Tim Huntsman,* Gamasutra, *June 30, 2000,* URL: http://www.gamasutra.com/features/20000630/huntsman_01.htm

9. A Primer for the Design Process, Part 2: What to Think About, *By Tim Huntsman,* Gamasutra, *July 7, 2000,* URL: http://www.gamasutra.com/features/20000707a/huntsman_01.htm (diese beiden artikel beschreiben, wie man ein spieldesign angehen soll, was man dabei beachten muss)

10. Dogma 2001: A Challenge to Game Designers, *By Ernest Adams,* Gamasutra, *February 2, 2001,* URL: http://www.gamasutra.com/features/20010202/adams_01.htm (diese vergleicht das spieleentwicklung mit der filmindustrie, wo oft nicht qualität im künstlersinn, sondern quantität der spezialeffekte zählt.)

11. Controlling Chaos in the Development Process *By Tim Ryan,* Gamasutra, *August 6, 1999 (welche risken es bei einem projekt gibt, und wie man damit umgehen soll)*

12. How do I make games – A Path to Game Development, by Geoff Howland (www.gamedev.net) (es ist ein kurzer tutorial, was man alles übern sollte, damit man ein game-entwickler werden kann.)

13. The three completes of a game, by Geoff Howland (www.gamedev.net)

14. It's All in Your Mind: Visual Psychology and Perception in Game Design, *By Hayden Duvall,* Gamasutra, *March 9, 2001,* URL: http://www.gamasutra.com/features/200103009/duvall_01.htm (ein allgemeines dokument über die natur des menschen in zusammenhang mit games)

15. The Game Development Cycle, (www.gdcentral.com) (beschreibt kurz die phasen der spieleentwicklung)

16. Creating Virtual Worlds by Tels **, (**www.gamedev.net**)**

17. Designing for Online Communities by David Michael, (www.gdcentral.com)

18. IGF 2001 Preview: Ten Prepared to Win, *By Damon Brown,* Gamasutra, *March 7, 2001,* URL: http://www.gamasutra.com/features/20010307/brown_01.htm

19. +Why the Hasbro Lawsuit Should Terrify Game Developers - And what we can do about it, by Diana Gruber, (www.gamedev.net)

20. Three Sides by Sean Timarco Baggaley, (www.gamedev.net)

21. On Gameplay - or Creating, Developing, and Balancing Your Game Concept, by The Constable (www.gamedev.net)

22. Biology and Gaming: Why Women Don't Play Games, by Philippe O'Connor (www.gamedev.net) [es wird versucht zu erklären, warum Frauen angeblich weniger spielen als Männer (Autor ist ein Mann)]

23. Killing Games: A Look At German Videogame Legislation, by Bernd Kreimeier (www.gamasutra.com) (hier wird unter anderem die handhabung des deutschen Index anhand Wolfenstein und Mortal Kombat erläutert)

24. How I Spent my Spring Break: A Report on the 2000 Game Developers Conference, by François-Dominic Laramée (www.gamedev.net)

25. Petersons Three Priciples of Game Design (www.gamedev.net)

26. Linux as a Gaming Platform, by Mark Collins (www.gamedev.net)

27. Heretic: What went right, What went wrong (www.gamasutra.com) (Facts rund um die Möglichkeiten von Code Licensing, und wie man in der Zeit bleibt)

28. id Software: Technology License Program (www.idsoftware.com)

29. Postmortem: Epic Software's Unreal Tournament, by Brandon Reinhart (www.gamasutra.com) (Tiefe Einblicke in die Engine, deren Entstehungsgeschichte, deren Architektur)

30. Prof. Dr. A. Min Tjoa: Softwareprojektmanagement (ergänzende Definitionen in Sachen Software Engineering)

31. Design and Architecture of a Portable and Extensible Multiplayer 3D Game Engine by Markus Hadwiger

32. Game Design: Secrets of the Sages by Brady Books, ISBN 1-56686-904-8

33. Future of the andventure game, by Tasos Kaiafas (www.gamespot.com)

34. Future of Video Games Goes Mobile, by Martyn Williams, IDG News Service Friday, March 30, 2001

Image Index