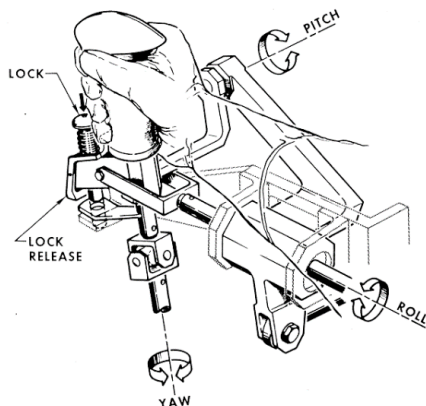


## A Guide for New GarageGames Customers Version 1.0



### Welcome to GarageGames

The process of building games, especially games that are more than just entertainment, is difficult and rewarding; this guide will help you understand our practices. For someone who has never developed a game, it's often a humbling experience that helps you to appreciate the care, coordination, and magic of game development. The information in this document is a guide and no single approach solves every problem; occasionally, we may deviate from the prescribed best practices.

Welcome to GarageGames..... 1

About this Manual ..... 4

Introduction - Roles..... 5

Introduction – Making Fun..... 7

Introduction – Software Evolution..... 9

Introduction – Development Lifecycle..... 10

Introduction – Synchronizing Teams ..... 12

Section: Game Design Document ..... 15

Game Design Document – Introduction ..... 16

Game Design Document – User Experience ..... 18

Game Design Document – Style Guide..... 19

Game Design Document – Script..... 20

Game Design Document – Glossary ..... 21

Section: Technical Design Document..... 22

Technical Design Document – Introduction ..... 23

Technical Design Document – Hardware/Software Specification ..... 24

Technical Design Document – Performance Specifications..... 25

Technical Design Document – Architecture..... 26

Technical Design Document – Interface API/Protocols ..... 28

Technical Design Document – Release Requirements..... 29

Section: Estimate Sheet..... 30

Estimate Sheet..... 31

Estimate Sheet - Goals ..... 32

Section: Project Schedule ..... 35

Project Schedule – Understanding Constraints ..... 36

Section: Execution ..... 39

Execution - Intro..... 40

Execution – Cadence..... 41

Execution – Vertical Slices..... 42

Conclusion ..... 43

# About this Manual

In this manual, we will cover the topics below. If this is your first time working with GarageGames, we recommend that you share this document with everyone in your company. It will save time, money, and it will contribute to a healthy team environment where groups collaborate better and accomplish more.

- **Intro**
- **The Game Design Document**
- **The Technical Design Document**
- **The Estimate Sheet**
- **The Project Schedule**
- **Execution**



**NOTICE**

All team members should read each section. While the technical design document may be the most important document to a technical person, the other sections are still very important. It takes a team to build a game so understanding the context of the other groups and individuals will help you do your job.

## Introduction - Roles

Congratulations on your purchase of a custom game. Procuring a game is exciting but it may also be intimidating if you have never been part of the process before. Making a game is hard because it takes many disciplines. The following roles are required to make a game:

- **Designer**  
Designers define the user experience, develop gameplay strategies, and balance competition.
- **Artist**  
Artists create the digital assets that define game aesthetics.
- **Programmer**  
Programmers use scripting or programming languages to add logic and systems to a game.
- **Manager**  
Managers coordinate the individuals. Producers, Art Directors, Project Managers, and Technical Directors are all important managers. A lead is another type of manager who is the best of peers in a given domain (i.e. Programming Lead). It's important for all managers to be in touch with their project. We suggest that managers spend at least 30% of their time getting their 'hands dirty'. Leads should only manage 20% or less.



### WARNING

After developing a game, teams often do a post-mortem (some call them post-partum). Typically, team communication is the number one topic cited by team members as the biggest area for improvement.



### WARNING

It's the designer's job to define the game characteristics. The designer uses past experience and opinions to set design and it's important to let the designer do their job. Typically, it's better to keep the design team size to a minimum, too many opinions water down a design and reduce the 'je ne sais quoi'. It is, however, useful for a designer to realize that constraints, such as technical, schedule, resource, force a designer to make decisions and be critical of their design.

### Introduction – Making Fun

Making fun separates game development from other forms of software development. To make **software**, you will need to achieve the following:

- **Intuitive User Experience**
- **Intuitive Controls**
- **Pleasing Aesthetics**
- **Useful Functionality**
- **Stability**
- **Maintainability**

In addition to the list, a **game** also needs the following:

- **Fun**
- **Replayability**
- **Joy Moments**



#### WARNING

Design changes should be well understood as soon as possible. Changing design during execution without careful consideration is a recipe for disaster.

To create fun you should consider one of two paths: **cloning** or **iterative design**.

- **Cloning**

Cloning means taking a proven design and copying it.

- **Iterative Design**

Iterative design requires developing prototypes (either on paper or with a prototyping application) and usability testing. During usability testing, we determine if what was built is fun. If it's not, you should prototype and try again.



### WARNING

Design is constrained by technical, schedule, and resource constraints. If the goal of your game is to be persuasive there is yet another level of challenge. Persuasive games have the challenges of 1) Software development 2) Game development 3) Instructionally correct. **That's a triple whammy!**



### Introduction – Software Evolution

Software development isn't just iterative, it's evolutionary. The fittest designs and practices survive and the bad ones must be retired. At a coarse granularity (i.e. the 'forest view'), we introduce large cycles to help facilitate software evolution. The common phases include:

- **Proof of Concept**  
Demonstrates that an idea is possible.
- **Prototype**  
Demonstrates that an idea is feasible.
- **Working Prototype**  
Demonstrates that an idea is feasible in a real world environment.
- **Rapid Development**  
Similar to a proof of concept with the exception that it occurs during development.
- **Development**  
Typical development during a software project.
- **Alpha**  
Feature complete. Not ready for 3<sup>rd</sup> party testing.
- **Beta**  
Feature complete. Ready for 3<sup>rd</sup> party testing.
- **Release Candidate**  
A candidate for Gold Master.
- **Gold Master**  
A shipped version of the software.

### Introduction – Development Lifecycle

There are stages a game development project must follow in order to complete a game. All games follow these stages to varying degrees. If the team isn't able to recite the goals, problems are bound to arise.

- **Vision**  
The goals of the project. It is extremely important to communicate goals, preferably in priority order, to all team members.
- **Pre-Production**  
During pre-production we assemble the team and set the standards for goals, design, and schedule. We document them in order to facilitate sharing.
- **Production**  
During production we execute on the game design and react appropriately to inevitable changes.
- **Release**  
Release is the end of production. Feature development is complete; final QA and certification commences.
- **Support**  
Once live, the game must be supported. In live development models, the base game is released while the other stages, listed above, occur in parallel with support.
- **Decommission**  
The game is retired from play.

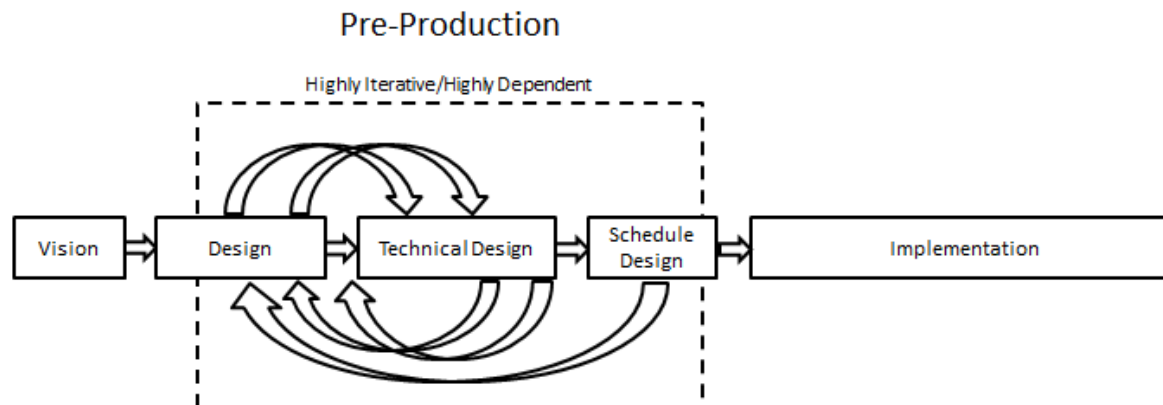
**WARNING**

The amount of time it takes to complete pre-production can be deceiving.

Typically, the amount of work (a measure of efficiently completed effort) often occurs over a long duration (amount of calendar time). The reason are the dependencies and iterations. Later decisions, such as technical implementation decisions, are gated by earlier decisions like design and vision. These dependencies create a development process that is inherently serial.

Conversely, later constraints, such as technical implementation and schedule, can affect earlier decisions such as design. These changes result in another iteration that is still inherently serial.

While this process may be slow, it would be significantly more time consuming if these designs were implemented instead of simply simulating them on paper in a game design document. By setting aside appropriate time for pre-production, we can reduce dependencies and iteration and increase parallelism during implementation.



*Figure 1 Preproduction - Both highly iterative and highly interdependent.*

### Introduction – Synchronizing Teams

You are now familiar with the roles, you understand the challenge of designing fun, and we've presented the high level stages of the game development lifecycle. Now it's time to get one step closer to execution. But there's still one more important step – synchronizing teams.

Communication across teams is vital and very difficult. The first challenge is to create a common, agreed upon set of guidelines. The second challenge is to actually get people to abide by them. We use several documents to synchronize our teams.

- **Intent Document**  
Defines the base design in one page. It's the pitch document before you write the Game Design Document.
- **Game Design Document**  
Defines the game design in greater detail. The most important role of the game design document is to tell the story of the user experience. The game design document tells us about game function.
- **Technical Design Document**  
The technical design document describes how to implement the functionality of the game design document. Typically, the technical design document describes runtime requirements, architecture, and development standards.
- **Estimate Sheet**  
The estimate sheet decomposes the implementation into smaller units for the purpose of making estimates. The estimate sheet includes all production stages such as development time, QA, usability, refactoring, and release.
- **Project Schedule**  
The project schedule takes the estimates from the estimate sheet and assigns resources to them. The project schedule gives us our milestone dates.



### WARNING

Using a document as an 'authoritative source' can be a great way to synchronize your team, but simply making the document will not achieve the goal of a synchronized team. There are several guidelines you should follow:

- **Single Owner**

While many people will contribute to the documents, there should be one owner of the document. It's everyone's responsibility to understand and contribute to the document, but a single owner should merge together everyone's comments and notify the group when a new revision of the document is ready for review.

- **Assumptions**

The team should recognize that the design document is the authoritative source. If something isn't specified, it should be considered ambiguous. If something is really important to you and the documents don't include the information you need to represent your important topic, ask the document owner to update it.

- **Updating**

You won't get the design 100% correct during pre-production. That means things are going to change. When they do, the document owner should update it and share it with the team. Typically, the document owner will update the document immediately and share a new version of the document on a weekly basis, typically in a project meeting or once a week during the sprint.



### PRO TIP

These documents are for collaboration and communication. You should always include page numbers and sections using a numbering system. It's much quicker to refer to a spec if it's numbered appropriately (i.e. "John, did you take a look at the design specification in section 2.3.4?").

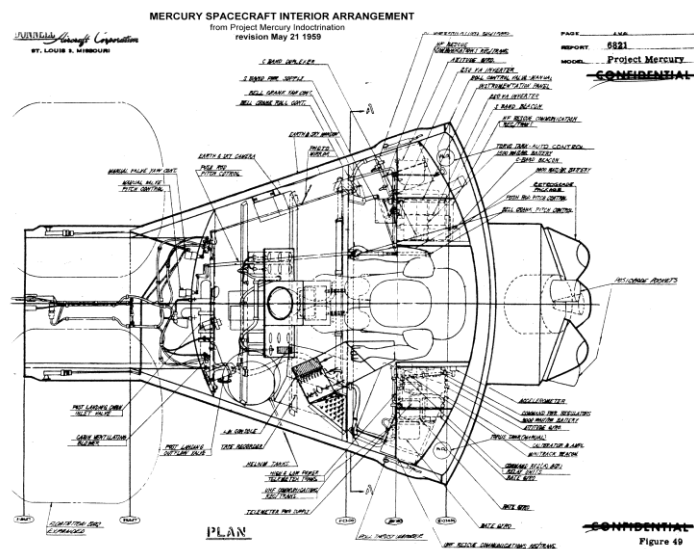


### PRO TIP

Have you gained a new team member? It's no big deal. Ask the new team member to read the GDD and TDD and come back with questions. Consider giving them a test. Even if you don't grade it, the very process of testing someone increases the likelihood that they will retain the information.

# Section: Game Design Document

## Setting the Course for Your New Game



### The GDD

The game design document (GDD) is a document designed for failing. No, really. By iterating in a document, one of our goals is to fail early. Failing on paper is much cheaper than failing in implementation, QA, or worse -with the end user.

### Game Design Document – Introduction

The GDD defines **what** you are building. It doesn't, and shouldn't, concern itself with the **how** and **when** of a project. At a minimum, the design document will define user experiences, style guidelines, and script/story.



#### WARNING

The design document is more like an initial rendering of a home, not the blue print. By focusing only on the end-result you are removing constraints that may overly limit your creativity. But there is a balance. It's not helpful to design a game that would take three years to build when you know you only have six months. Good designers understand that some things that seem difficult to implement can actually be very easy and some things that seem easy to implement can be very hard. Keep an open mind and talk with your technical resources often. Don't negotiate against yourself and assume that something is hard when it may be simple. Conversely, don't make assumptions about something you perceive as simple – it may not be.





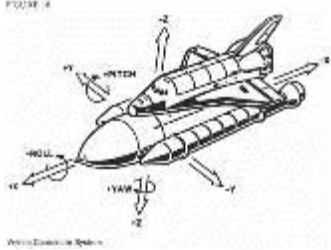
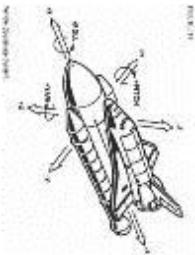


### PRO TIP

Move the rock! The most difficult part of a new idea is getting started. Don't be afraid to be wrong. Move the rock by putting down the first proposal and sharing it with someone else or the team. Don't get worried about formatting at this stage; you should assume that you will be removing or refining a lot of your ideas. Even if you share something that's 95% wrong, you are still 5% right and your next step will be clearer.

Game Design Document – User Experience

The user experience provides the basis for development of the entire game. The user experience uses wireframes, sketches, and descriptions to define the user’s inputs and the game’s outputs. We typically use a format similar to the following:

	The user is in the shuttle. She pulls the yoke towards her stomach.
	The ship rotates along the y-axis
End Of Experience	

# Game Design Document – Style Guide

The style guide defines the look and feel of the application or game. If the customer doesn't know what look-and-feel they want, we will provide them with a series of choices based on popular designs. Once we determine a reference from existing samples, we will develop custom concept samples using initial designs.

In addition to the style guide, we may build common art during the pre-production phase. Developing common art early in development helps us maintain a consistent look across all experiences and increases parallel development.

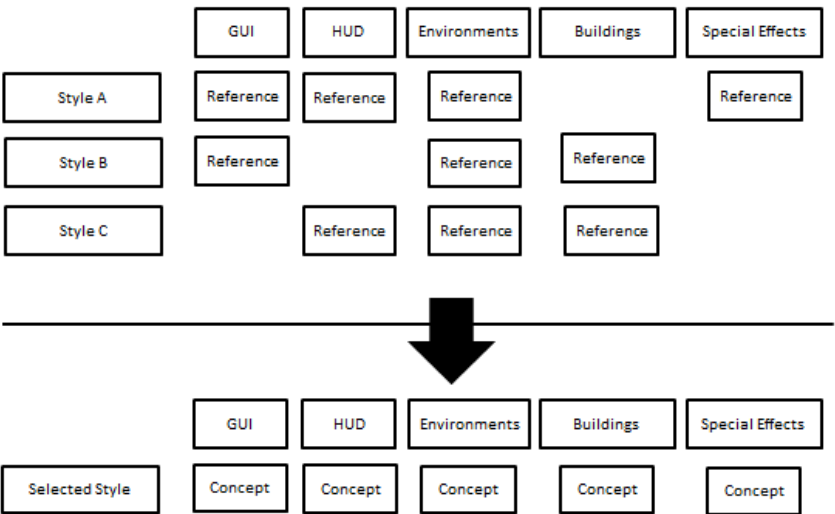


Figure 2 Typical steps of moving from reference examples to custom developed concept pieces.

# Game Design Document – Script

The script may exist in several forms. For a class, the script may define the content you intend to teach. In an MMO, the script may include a list of all quests available by player level. The script defines all of the content the user may encounter. To save development cost, game developers may only generate one user experience that defines the template for a user experience. Once that is defined, the designer can define multiple implementations of the user experience in the script.

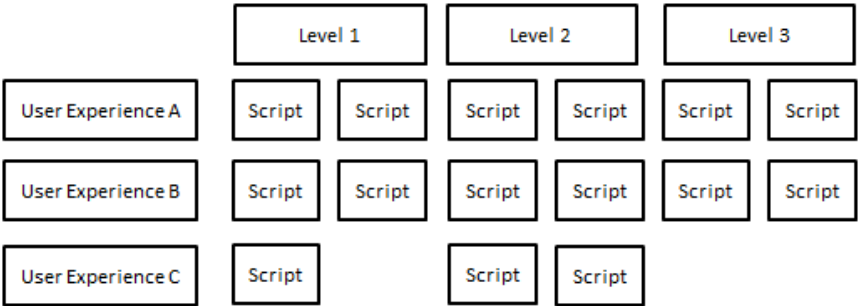
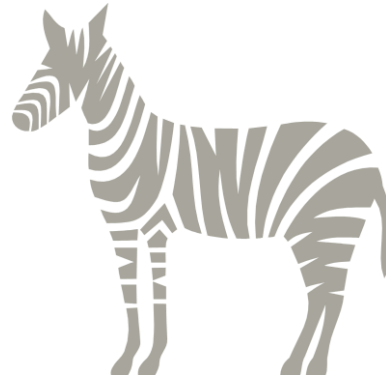


Figure 3 Potential layout of levels, users experiences, and scripts.

### Game Design Document – Glossary

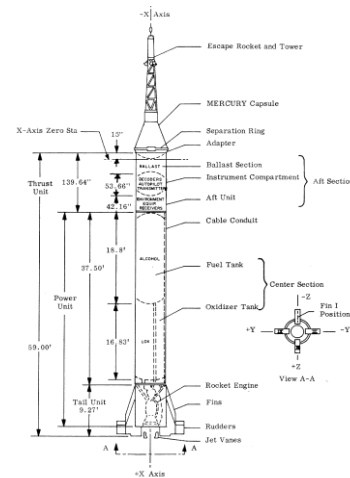
Terminology seems like a trivial issue, but when it comes to communication, it's very helpful to get everyone on the same page. There are many different disciplines in the game world and each group already has a set of vocabulary within their given domain. When you hear dissenting terms, pick one and add it to the glossary.



*Figure 4 The GDD calls this a zebra, the TDD calls it a stripy horse. In the meeting, everyone is confused.*

## Section: Technical Design Document

How getting done is done



### The TDD

The technical design document defines how we will implement your game design.

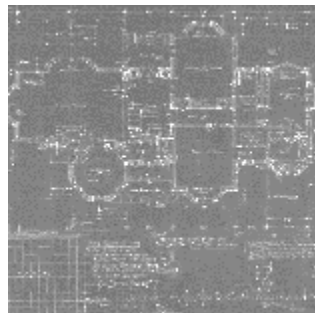
## Technical Design Document – Introduction

The TDD defines **how** we are building what the GDD defines. It doesn't, and shouldn't, concern itself with game design and schedule other than to reference the game design document or project schedule. At a minimum, the technical design document will cover hardware/software specifications, performance specifications, define architecture, implement interface/API specifications, and release requirements.



### WARNING

The technical design document is more like the blueprints of a home, not the rendering. It should be specific and detailed enough that developers will be able to reference it to implement the project.



# Technical Design Document – Hardware/Software Specification

The technical design document will tell developers and QA what platforms and hardware systems the software will work with. At a minimum, this specification should cover the runtime environment, but it may also be important to define the specifications of the software you will be using to develop the product.

The following is a list of possible software modules you may need to formally define.

Domain	Module
End-User Configuration	<ul style="list-style-type: none"><li>▪ Min System Ram</li><li>▪ Available Mass Storage</li><li>▪ Graphics Card Shader Model</li><li>▪ CPU Benchmark Performance</li><li>▪ Form Factor (Laptop, iPad, Galaxy, etc.)</li><li>▪ Min Network Bandwidth</li><li>▪ Min Network Latency</li></ul>
Operating System	<ul style="list-style-type: none"><li>▪ Type (Windows, Mac, iOS, Android, etc.)</li><li>▪ Version – Release, Update, Service Pack, etc.</li></ul>
Browser Support	<ul style="list-style-type: none"><li>▪ Brand (Chrome, IE, Firefox, etc.)</li><li>▪ Version</li></ul>
Standard	<ul style="list-style-type: none"><li>▪ Protocol (HTML5, OMQ, XML, SCORM, DIS, MP4, etc.)</li><li>▪ Version</li></ul>
Tools/Framework	<ul style="list-style-type: none"><li>▪ Product (Torque, Flash, Zend, 3D Studio Max, etc.)</li><li>▪ Version</li></ul>



### Technical Design Document – Performance Specifications

Scale can have considerable impact on the development practices and, therefore, the cost of software development. Both parties should not make assumptions regarding the load that the system will be able to handle. This is especially true for servers or services.

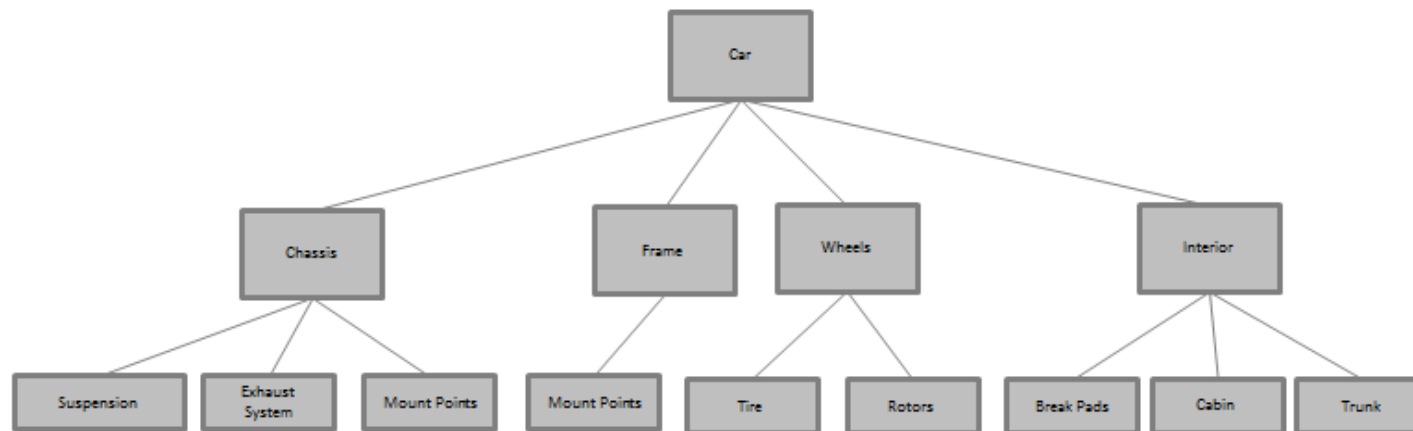
At a minimum, you should consider the following measures:

- Max Concurrent Users
- Average User Throughput
- Max User Throughput
- Max Payload (bandwidth)
- Max User Wait Time (download, processing, etc.)

## Technical Design Document – Architecture

Architecture may start with some concept of a design paradigm like MVC, MVVM, etc. You could use this paradigm to define the architecture. Finer granularity in architecture may be defined using design patterns such as those defined by the gang of four Design Patterns book (<http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612>).

One way to start defining your architecture is to simply decompose the product into anticipated modules. Similar to a Work Breakdown Structure (WBS), you can use the 100% rule to decompose the product into smaller pieces. After decomposing the product, you can then look for opportunities for shared systems such as managers and utilities.





### WARNING

Architecture is very difficult. Trying to design too much up front can lead to disastrous results. Instead, let your architecture evolve by building your product in steps based on what the product should do. For example, if you have never seen a car before, it would be too difficult to sit down and design a car. Instead, it would be better to evolve your architecture over time in phases.

Don't design a car; instead, break your project into phases:

- A rolly thing
- A rolly thing that can slow down.
- A rolly thing that can slow down, steer, and is controlled by a person.
- A rolly thing that can slow down, steer, accelerate, and is controlled by a person.
- A car.



### WARNING

Building a system that is too rigid is bad. Things will need to evolve over time so your system needs to be flexible enough to adapt. Surprisingly, making your system too flexible can also be bad. When systems are engineered to be too flexible, doing very simple things can become very complex due to all the systems required to couple systems together (if your system is too complex, a solution is to build a tool that makes the process easier). And if those systems couple together at runtime, debugging can become very difficult. Balance is important. The best architectures are both flexible and simple.

### Technical Design Document – Interface API/Protocols

Most applications have GUIs. It's a lot of work to define user interfaces that are intuitive and well organized. Applications also interface with other applications; when they do, they use interfaces such as Application Programming Interfaces (APIs) and network protocols.

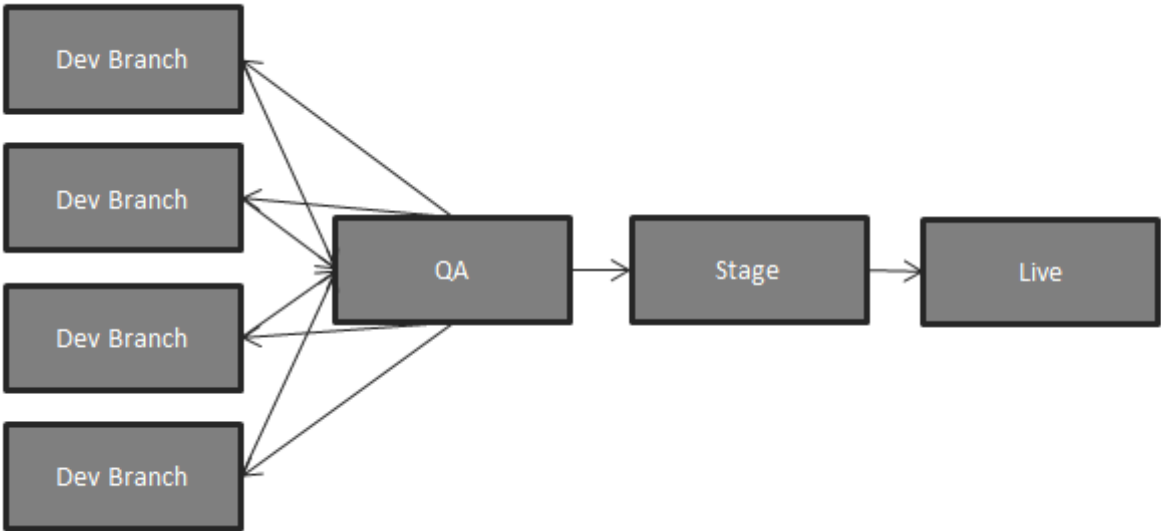
It's possible and, in many ways, beneficial to design your interfaces prior to developing them. Too often, APIs are a result of the underlying foundation only, a development practice that tends to result in non-intuitive APIs.

At a minimum, API specifications should have the following definitions:

- **Startup**
- **Shutdown**
- **Disconnect (if applicable)**
- **Reconnect (if applicable)**
- **Functions**
- **Parameters**
- **Data structures and data alignment**
- **Examples**
- **Error Handling**

# Technical Design Document – Release Requirements

Finally, the Technical Design Document should define how software is released, updated, and if applicable, decommissioned. There are many ways to update an application. Some, like the distribution of a new installer, are simple; others, like a live update of a product while users are still using the application, are much more difficult and require special consideration. A common update system resembles the following diagram. Under this system, product owners are able to deploy releases with reasonable assurance that a bug or similar artifact isn't created by the versioning system itself.



## Section: Estimate Sheet

How much work will this take?



### Estimate Sheet

You understand the goals, the intended design, and the likely implementation – it's now time to determine the work effort of tasks.

# Estimate Sheet

Armed with our goals, design, and technical design document, we are ready to start thinking from a production perspective (as opposed to simply thinking of decomposition of the product modules) and develop tasking.

Let’s first take a look at what a section of our estimate sheet might look like:

Task	Estimate (h)	Confidence	Risk Adj. Est.	Role	Estimate By	Assumed Resource	Comments
Phase I							
MS 1 - Build A Rolly Thing							
Acquire Metal	32	8	39.1	PROC	Joe	John	Assumes parts are available
Forge Parts	8	7	10.7	FAB	Joe	John	
Assemble Parts	16	7	21.3	BUILD	John	John	Assumes method in TDD §3.4
Release							
QA	16	10	16.0	QA	Mike	Mike	
QA Reaction	16	8	19.6	FAB, BUILD	Joe	Joe	
Usability	4	10	4.0	BUILD	Joe	Joe	
Customer Review	4	10	4.0	CUST	Joe	Joe	
Management	19.2	10	19.2	MGR	Joe	Joe	
MS 2 - Build A Rolly Thing w/ Breaks							
Refactor	16	10	16.0	BLKSMTH, BUILD			

# Estimate Sheet - Goals

There are several goals of the estimate sheet:

- **Provides estimate**  
The obvious reason for estimating is to provide estimates for the project schedule.
- **Additional Decomposition**  
Drives finer-grained decomposition of the architecture. This is why we tend to estimate in hours instead of days. Estimates over two days are reviewed as candidates for further decomposition.
- **Shifts thinking from “what” to “how”**  
After spending so much time thinking about what you are building, it’s important to add the production side of the process to your mindset.
- **Defines production overhead**  
Management, QA, usability, review, and refactor are all very important parts of development that can often be missed.
- **Estimates help define a given implementation**  
Often, the estimate can provide a level of certainty for a given definition where there may be ambiguity.
- **Quantifies risk through confidence**  
Confidence helps us understand risk to some degree. A two week job estimate that I’ve done before (high confidence) is very different than a two week job I’ve never done before (low confidence).





### PRO TIP

Don't be surprised when the estimate process forces revisions of design or implementation. Track the changes in the TDD and GDD so that the new approach can be shared with the remaining team members.



### WARNING

Warning, estimates change when you switch from one developer to another. Add the assumed resource to add validity to the estimate.



### WARNING

Estimates should be in work effort, not duration. Work effort defines how many hours the work takes to complete, not how long it takes on a calendar. Consider a two hour job that starts at the end of the day Friday and finishes early Monday morning. The work effort is two hours, but the duration is forty-one hours.



PRO TIP

GarageGames Confidence Scale:

1. No clue -- don't make decisions on this. We need to talk more.
2. Hardly a clue -- don't make decisions on this. We need to talk more.
3. Someone else has done this and I read about it; job not well defined -- don't make decisions on this. We need to talk more.
4. Someone else has done this and I understand this; job might not be well-defined -- don't make decisions on this. We need to talk more.
5. Done something similar to this before and this relates to that work -- this estimate has a variance of +/- 50 percent of estimate.
6. I think I understand this fairly well and I understand the goal.
7. The average case for programming when the requirements are understood.
8. A confident case for programming; average case for a lot of art work.
9. I've done this before and it's very similar.
10. I'm probably not working on software... or... no matter what, I'm only going to work on this for a period of time (i.e. 1 week of QA).



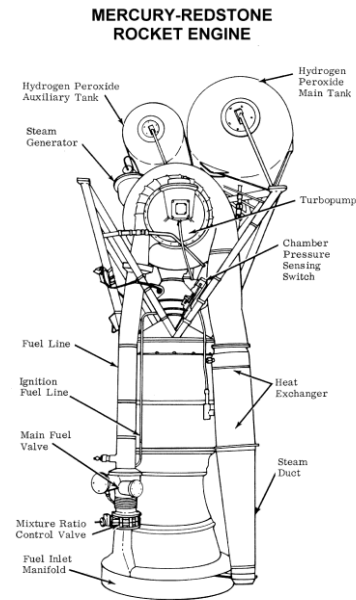
PRO TIP

It isn't a perfect science, but at GarageGames we use the following formula to calculate risk adjusted time:

$$\text{Risk Adjusted Estimate} = \text{Estimate} + (\text{Estimate} * (10 - \text{Confidence}) / 9)$$

### Section: Project Schedule

When will this Game be done?



#### Project Schedule

You have a solid estimate sheet and you feel confident about your numbers. Next, it's time to let the schedule dictate when the project will be done.

### Project Schedule – Understanding Constraints

Moving from the estimate sheet to the project schedule is fairly straightforward. To build the schedule we copy and paste directly from excel to Microsoft Project. In Project, we consider the following constraints in order to calculate milestone and project end dates:

- **Dependencies**

A dependency prevents a job from starting because it is dependent on another job. A classic example is that QA can't begin its review until the software is feature complete (unless you are OK with the inefficiency of doing it again).

- **Critical Path**

A critical path exists when the date of a given milestone or release date is wholly dependent on a single resource. We can reduce critical paths by balancing workload across multiple resources. Critical paths that aren't resolved need to be managed closely since their impact on a release date can be profound.

- **Over-allocations**

Workers' time should not be allocated greater than 40 hours. If you find that you need more than 40 hours from any single employee, your project has systemic issues that need to be resolved with more money, more time, or a reduction in features.

- **Work Calendar & Vacations**

Holidays and vacations can have a profound effect on release dates. They should be added to the project schedule.

- **Work Balancing**

Balancing work across employees increases velocity. Conversely, working in parallel increases management overhead.

- **Hiring Needs**

Your project schedule is a great tool to help you assess your hiring needs. It's not unusual to spend two or three months trying to find the right hire. If you don't know well in advance, you will end up compromising your needs and make a bad hire.



### WARNING

Remember, estimates are estimates and many things will change over time. It's important to update your project schedule weekly if you want it to be useful for the entire life of your project.



### WARNING

Don't get too granular or expect too much fidelity from your project schedule. It's a tool to use for guidance but it's not perfect. Don't be upset if someone is doing a task on Monday that is scheduled for Friday (or vice versa). We estimate in hours but we look at the project schedule in granularities of weeks not days.



### WARNING

The milestone and release dates are a function of feature count, complexity/risk, and parallel development and typical development constraints. Until the project schedule is built, milestone dates and releases are guesses. After the project schedule is built, milestone and release dates are educated guesses.



### WARNING

It's important to get team buy-in on the estimates. Your estimates aren't nearly as important as the people who will live up to the commitment. If you want your developers to truly commit to their work load, you need to get their buy in.

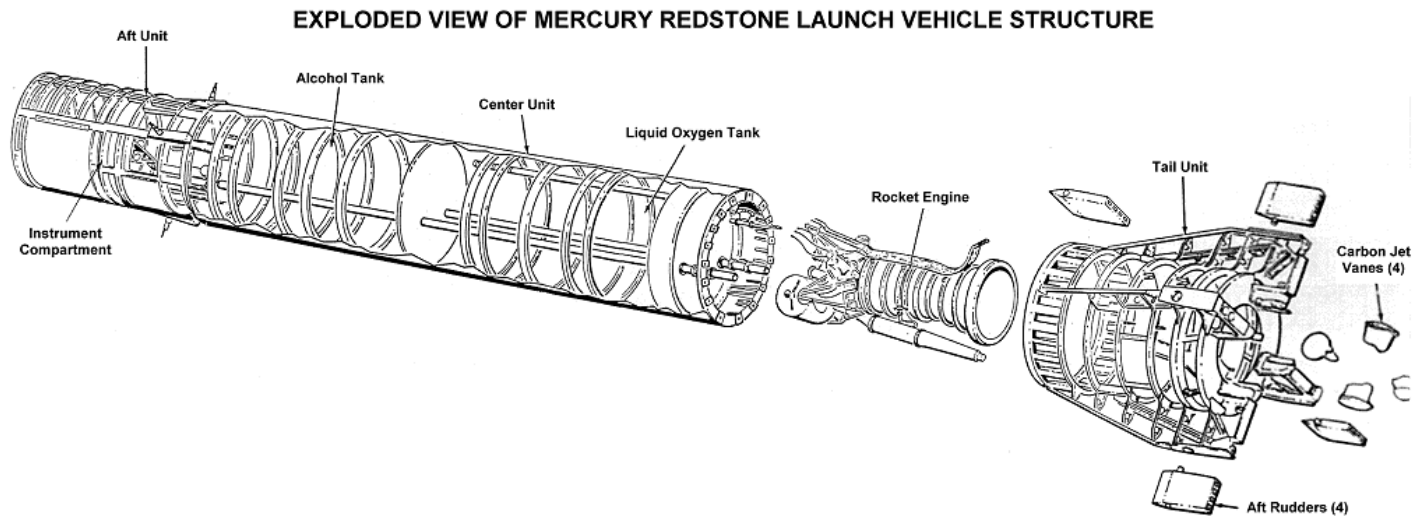


### WARNING

Generally, you should place higher risk tasks earlier in the project schedule. By doing this, you will have more time to react to changes discovered when you actually implement the task in question.

## Section: Execution

Now we discuss the hard part



### Execution

Congratulations, now comes the hard part. Execution.

### Execution - Intro

Executing your project, commonly called production, is an extremely complex process that requires highly-talented and seasoned professionals. This document will scratch the surface of execution by discussing some of our best practices.



#### PRO TIP

At GarageGames we use a ticket tracking system to task those who are doing the work. Sharing a project schedule is not granular enough. We use this same ticket system for tracking bugs. Customers who want to put their finger on the pulse of the project are welcome to review the tickets at any time.



## Execution – Cadence

So far, we've not discussed agile methods. We use the aforementioned process to help us determine our path for a reasonable horizon. We use agile methods to implement that path. The agile methods we use include:

- **Weekly Sprints**

Once a week, we use the project schedule and backlog (a place to put discovered and incomplete work) to build a project sprint. At the beginning of a sprint we commit to the work we will perform. The goal is to complete your work by the end of the sprint.

- **Daily Standups**

Every day we meet as a team to discuss what we did yesterday and we are doing today. The goal of this meeting is to meet no longer than you need to in order to get your work done.



### WARNING

Don't nag developers during the sprint. If you've done it right, they are committed to deliver on their promises. Measure their performance at the end of the sprint but don't make the sprint so long that your developers fall victim to procrastination.



### WARNING

Be considerate during the daily standup. Your goal is to inform others so that you can work out interdependencies between the team. It's not a time to argue implementation. Handle those discussions after the meeting.

## Execution – Vertical Slices

In order to work iteratively and reduce risk, we like to begin development by building a vertical slice. A vertical slice is a representative set of work done at final quality. This will help discover issues early in a project when there is still time to make adjustments.



### PRO TIP

After developing a vertical slice, usability testing is advised. To test usability, we put new users in front of the product and ask them to use it. If they can't use it without additional information, your product will not thrive in the real market. As game developers, we are too experienced to represent a typical user; therefore, we must find testers that are more representative of the game's audience.

# Conclusion

## Thank You!

In closing, we'd like to thank you again for choosing GarageGames for your game development service needs. The success of our project together will hinge on our ability to collaborate effectively.

We hope that this guide will provide you with insight into the challenges of game development and the methods we've developed to manage the challenge. We encourage you to share this document with everyone who we might be working with.

We would like to thank NASA for the use of the images. <http://history.nasa.gov/diagrams/mercury.html>

If you have any questions, please feel free to reach out directly to your point of contact or our CEO. He can be reached at [ericp@garagegames.com](mailto:ericp@garagegames.com).