SOFTWARE ARCHITECTURAL DESIGN

Software Engineering Process and Practice

28 April 2003

Dr Michael Papasimeon

What is Engineering?

The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.

EXPECTATIONS

- To learn what is meant by “Software Architecture”, its importance in the software development process, and how to recognise the good and bad characteristics of different software architectures.
- Don't expect you to be world leading software designers at the end of this class; this will come with experience and lots of practice.

WHAT IS SOFTWARE ARCHITECTURE?

- **Software architecture** refers to the theory behind the actual design of computer software. In the same way as a building architect sets the principles and goals of a building project as the basis for the draftsman's plans, so too, a software architect sets out the software architecture as a basis for the actual design specifications. **http://www.wikipedia.org/wiki/Software_architecture**
- A Software Design must be **documented** to exist.
- A Software Architecture must describe both the **STRUCTURE** and the **BEHAVIOUR** of a software system.

WHY IS SOFTWARE ARCHITECTURE IMPORTANT?

Barry Boehm

If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.

- Mutual Communication
- Early Design Decisions
- Transferable Abstraction of a System

ARCHITECTURAL VIEWS

- Functional/Logic View
- Code View
- Development/Structural View
- Concurrency/Process/Thread View
- Physical/Deployment View
- User Action/Feedback View

WHY DO WE NEED A SOFTWARE ARCHITECTURE?

- To understand how the system is intended to work.
- To organise software development.
- To foster reuse.
- To evolve the system.

WHAT INFLUENCES THE SOFTWARE ARCHITECTURE

Requirements and Software Patterns

- Requirements and use cases
- Experience -- previous architectures and architectural patterns

Constraints and Enablers

- System software
- Middleware (e.g. frameworks)
- Legacy systems
- Standards and policies
- Non functional requirements
- Distribution needs

ARCHITECTURAL DESCRIPTION

- Architecture should be developed after requirements have been gathered and analysed.
- Architecture should be documented and should remain relatively stable throughout the life of the project.
- Architecure may be updated, but shouldn't grow dramatically. Changes may include:
  - Finding new abstract classes and interfaces
  - Adding new functionality to existing sub-systems/modules
  - Upgrading to new versions of re-useable components
  - Re-arranging the process structure.

ARCHITECTURAL EXAMPLES

- Pipeline
- Transaction
- Layered
- Repository
- Client-server
- Distributed computing
- Peer-to-peer system
- Model-View-Controller
- Monolithic system
- Three-tier model
- Structured
- Component architecture

MODULAR SOFTWARE DESIGN

- Reduces Complexity.
- Facilitates Change (Easier Maintainability).
- Easier Implementation by encouraging parallel development.
- Module is the basic unit of software used in a design description.
- Determined using step-wise refinement (logical parts)
- Modules, sub-systems, components, packages, and classes are all software elements used for **abstraction** and **information hiding** [1].

[1] Pressman 3rd Edition (Section 10.4 pg 332)

LEVELS OF ABSTRACTION

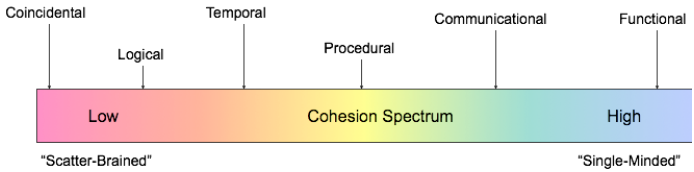
- Data Structures and Algorithms
- Classes (data structures and many algorithms)
- Packages/Modules (groups of related, possibly interacting classes through design patterns).
- Modules/Subsystems (interacting modules each containing many classes, but only the public interfaces interact with other modules/subsystems).
- Systems (systems interacting with other systems, hardware, software and human).

Software architecture tends to be concerned with the last two.

MODULE COHESION

- A measure of how functionally coherent a software module is.
- A software module should have **High Cohesion**.
- This means that a software module is concerned with only doing one thing – and doing that thing well.
- For large modules, their components/classes should be functionally related.

TYPES OF COHESION



A measure of the relative functional strength of a software module

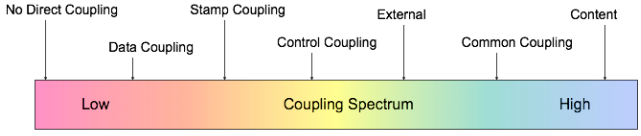
Software Module Cohesion...

- **Coincidental:** multiple, completely unrelated actions or components
- **Logical:** series of related actions or components (e.g. library of IO functions)
- **Temporal:** series of actions related in time (e.g. initialisation modules)
- **Procedural:** series of actions sharing sequences of steps.
- **Communicational:** procedural cohesion but on the same data.
- **Functional:** one action or function

MODULE COUPLING

- Coupling is a measure of how inter-related or interdependent software modules are.
- There should be **LOW COUPLING** between software modules.
- This means that should be only a few well defined, functionally relevant dependencies between software modules.
- Highly coupled software is BAD software. (difficult to understand, maintain, debug etc…)

TYPES OF COUPLING



A measure of the interdependence among software modules

Software Module Coupling...

- **Content:** one module directly references the content of another
- **Common:** both modules have access to the same global data.
- **Control:** One module passes the element of control to another.

DESIGN FOR REUSE

- Reusable Components/Modules are general
- Reusable Components are Loosely Coupled
- Requires High Cohesion and Published Interface

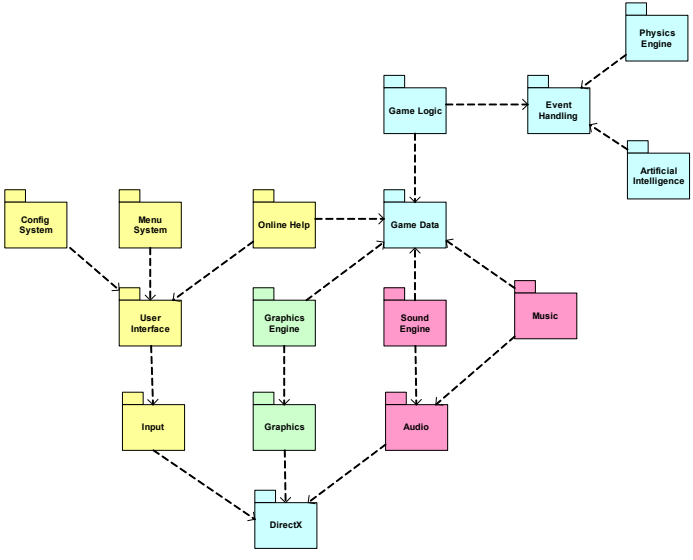
PERFORMANCE CONSTRAINTS

- Having lots of layers can result in performance issues.
- It is a good idea to generally design your architecture with high cohesion and low coupling and then adapt it to iron out performance issues.

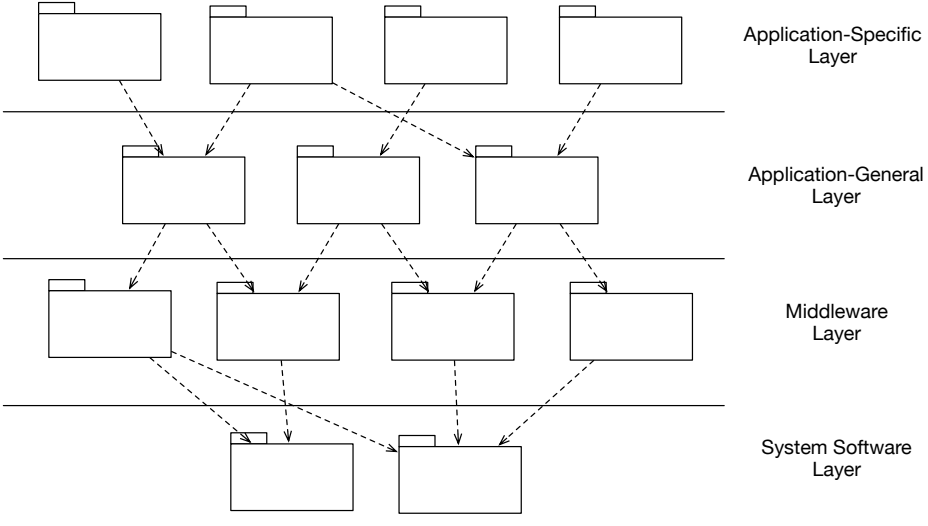
STEPS IN CREATING A SOFTWARE ARCHITECTURE

- Identify major modules (or sub-systems) and sub-modules.
- Identify interactions with external actors and systems.
- Identify dependencies between modules.
- Determine the public classes, functions or sub-modules for each module.
- Identify which public classes from each module will interact with each other.
- Determine the interaction order between the modules.
- Determine how two modules will interact using experience and design patterns.
- Identify the methods (functions) used that will interact with other modules methods (functions).
- Determine the dynamic interactions (run-time)
- Prototype the architecture
- Document it.
- Review it.
- Iterate through it a couple of times to make sure it is modular, extensible, satisfies requirements (including performance etc).
- Compile it...

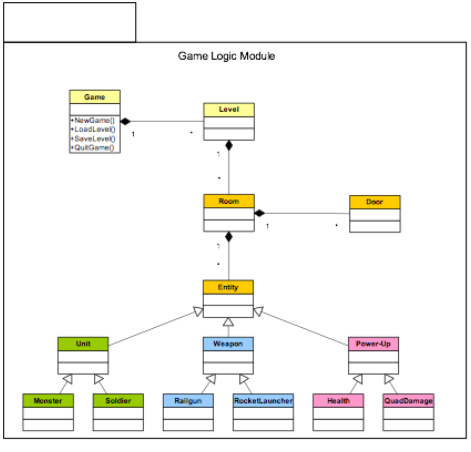
EXAMPLE GAME ARCHITECURE: MODULE DEPENDENCIES

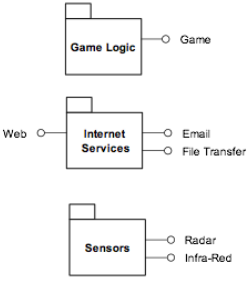


LAYERED ARCHITECTURE

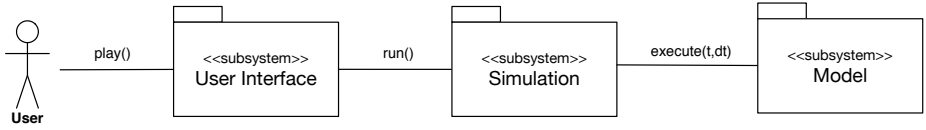


MODULE INTERFACES





ARCHITECTURAL BEHAVIOUR



DETAILED DESIGN: MODULE SPECIFICATION

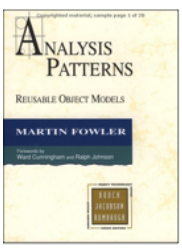
- What the function of a module is:
- Description of what the purpose and function of a module is.
- Inputs and Outputs
- Dependencies
- Errors and Exceptional Conditions

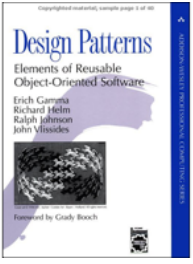
SOME QUESTIONS FOR THE PRELIMINARY ARCHITECTURAL DESIGN REVIEW

- Does the architecture satisfy the requirements?
- Is effective modularity achieved?
- Are interfaces defined for modules and external system elements?
- Is the structure of the data and it's organisation consistent with the domain of the requirements?
- Is the structure of the data consistent with the requirements?
- Has maintainability been considered?
- Have quality factors been explicitly assessed?

BOOKS ABOUT SOFTWARE DESIGN

- Design Patterns: Elements of Reusable Object Oriented Software (Gamma et. al)
- Pattern-Oriented Software Architecture,
  Vol.1: □ A System of Patterns (Buschmann et. al)
- Software Engineering : A Practitioner's Approach (Pressman)
- The Unified Software Development Process (Jacobson, Booch, Rumbaugh)
- Use Case Driven Object Modelling and Design with UML: A Practical Approach (Rosenberg)
- UML Distilled (Fowler)
- Analysis Patterns (Fowler)





BOOKS ABOUT SOFWARE ARCHITECTURE

- Applied Software Architecture (Hofmeister)
- Evaluating Software Architectures: Methods and Case Studies (Clements)
- Design and Use of Software Architecture (Bosch)
- The Software Architect's Profession: An Introduction (Sewell et al)
- Software Architecture: Organizational Principles and Patterns (Dikel)
- Documenting Software Architectures: Views and Beyond (Clements)
- Software Fortresses: Modeling Enterprise Architectures (Sessions)
- Beyond Software Architecture: Creating and Sustaining Winning Solutions (Hohmann)