Software Project Plan

Introduction

Project Scope

GameForge is a graphical tool used to aid in the design and creation of video games. A user with limited Microsoft DirectX and/or Visual C++ programming knowledge will be able to construct a basic 2D-arcade game. The idea is to limit the amount of actual code written by the user. It will also assist experienced programmers in generating the Microsoft DirectX and Microsoft Windows9x overhead necessary for basic game construction, allowing them to concentrate on more detailed game design issues and implementation.

Critique: Bounding is a critical element of the project scope and the project plan. It would be a good idea to try to "bound" all the general statement of scope noted here. For example, “a basic 2D arcade game” is open to very broad interpretation. What is basic to one reader might be unacceptable to another.

The software will consist of a number of inputs, graphically assisting the user in creating on-screen objects including the following:

- User Created Objects (player character, creatures, static objects)
  - Bitmaps (with animation)
  - Collision Detection Areas
  - Movement Routines
  - Additional Object Attributes
- Backgrounds
- Input Device Setup
- Sound Events

The software will also consist of a number of graphical processing functionalities including the following:

- Defining/Editing Objects (including characteristics)
- Object Positioning
- Opening/Closing/Saving Game Project Files
- Exporting Game Projects to compilable C++ Files

Outputs include:

- User Created Sprite Objects
- Bitmaps
- Microsoft VC++ (with DirectX code) Files
- Game Project Files
- Text Files (containing sprite attributes)

- Database Files

Comment: The author have done a good job of providing the reader with a conceptual model of the information transform that is to occur.

Major Software Functions

Process and Control Functions

- VB interface – The interface is the subsystem the user interacts with. It creates a project space for all project files to be stored in. It gathers all necessary data from the user, as well as interacting with the access databases. The interface then generates data files containing all specifications of all the sprites, as well as input device information and sound information. All necessary files such as .wav files and .bmp files are moved to the project directory. This subsystem contains the screen representing the game and a list of all sprites and their attributes.

Critique: A fair amount of application specific jargon is introduced here without definition. Might be a good idea to refer the reader to a glossary or provide a brief definition as footnotes.

- C++ engine – This subsystem contains the main function of the system. The engine creates a .cpp file for the game. The file contains references to the data files generated by the user interface and references to DirectX code contained in custom header files.

User Interface Processing

- Input Wizards – There are a number of wizards provided to guide the novice user through the necessary steps for game development. They range from sprite generation, to game logic, to input devices. The wizards interact directly with the user interface.

- Level Editor – This is the main interface, and displays a graphical representation of the game/level a user is designing. A tree-view of all created objects is also represented here. All wizards and other functions can be accessed from this interface.

- Help/Tutorial Files – These files include a wide range of help topics, including FAQ's, Tutorial, detailed descriptions of objects and VC++ code, and a search engine to find needed information.

Input Processing

- Databases – GameForge utilizes a Microsoft Access database to store sound libraries and image libraries, as well as pre-designed sprites. The databases are accessed by the user interface.

Output Processing

- Data files – files containing information specified by the user that are read by the C++ code. The files are generated by the user interface (information is taken from the resulting database). The user's game can be tweaked by editing these files rather than rewriting and recompiling the C++ code.

- GameForge Files (.gmf) – Files are stored with a unique extension used exclusively by the GameForge system. These files are similar to .cpp files but will not be compilable. They are intended as temporary storage during game creation. They are generated by the user interface.

- VC++ Files (.cpp) – Finished projects can be saved as .cpp files that can be compiled with Microsoft's Visual C++ compiler to create an executable file for the game. The VC++ engine runs these files.

Performance/Behavior Issues

GameForge is designed to be compatible with the Microsoft Windows 9x operating system. Microsoft Windows NT 4.0 and earlier versions will not be supported (Windows NT only supports Microsoft DirectX up to version 3.0. DirectInput had not been implemented at this time, making this version of DirectX very limited.) Microsoft Windows 2000 should also be compatible.

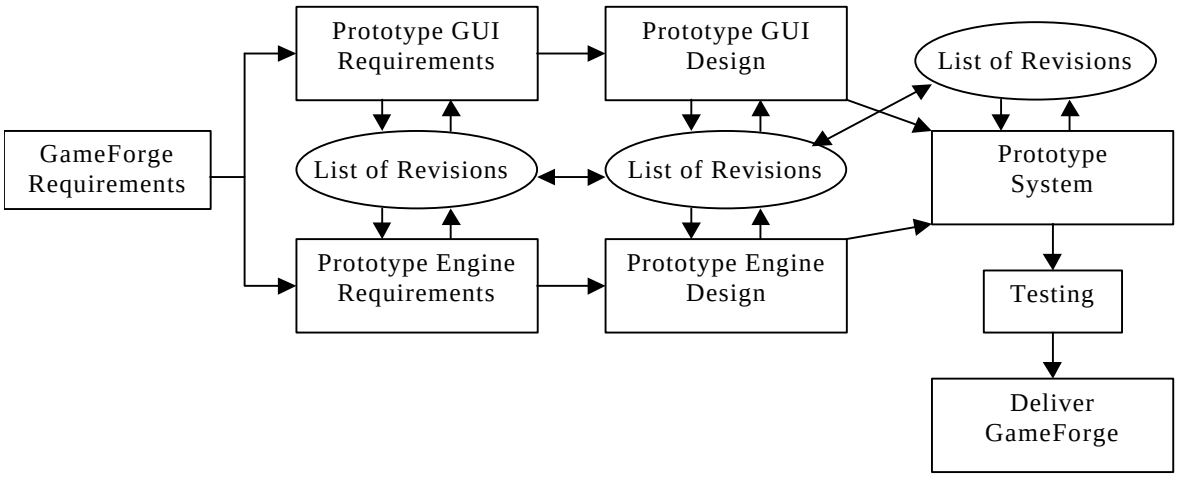
GameForge also requires Microsoft DirectX 7.0 or above. Users may also want to obtain the DirectX 7.0 SDK if they plan on expanding the GameForge library files beyond their original scope.

GameForge also requires the Microsoft Visual C++ 6.0 compiler. GameForge's VC++ code may be compilable using Borland or some other VC++ compiler, but functionality is not guaranteed.

Management and Technical Constraints

GameForge has a drop-dead delivery date of 04/17/00.

PA Software will be using the Rapid Prototyping model during design and implementation:



Comment: The above diagram presents a useful overview of the project approach. It does not replace a detailed timeline schedule, but it does provide a "quick look" at what the team will be doing.

Project Estimates

Historical Data Used for Estimates

A reference Function Point metric was calculated using Function Points calculated from previous projects (namely, Demon Tree from CIS 490a and Function Point Calculator from CIS 375.)

Reference Function Point Calculations:

Demon Tree FP: 121.03
Demon Tree Person Months: 2.5

Function Point Calculator FP: 83.74
Function Point Calculator Person Months: 1.5

Reference Estimated Person Months:

Average Function Point per Person Month: 52.119

Critique: Given the situation, the above computation is acceptable. However, it is important to note that the sample for averaging is too small to be meaningful. In the real world, the average should be computed using at least 5 to 10 projects in the same application domain.

Estimation Techniques Applied and Results

The following is a breakdown of the numbers used in estimating the Function Point for GameForge:

Estimation Technique: **Function Point**

Interface	Simple	Average	Complex
Number of User Inputs	12	3	4
Number of User Outputs	8	5	2
Number of User Inquiries	10	3	
Number of Files	2	3	1
Number of External Interfaces		1	

14-Point Questionnaire: **34**

Engine	Simple	Average	Complex
Number of User Inputs		4	2
Number of User Outputs			1
Number of User Inquiries	15		
Number of Files	6	3	10

Number of External Interfaces	1		1

14-Point Questionnaire: **42**

Estimate for: **Function Point**

Based on the estimations from the previous section, and dividing by the time estimate from previous projects, we can calculate a duration estimate for GameForge:

Interface: 245.98
Engine: 339.19
Total Function Points: **585.17**
GameForge est. Person Months: **11.23**

LOC = FP*30
GameForge est. Lines of Code: **17,555**

Estimation Technique: **Constructive Cost Model (CoCoMo)**

The CoCoMo model was also used to verify the estimate calculated by using the Function Point metric.

GameForge assumes itself to be an Intermediate, Semi-Detached software project.

$$\text{Effort} = a\,(\text{KLOC})^{b}$$
$$\text{Duration} = c\,(\text{Effort})^{d}$$

Equation values for Effort calculation:

a = 3.0
b = 1.12

Equation values for Duration calculation:

c = 2.5
d = 0.35

Estimate for: **Constructive Cost Model (CoCoMo)**

$$\text{Effort} = 3.0\,(17.5)^{1.12} = \mathbf{74.016}$$
$$\text{Duration} = 2.5\,(74.016)^{0.35} = \mathbf{11.277}$$

Reconciled Estimate

Effort (in Function Points) Estimate:
Total Function Points: **585.17**

Effort (in CoCoMo) Estimate:
Effort = **74.016**

Time in Person Months Estimate:
Function Point: **11.23**
CoCoMo: **11.277**
Average: **11.2535**

Comment: The two estimates are amazingly close to one another. Don't expect this to be the case in most software projects.

Total Cost Estimate:
Industry average cost per Person Month: $8,000.00
X GameForge est. Person Months: $11.2535
GameForge est. total cost (w/o equipment): **$90,028.00**

Project Resources

While a complete team would contain all of the following personnel, PA Software has four members. Each team member will be performing multiple jobs.

Required Staff

- Lead VC++/DirectX programmer
- Assistant VC++/DirectX programmer
- Lead VB/DirectX programmer
- Assistant VB/DirectX programmer
- Windows Help programmer / Tutorial programmer
- Documentation/Librarian
- Manual Designer
- Graphic Designer
- Web Designer
- Beta Testers

No special development systems are required for GameForge. PA software will be using PCs and commonly available software.

Required Hardware

- 4 Development Systems
  - PIII 600Mhz

    - 256 MB RAM
    - 20 GB HD
    - 16 MB Video Card
    - Zip Drive
- 1 CD-ROM Writer
- 1 Scanner

Required Software

- Windows 98SE (4 licenses)
- Microsoft Visual C++ 6.0 (2 licenses)
- Microsoft Visual Basic 6.0 (2 licenses)
- Microsoft MSDN Library (newest version) (4 licenses)
- Microsoft DirectX 7.0a SDK (4 copies)
- Microsoft Office 97 (4 licenses)
- Adobe Photoshop 5.5 (1 license)

Risk Management

Project Risks

Major risks we have determined for this software are as follows:

- Equipment failure
- Late delivery of software
- Technology will not meet expectations
- End users resist system
- Changes in requirements
- Deviation from software engineering standards
- Less reuse than planned
- Poor commenting of source code

Comment: It would appear that "late delivery" is a significant issue, givene the estimates presented earlier in the plan.

For a more detailed list of project risks, see the Risk Mitigation, Monitoring, and Management (RMMM) document.

Risk Table

Risks	Category	Probability	Impact
Equipment failure	TI	70%	1
Late delivery	BU	30%	1
Technology will not meet expectations	TE	25%	1
End users resist system	BU	20%	1
Changes in requirements	PS	20%	2
Deviation from software engineering standards	PI	10%	3
Less reuse than planned	PS	60%	3
Poor comments in code	TI	20%	4

Critique: Team should define the meaning of the categories in the "category" column and the numbers in the "impact" column.

Overview of Risk Mitigation, Monitoring, and Management (RMMM)

Risk mitigation, monitoring, and management helps us pre-determine any possible major risks that may occur during development of this software. The Requirements Specification and the System Specification will be reviewed and analyzed to determine the major risks of developing this software. Each major risk found will be further analyzed to determine its overall impact upon the system. These risks will be recorded and a method will be devised to determine the best course of action if the risk should occur. Certain risks will have preventative measures devised for them. This is to reduce the possibility of more severe risks from occurring. All risks that could occur will

have a specified method to handle the risk. This is to ensure that if a risk does occur, there is predetermined path to follow when attempting to manage the risk.

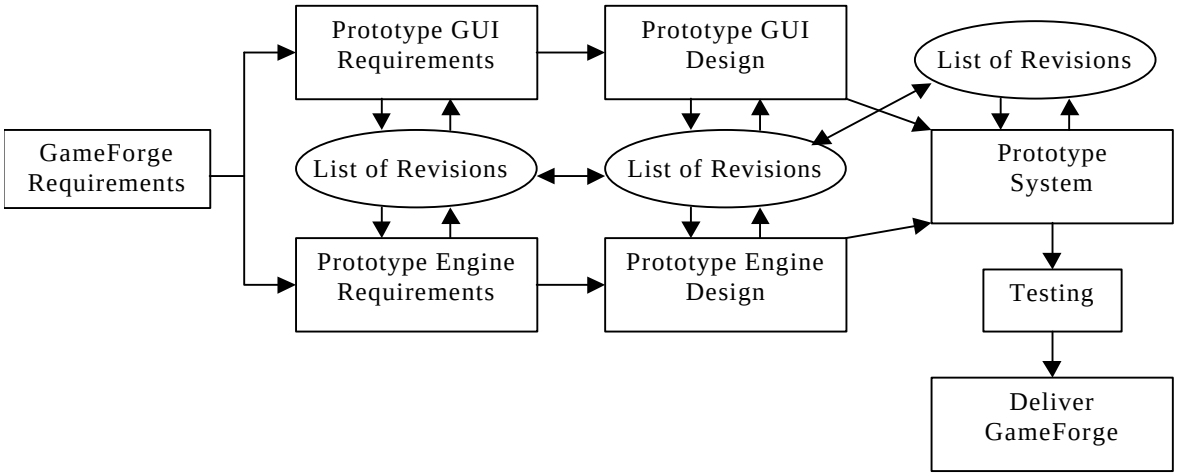
For more information see the Risk Mitigation, Monitoring, and Management (RMMM) document.

Project Schedule

Project Task Set

Process Model

PA Software will be using the Rapid Prototyping model during design and implementation:



Framework Activities

- Customer Communication
- Planning/Design
- Risk Analysis
- Programming
- Testing
- Customer Evaluation

Task Set

- Requirements specification
- Interface construction
- Engine construction
- Help construction
- Testing

List of deliverables

Documentation

System Requirements Specification

Software Requirements Specification

Design Document

Project Plan

Software Quality Assurance Plan

Risk Mitigation, Monitoring, and Management Plan

Software Configuration Management Plan

Test Plan

Code

Engine Prototypes

Interface Mockups

Interface Database

Complete Engine

Complete Interface

Integrated System

Complete Product

Functional Decomposition

Interface Task Breakdown

- Level Editor construction
- New Project wizard construction
- New Sprite wizard construction
- Database construction
- Database communication with interface
- Exporting game files ability construction
- Exporting .cpp files ability construction

Engine Task Breakdown

- Object Handler construction
- Sprite Handler construction
- Image Handler construction (DirectDraw)
- Sound Handler construction (DirectSound)
- Input Handler construction (DirectInput)
- Text Handler construction
- Logic Handler construction
  - Attribute Handling
  - Unit Pathing
- File I/O Parser construction

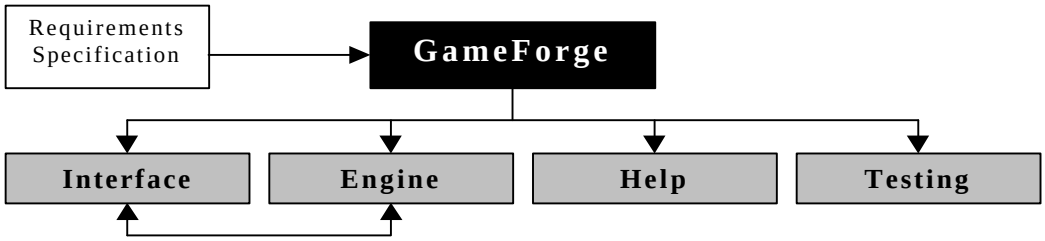
Help Task Breakdown

- Interface Help construction
- Engine Help construction
- FAQ construction
- Game building tutorials
- Manual construction

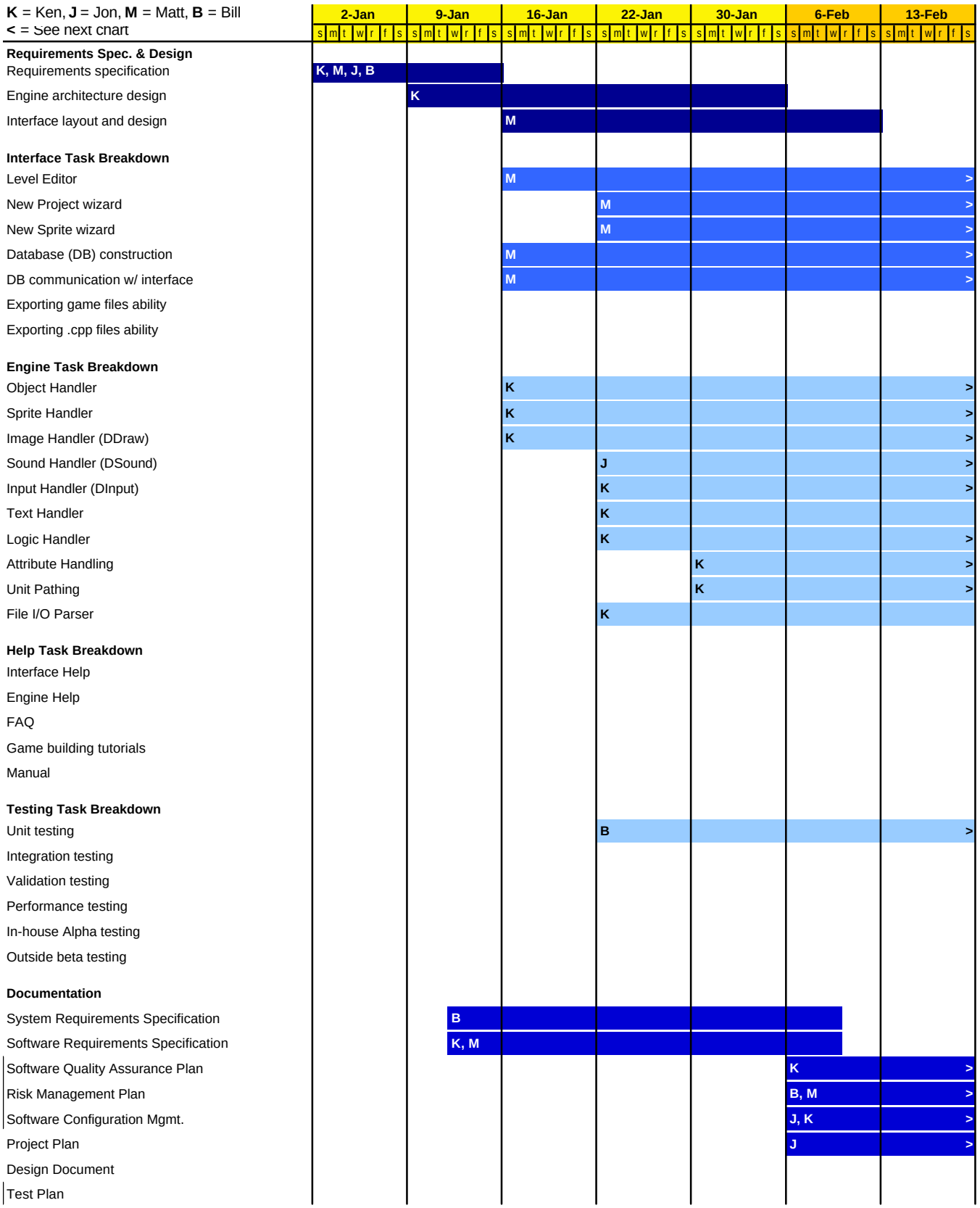
Testing Task Breakdown

- In-house, white-box and black-box testing
- Outside beta testing (including experienced programmers and novice users.)

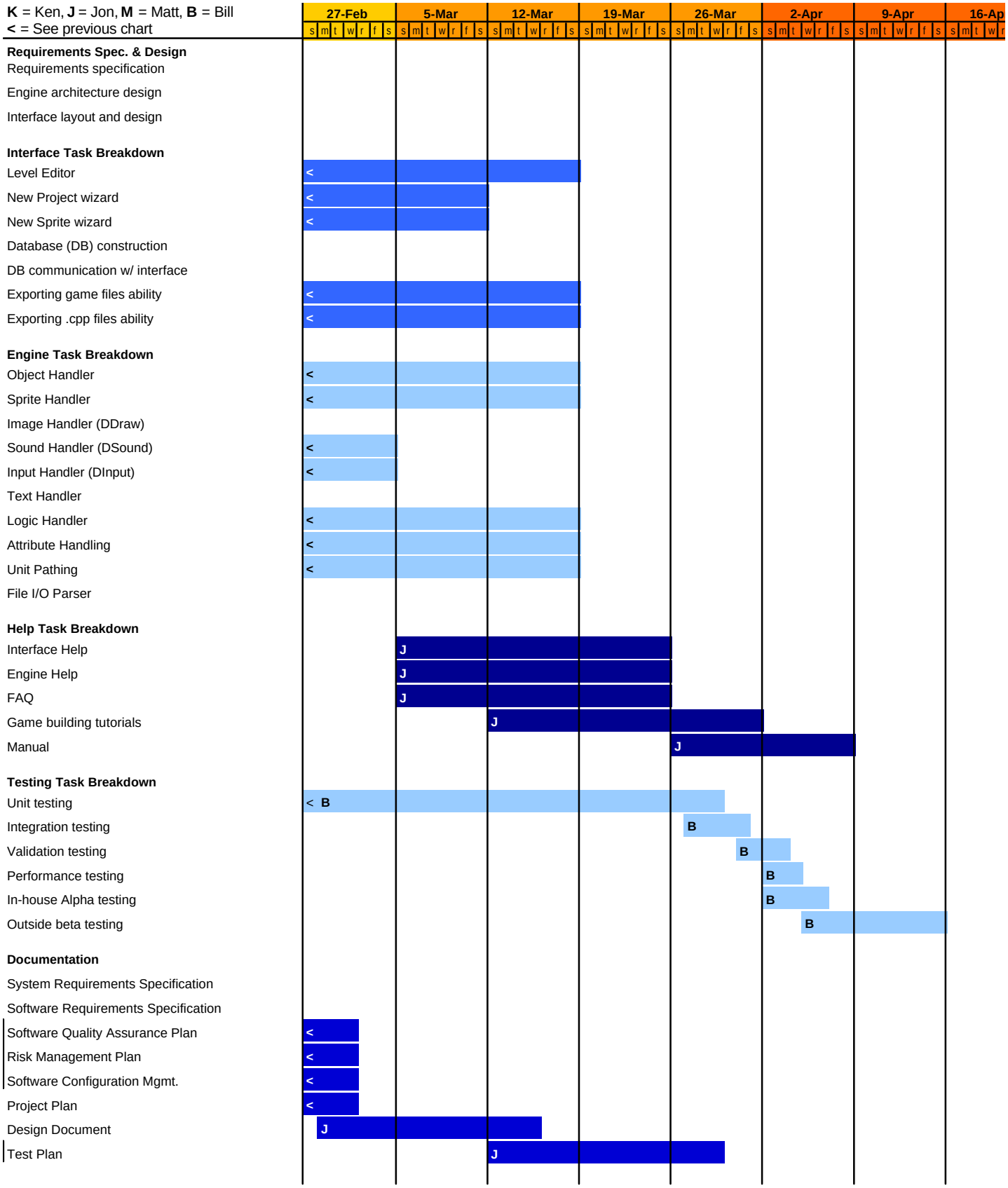
Task Network



Timeline Chart



More Timeline Chart



Staff Organization

Team Structure

PA Software uses the egoless (democratic) model for team structuring:

Role Definitions

Ken Nelson

Lead Engine Programmer: Ken is the complete DirectX engine programmer, with the exception of DirectSound. This includes all logic programming.
Lead Engine Designer: Ken is also the primary engine designer.
Interface Designer: Ken is part of the interface design team.
Help/Tutorial Programmer: Ken is part of the Windows Help team.
Documentation: Ken is responsible for much of the required documentation.
Additional Responsibilities: Ken is also the primary consultant for Jon on DirectSound issues, and for Matt on interface design issues.

Jonathan Schmoll

Assistant DirectX Programmer: Jon is coding the DirectSound portion of the engine.
Engine Designer: Jon is part of the engine design team.
Interface Designer: Jon is part of the interface design team.
Help/Tutorial Programmer: Jon is part of the Windows Help team.
Web Master: Jon is the author and maintainer of www.patheticattempts.com.
Documentation: Jon is responsible for much of the required documentation.

Matthew Forster

Lead Interface Programmer: Matt is the complete interface programmer, including all database (SQL) programming, Visual Basic programming, and DirectX (for VB) programming.
Interface Designer: Matt is part of the interface design team.
Documentation: Matt is responsible for much of the required documentation.

Bill Lord

Engine Designer: Bill is part of the engine design team.
Documentation: Bill is responsible for much of the required documentation.

Management Reporting and Communication

Mechanisms for Progress Reporting

Progress is communicated via e-mail. All files sent to other teams and/or team members are done via email or ICQ. These communications are done informally, unless special documentation of progress is required. A test log is kept for error tracking.

Mechanisms for Inter/Intra Team Communication

The GameForge team conducts weekly meetings to update other team members on their progress and ask questions that may not be answerable via electronic communication. All other communication is done electronically. Most is done via e-mail, but the GameForge team uses ICQ for real-time electronic communication, when needed.

PA Software contacts our clients via email, and sets up in-person meetings when necessary. A beta tester report form is used for formal testing outside of PA Software.

Tracking and Control Mechanisms

Quality Assurance and Control

Scope and intent of SQA activities

The SQA team's objective is to ensure that the product does not deviate far from the original design specifications. If it is discovered that deviation has occurred, the SQA team will notify the development team to prevent future deviations and to correct the previous deviations. Also, the SQA team will perform a walkthrough to analyze the product's quality at any particular stage of development. Error detection and possible enhancements are also expressed to the development team.

SQA organizational role

The SQA organizational role is to review the product(s) at specific times during product implementation. Upon reviewing, the SQA team's duties will be to evaluate the software at its current development stage and recognize any defects in the subsequent stage (design or implementation). The SQA team will directly interact with the software engineering team in group discussions, discussing any errors or possible enhancements that have been identified. In addition, the SQA team will ensure that the software engineering team has not deviated in any way from the initial design specifications.

Change Management and Control

Scope and intent of SQA activities

The primary focus of the Software Configuration Management (SCM) is to identify and control major software changes, ensure that change is being properly implemented, and to report changes to any other personnel or clients who may have an interest.

The objective of SCM is to limit the impact changes may have on the entire system. This will help to eliminate unnecessary changes, and to monitor and control any necessary changes. This allows software development to continue, despite large and/or insignificant changes without significant backtracking, lessening development time and resulting in a higher-quality product.

The SCM team will oversee these activities, and any changes to existing code or architectural design must pass their inspection before they are carried out.

SCM Organizational Role

The SCM team will work closely with the SQA (Software Quality Assurance) team, cross-examining many of the submitted documents and software change requests. Software Engineers will submit change requests directly to the SCM team for their inspection and approval.

An SCM leader will be appointed to oversee all SCM activities. He will receive all change requests, and will make any final decisions regarding those changes, including which software engineer will carry out approved changes. The SCM leader also keeps a library of all submitted requests, even those that have been denied.

Critique: The sections on tracking and control need to be more specific. Who (by name) is responsible for SQA and SCM for this project? What are major SQA checkpoints, reviews? Where can we get more information on change control procedures for this project?

Appendix

Questions for Function Point 14-Point Questionnaire

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Note: Each question is answered with a value of 0-5, based on importance (0 being the least important and 5 being the most important)

CoCoMo Value Chart

	a		**b**	
mode	**basic**	**intermediate**	**basic**	**intermediate**
organic	2.4	3.2	1.05	1.05
semi-detached	3.0	3.0	1.12	1.12
embedded	3.6	2.8	1.20	1.20