Software Engineering in Game Development

Sandro Spina 2008

Software Life Cycles

- You have been learning about different software life cycles
- Game development due to its specific nature has specific needs
- Software Engineering in the context of a computer game project

Software Engineering

- Consists of the study and practice of how to improve software development.
- SE consists of a set of techniques used to produce 'good' computer programs ...
- by 'good' we usually mean that people are then willing to buy them!!
- SE as something that the programmers do .... not just the designers or sw engineers

Business Case – Why Games

- In 2007 Games made more money at retail than Music
- In 2008 Games will probably make more money than DVD
- The main reason is that the gaming industry is always introducing new and attractive products into the market to stimulate customers … (Nintendo is the classic example)

3D Engines vs Game Engines

- We need to distinguish between a 3D engine and Game engine
- Usually a Game project assumes the 3D engine is in place
- Both have different requirements

Quality

- For a computer game project there are four main criteria to be satisfied
  - Concept - Original and Interesting idea (btw these are not easy to come by)
  - Interface - a good GUI takes a lot of thought - it will make or break your system
  - Documentation - The last thing sw engineers and programmers think about - Inline documentation
  - Stability - User interaction is usually massive ... your software needs to remain consistent no matter what the user does

Components of the Cycle

- The Software Cylce of a game project would typically (somewhere) include :
  - Computer Graphics APIs
  - Physical Simulation Engines
  - Artificial Intelligence
  - Computer Art
  - Interface Design
  - Code Optimization - ^FPS and ...
  - Optimization of the Code Optimization ^^FPS

Modularization

- Game Production =
  - { Scripting Engines,
  - 3D Components,
  - Collision Detection,
  - Visualisation,
  - Testing Tools,
  - Simulation,
  - Audio,
  - Others!}

High Level Classification of Software Games

- Based on Dimensionality of Player, World and Viewer
- 2 Dimensional | 3 Dimensional | 2.5 Dimensional (3D with constrains)
- To classify games we distinguish between:
    - Dimensionality of Player's motion
    - Dimensionality of World Motion
    - Dimensionality of the Viewer motion

A number of examples

- Examples :
  - Space Invaders - 1 / 1 / 0
  - PacMan - 1.25 / 1.25 / 0
  - Doom/Quake/etc - 3 / 3 / 3
  - SimCity/Age of Empires - 2.5 / 2.5 / 2.5

Different Platforms

- Windows Based
- Console Based
  - PS3, Wii, XBOX
  - PSP, Nintendo DS/DSi
  - Mobile devices, iPhone, etc.
  - Etc….
- Internet Based - MMOG

Playability Requirement

- Can be regarded as the most important spec of the software being built if the software is a game.
- Game Design
- Bloody good interface !!!! Wii vs PS3 is the perfect example

The Constraint Triangle !!

- A basic notion in software engineering
- Time -> Cost <-> Quality <- Time
- This is particularly important for game development

Requirements and Specification

- The development starts with a requirement for a certain kind of program
- ... and a brief specification for what such a program might be
- 'Write a really nice and great game' is slightly open-ended !!
- A detailed requirements list in a gaming/visualisation project is usually necessary to counter for all aspects of CG

Game Creative Design = Requirements

- One would need to distinguish between :
- Game 'Creative' Design and ...

- Game Software Design = how are we going to go about meeting the creative 'artistic' design

UML diagrams in Game Development

- Use case diagrams for software requirements
- Class diagram for the high-level structure
- Sequence diagrams for interactions of program objects
- These are especially useful if using a previously developed framework (eg XNA | POP)

Requirements gathering

- For traditional software projects the waterfall model is usually sufficient
- For game development projects requirements will usually (always) alter during implementation
- Depending on the project - Eg. MMOG, Console Games, Middelware needs to be identified - Eg. BigWorld

Architecture and High-Level Design

- These are ideally fixed at the onset …. extremely important
- Then one can have requirements which can be accommodated within this high-level design
- Similar to a Staged-Delivery lifecycle

Inventor Lifecycle

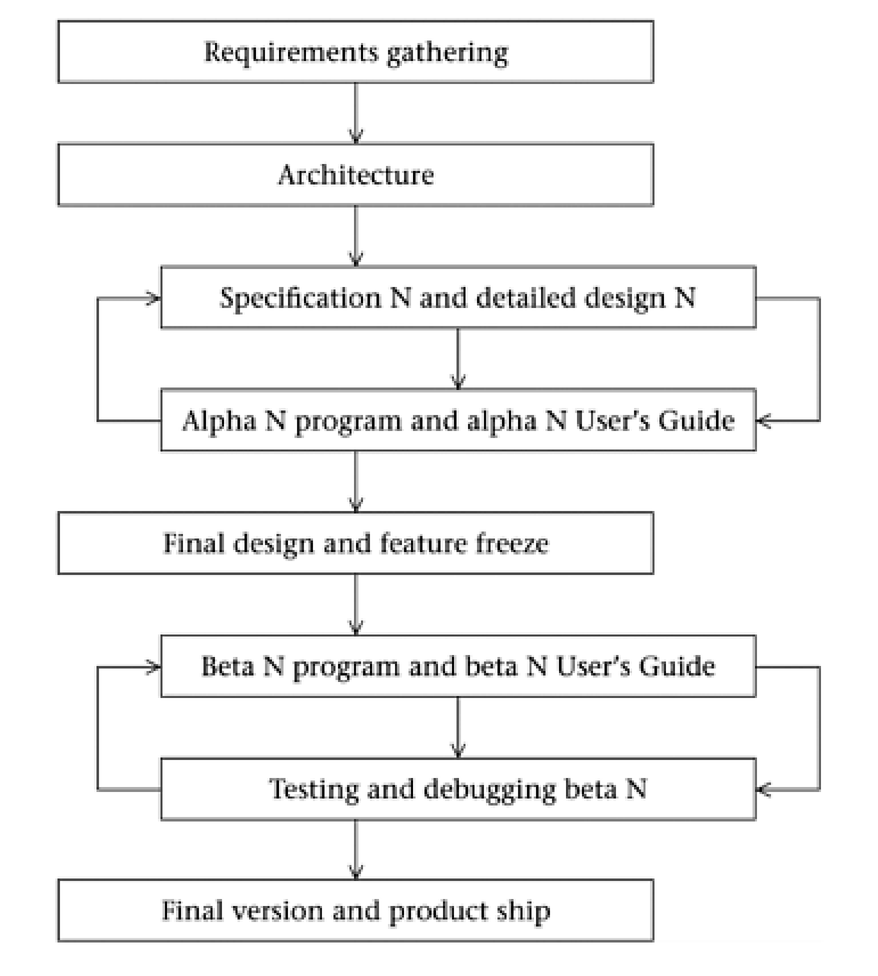

Requirements gathering
Architecture
Specification N and detailed design N
Alpha N program and alpha N User's Guide
Final design and feature freeze
Beta N program and beta N User's Guide
Testing and debugging beta N
Final version and product ship

Image taken from book: Software Engineering and Computer Games by Rudy Rucker

The Development Spiral

- Analysis -> Design -> Maintenance -> Implementation
- Always keeping in mind the overall Architecture and High-Level Design
- ... which more often than not occasionally changes as well !!

Managing your project in teams

- You'll have to use some form of source control - otherwise it becomes impossible to manage
- Each member of the team is usually assigned a different aspect of the game engine ... eg. visualisation, physics, scripting, AI, sound

Object-Oriented SE

- "software design appears to be a collection of interleaved, iterative, loosely-ordered processes under opportunistic control ... Top-down balanced development appears to be a special case occuring when a relevant design schema is available or the problem is small ... Good designers work at multiple levels of abstraction and detail simultaneously" - B. Curtis

Object Oriented Design

- OO is ideal for game development
- C++ is the preferred language
- Instead of having to analyse a problem in terms of many (many) tasks, we look at the problem in terms of a few high-level classes.
- Abstraction is the key !!

OOA -> OOD -> OOP

- OO Analysis : Which classes? UML Diagrams
- OO Design : UML diagrams, *.h headers
- OO Programming : *.h headers, *.cpp implementations
- Note : Boundaries are very fuzzy in the sense that usually you don't finish one stage and start the other

Top-down design

- "Our experience indicates that design is neither strictly top-down, nor strictly bottom-up. Instead . . . well-structured complex systems are best created through the use of 'round-trip design.' This style of design emphasizes the incremental and iterative development of a system through the refinement of different yet consistent logical and physical views of the system as a whole . . . Object-oriented design may seem to be a terribly unconstrained and fuzzy process. We do not deny it. However, we must also point out that one cannot dictate creativity by the mere definition of a few steps to follow or products to create." - G Booch

Software Design Patterns

- A number of design patterns are usually essential when creating a game project
- Composite, Bridge, Singleton etc ....
- For 3D engines Singleton is particularly useful
- Singleton - addresses the problem of when the programmer wants to have a class that one only wants to have one single, easily accessible instance of. Eg 3DEngine,

Meson Framework

- It is a platform which provides unified cross-platform scripting, physics and visualisation services.
- Simulation Engine
- Three main components + common
  - Visualisation
  - Physics
  - Scripting
  - Common

Vistas Component (Keith Bugeja)

- Vistas is the middleware between low-level graphics APIs and real-time graphics applications.
- It provides functionality such as scene management and manipulation, visibility determination, etc…
- IMP: it provides a mechanism by which the developer can modify, change or extend it.

Design Considerations (Keith Bugeja)

- Provide high-performance graphics rendering
- Ease cross-platform development by providing a high level of portability
- Provide flexibility and customisation
- Be devoid of assumptions that may limit or lock applications in predetermined operation modes.
- Allow functionality to be added through extensibility
- Provide a consistent and homogenous programming interface
- Provide a well-established programming paradigm which makes it clear and easy to use

Vistas Scenarios (Keith Bugeja)

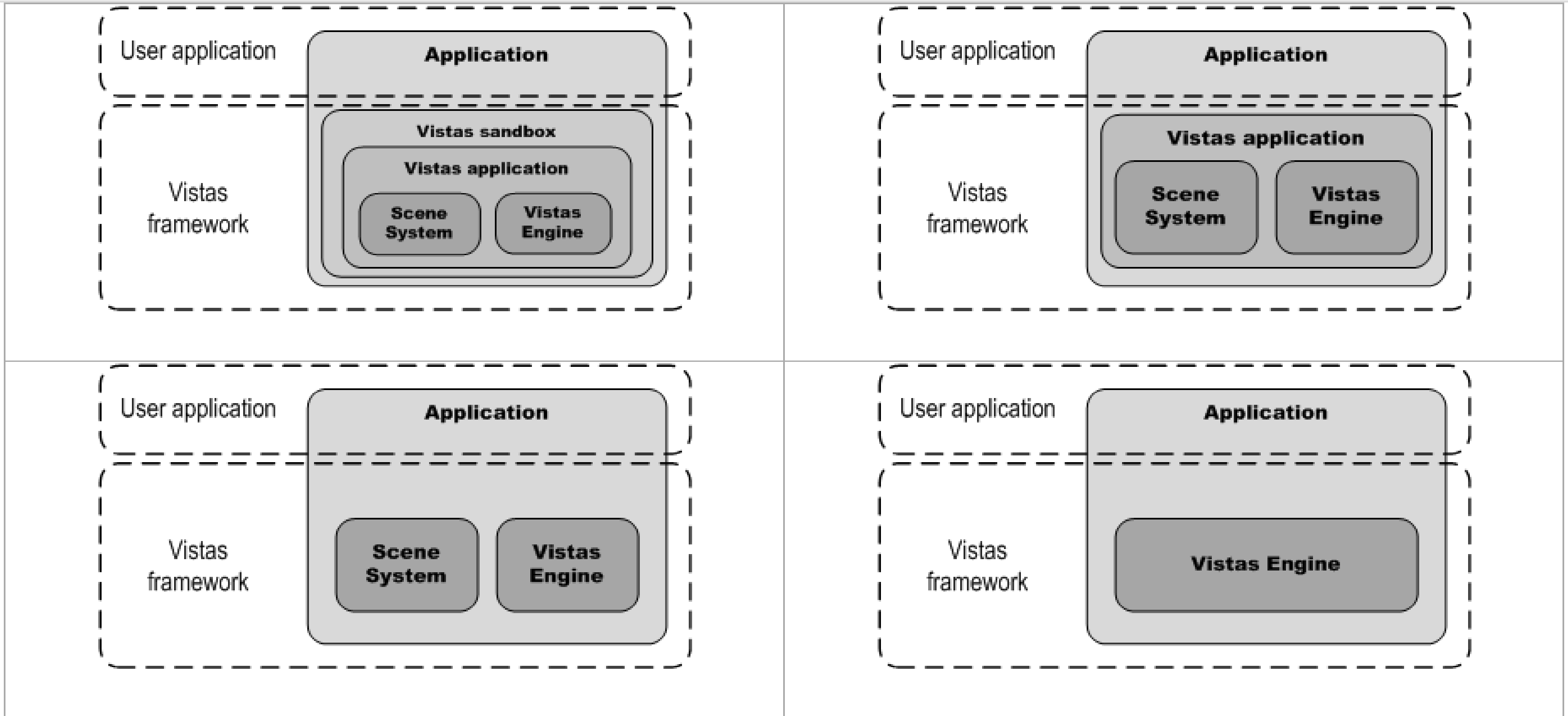
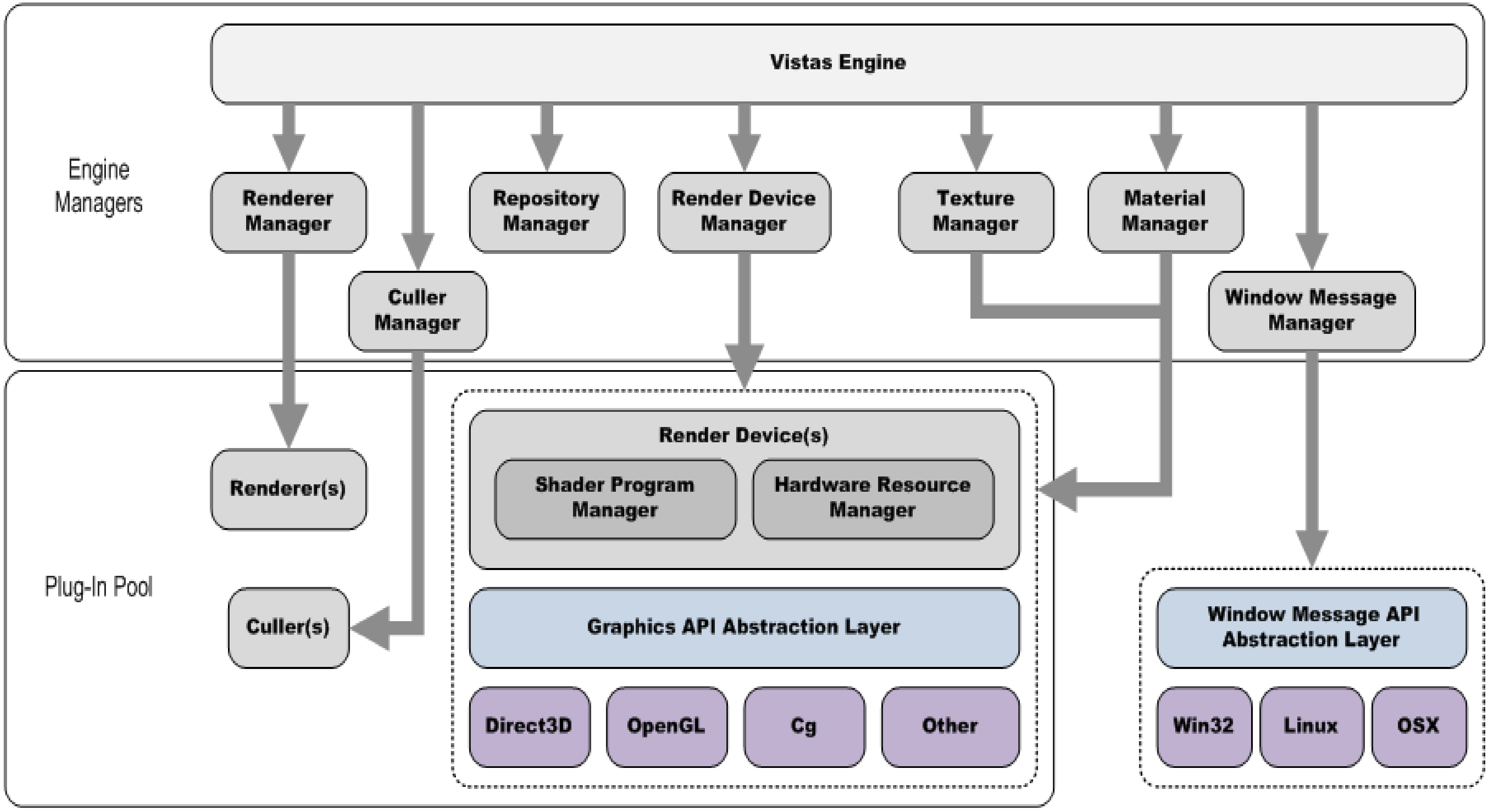


Figure 5-1 Vistas Architectural Scenarios. Top left: Application sits on top of Vistas Sandbox. Top right: Application sits on top of Vistas Application. Bottom left: Application talks directly to Scene System and Vistas Engine. Bottom right: Application discards Scene System and talks directly to Vistas Engine

Vistas Architecture



Plug-In System

- Non core functionality is located in the plug-ins system.
- New techniques (eg. Cullers, renderers, etc) are implemented here.
- There is a clear separation (at the design stage) between what is core to the Framework and what is not.

Designing Scene Graphs

- A real-time 3D visualisation engine is required to process complex virtual worlds composed of a large number of entities, determine which of these entities is within the filed of view of the observer, and draw the visible entities …

- Scene Graphs and Nodes – The data structures used need to exploit spatial and render-state coherency

Typical stages in a games program

- Initialise Engine + Load Resources
- Gaming Loop
  - Compute Logic
  - Update Scene Graph
  - Cull
  - Render
- Software engineers need to appreciate the complexity of data structures (acceleration structures like kd-trees) otherwise they'll never be able to achieve decent real-time performance.

Events and Listeners to Events

- Event-based system are particularly suitable for highly interactive systems.
- 3D and Game engines make extensive use of events and listeners.