ARTHUR ANDERSEN

Introduction to RAD*Frame*™/OO

Martin Israelsen
Arthur Andersen
Advanced Technology Group
Atlanta

Methodologies

- There are three types of methodologies
  - Structured Methodologies
    - Focus on the functionality of the system
    - Introduced between the late 60's and late 70's
  - Information Engineering
    - Focus on the data of the system
    - Introduced in the late 80's

      -> used in RAD*Frame*lClassic
  - Object Oriented
    - Focus on both data and functionality
    - Introduced between the late 80's and early 90's

      -> used in RAD*Frame*lOO

Methodologies

- There are three types of methodologies
  - Structured Methodologies
    - Focus on the functionality of the system
    - Introduced between the late 60's and late 70's
    - Systems are structured around the functions they have to perform
    - Data structures are determined by the function structure
    - Best where the functionality is more complex and important than data

      Note - Data is not encapsulated
    - Functions are not stable elements, and often change during the project life cycle
  - Information Engineering
  - Object Oriented

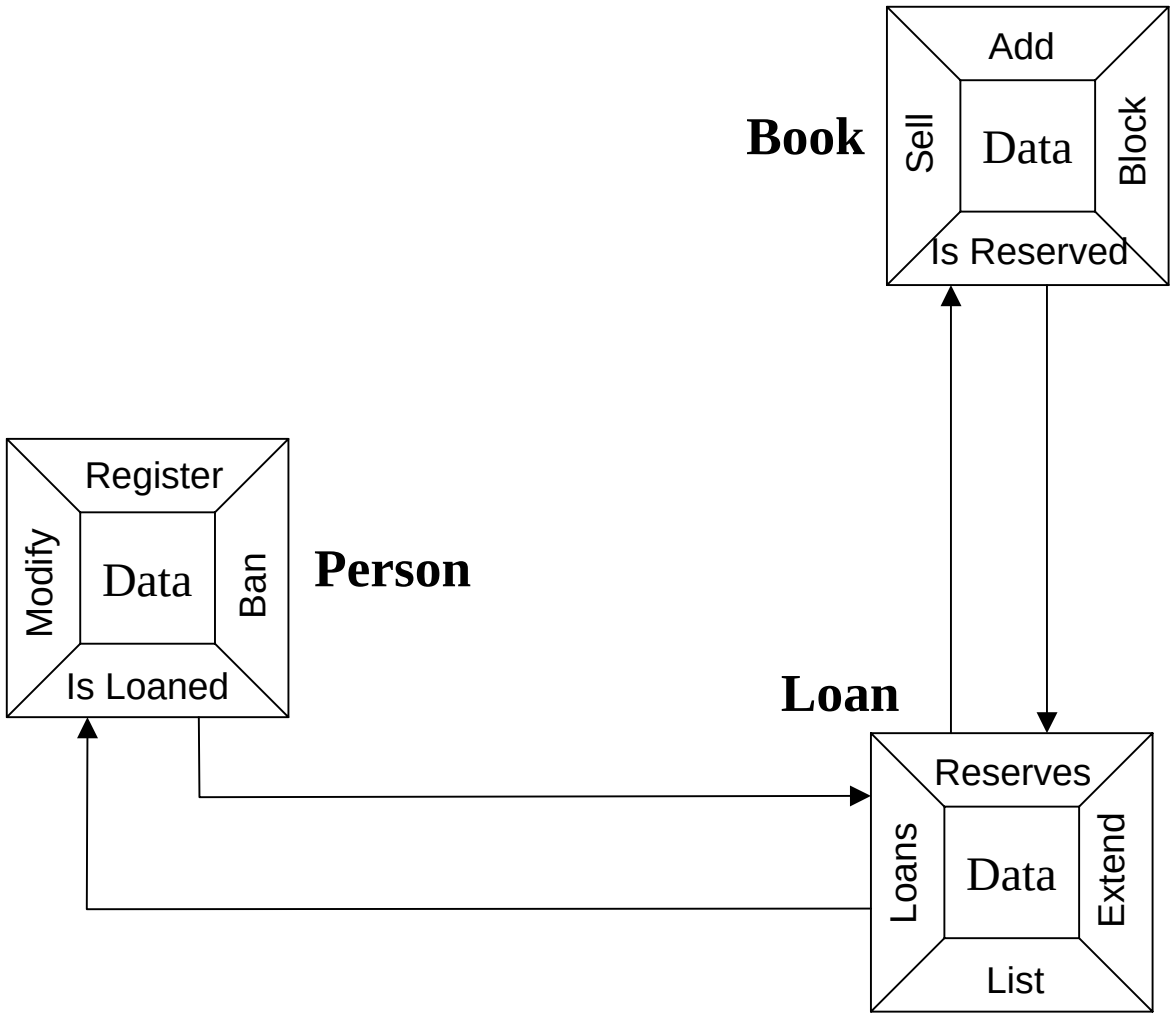
Methodologies

- There are three types of methodologies
  - Structured Methodologies
  - Information Engineering
    - Focus on the data of the system
    - Introduced in the late 80’s
    - The systems are structured around data, the most stable elements
    - Enterprise-wide data model forms the basis for the development of applications
    - Applications developed using functional (ie structured) techniques
    - Realised with integrated CASE tools
    - Functional design of the applications still prejudices maintainability and reusability
  - Object Oriented

Methodologies

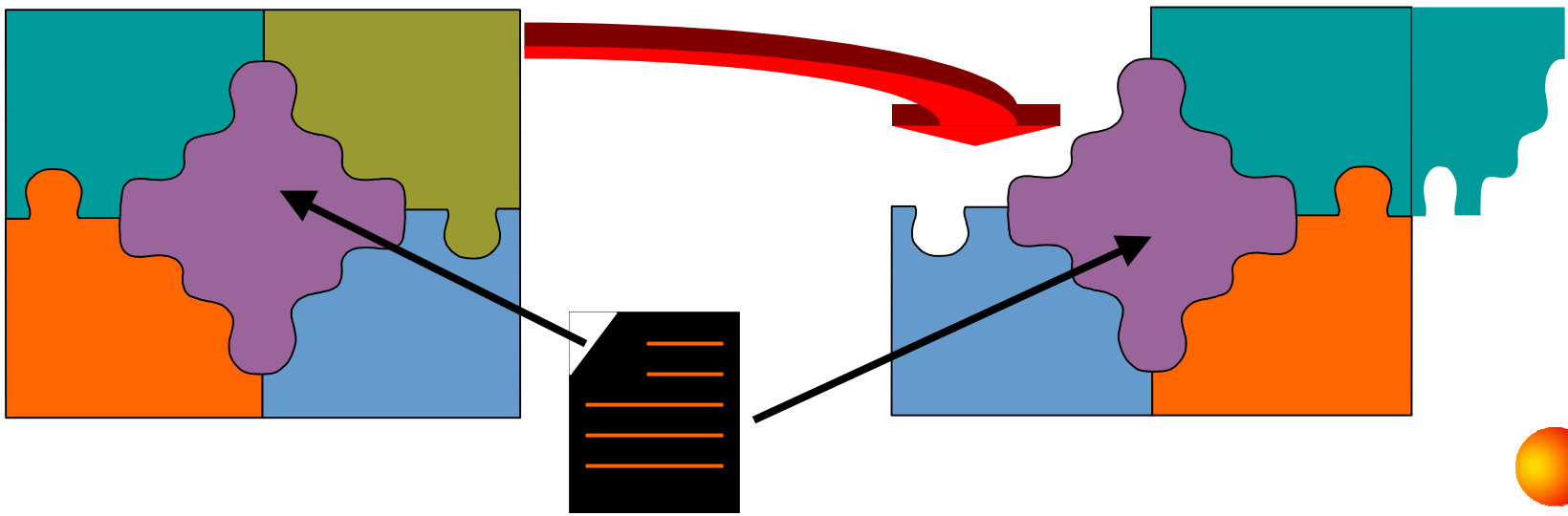
- There are three types of methodologies
  - Structured Methodologies
  - Information Engineering
  - Object Oriented
    - Focus on both data and functionality
    - Popularised between the late 80's and early 90's
    - *Objects* encapsulate both data and functions
      - Only functions of the object can access its data
    - *Objects* communicate by sending messages to each other
    - The expertise available in the market is still maturing
    - The first OO project will require significant investment to build experience

Object-Oriented Methodologies



Why Object oriented?

- The system can continue to evolve even after it goes live
  - Less ‘ripple’ effects when an alteration is made
- Easy to reuse components
- Business requirements and the coded objects share the same metaphor



Rapid Application Development (RAD)

- Coined by industry guru James Martin
- Is an incremental method allowing for quick systems development without sacrificing quality.
- RAD*Frame*™ is “Rapid Application Development Framework”
  - The Arthur Andersen approach to software development

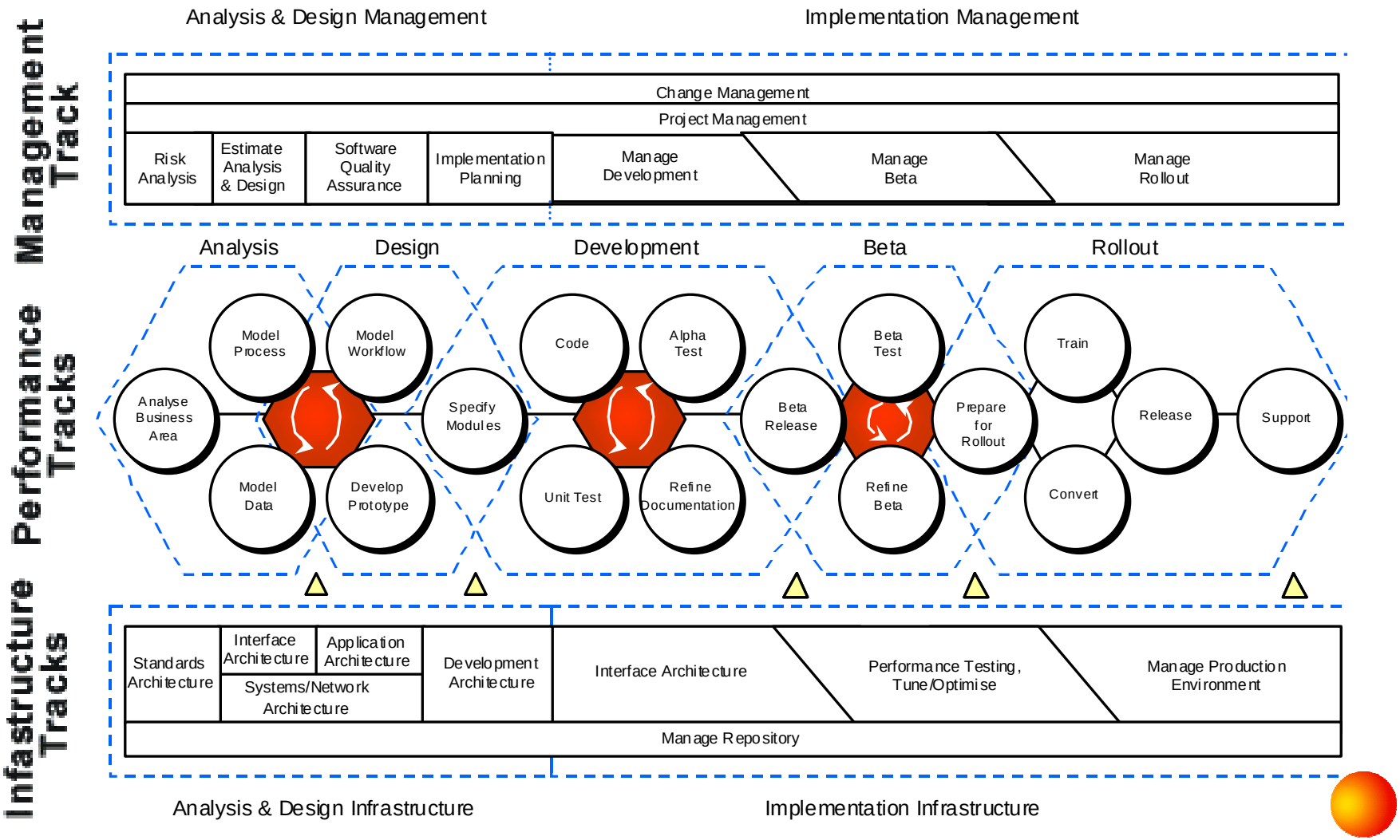
Advantages of RAD

- Focuses on "**time boxing**" which should allow for delivery and implementation of functional pieces every 3 to 4 months.
- Responds to a rapidly **changing business environment**.
- Incorporates **management** and **user involvement**.
- Facilitates **prototype development** by employing a variety of automated design and development tools.

Fundamental Concepts of RAD*Frame*

- Rapid Application Development
- A Framework, not a Methodology
- A Synthesis of Proven Methods
- Evolution, not Revolution
- Software Engineering, not Hacking
- Integration of Process and Technology

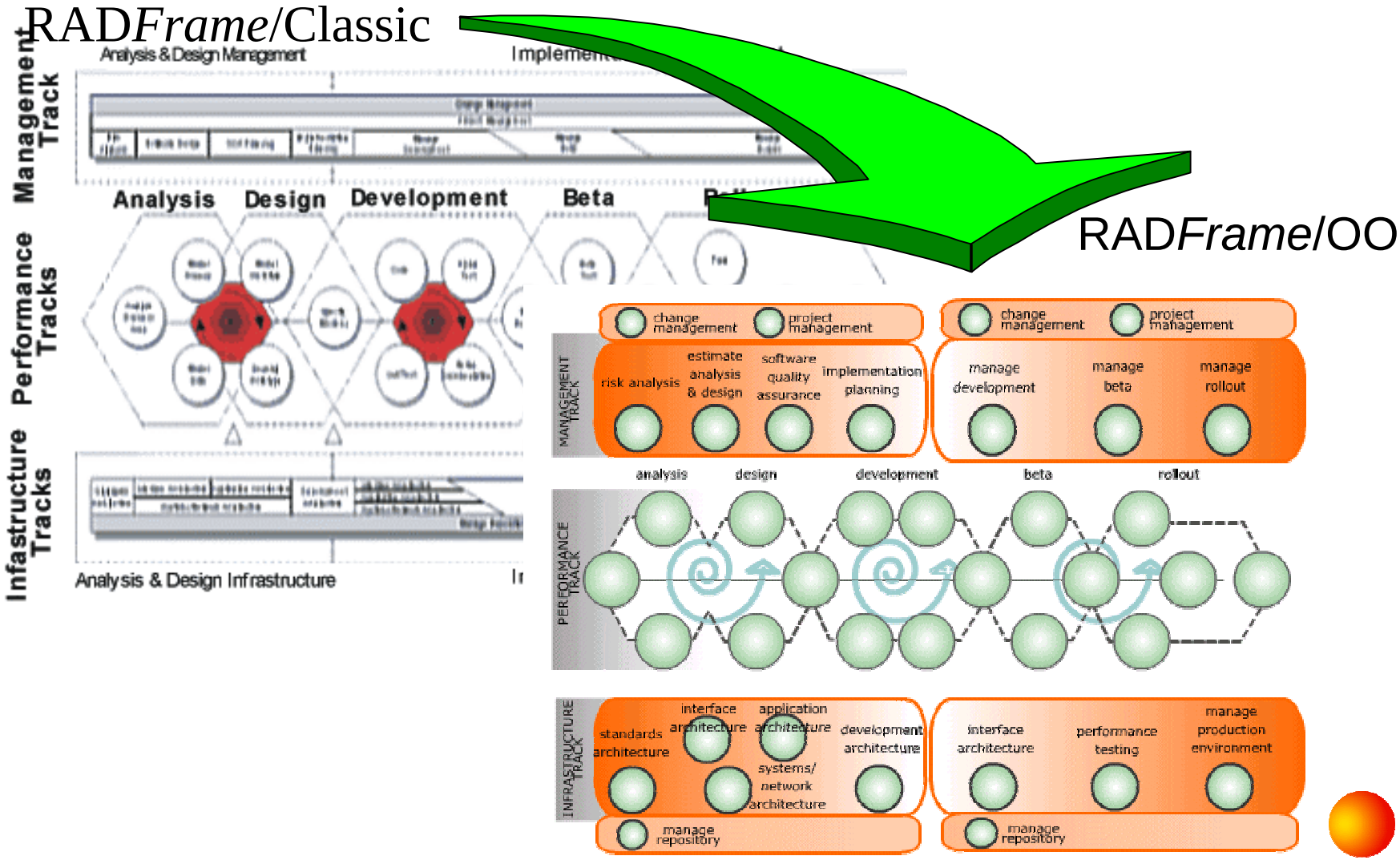
RAD*Frame*/Classic (1995)



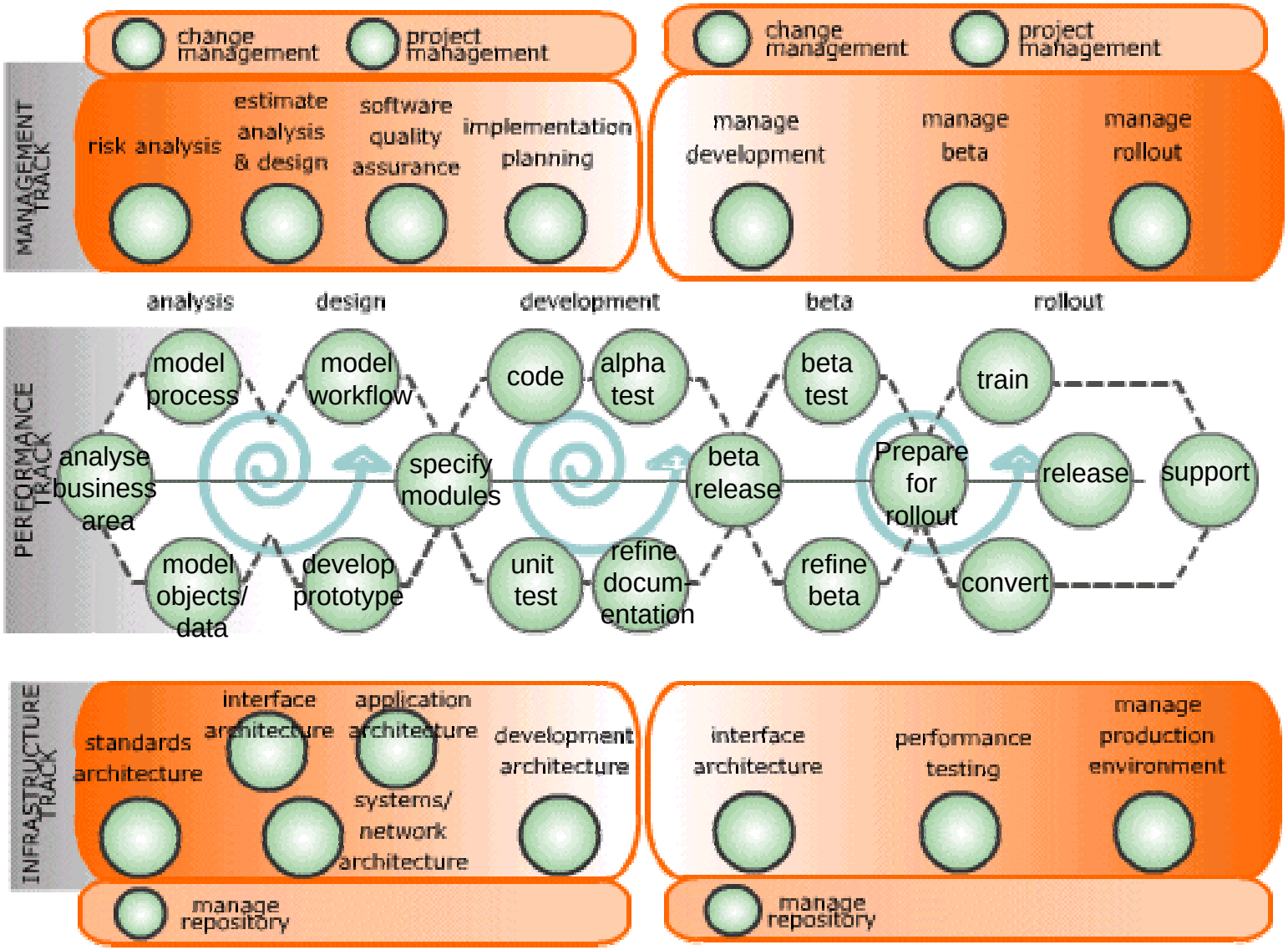
RAD*Frame*/Classic (1995)

- Rooted in the needs of development in non-Object Oriented languages & RDBMS
- Processes and data are defined together yet described separately.
- Best suited to extensions of non-object oriented environments
- Revamped in 1999 to include new toolset

RAD*Frame* : evolutionary approach



RAD*Frame*/OO (1999)

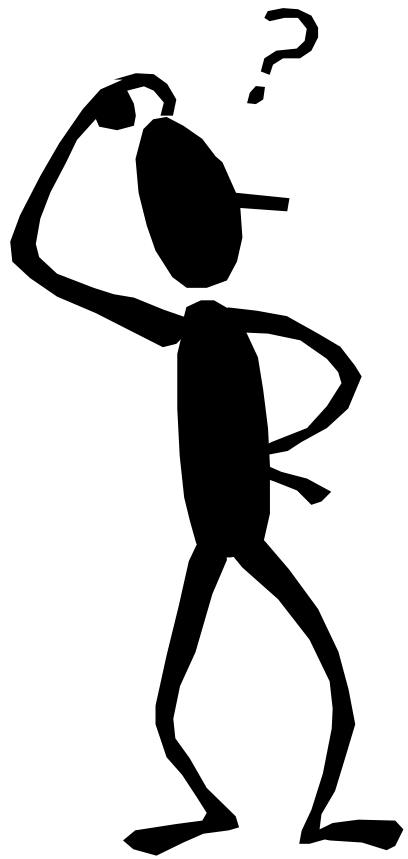


RAD*Frame*/OO (1999)

- Developed in response to the increasing use of OO languages
- Uses practises and tools that maximise the value of other OO approaches
- Uses Unified Modelling Language (UML) throughout the analysis and design phases.

ARTHURANDERSEN

Questions?



RAD*Frame*™

- What is RAD*Frame*™?
  - Tracks,Phases, Iterations
- Why use RAD*Frame*™?
  - Strengths
  - Benefits to the practice!
  - Benefits to you!

What is RAD*Frame*™ ?

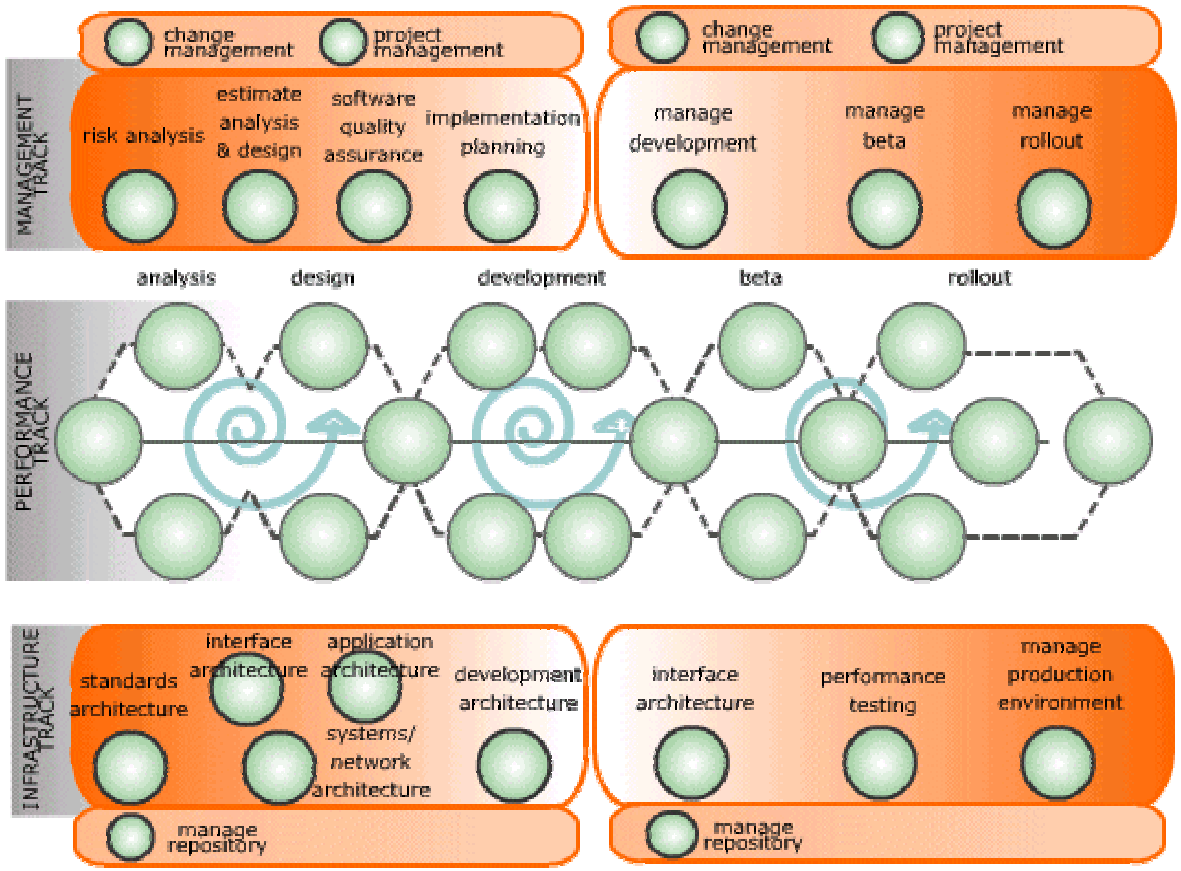
- RAD*Frame*™ is a process framework for software engineering used to deliver custom information systems to enterprises.
- Describes best practice in software engineering within Arthur Andersen
- Applicable to all software engineering engagements

RAD*Frame*™ Objectives

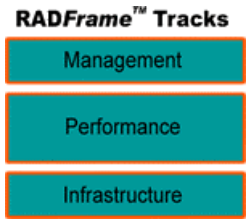
- Production of an engineered information system integral to the needs of the organisation
- Delivery of an end product which exceeds user expectations, performs above the required level, and finishes within budget
- Integrate Process Technology perspectives, and workflow to provide an innovative solution
- Capture and use knowledge to optimise the current development and improve future development
- Rapidly design, develop, and deliver an application without sacrificing reliability, reuse, or maintainability

RAD*Frame*® Map

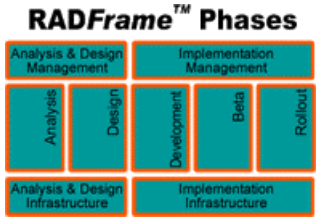
- A3S website
  - Menu: Toolbox - Component Methodology Selector
    - RAD*Frame*



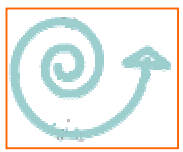
Tracks, Phases and Iterations



Run horizontally across the model and roughly equate to the roles played in the process

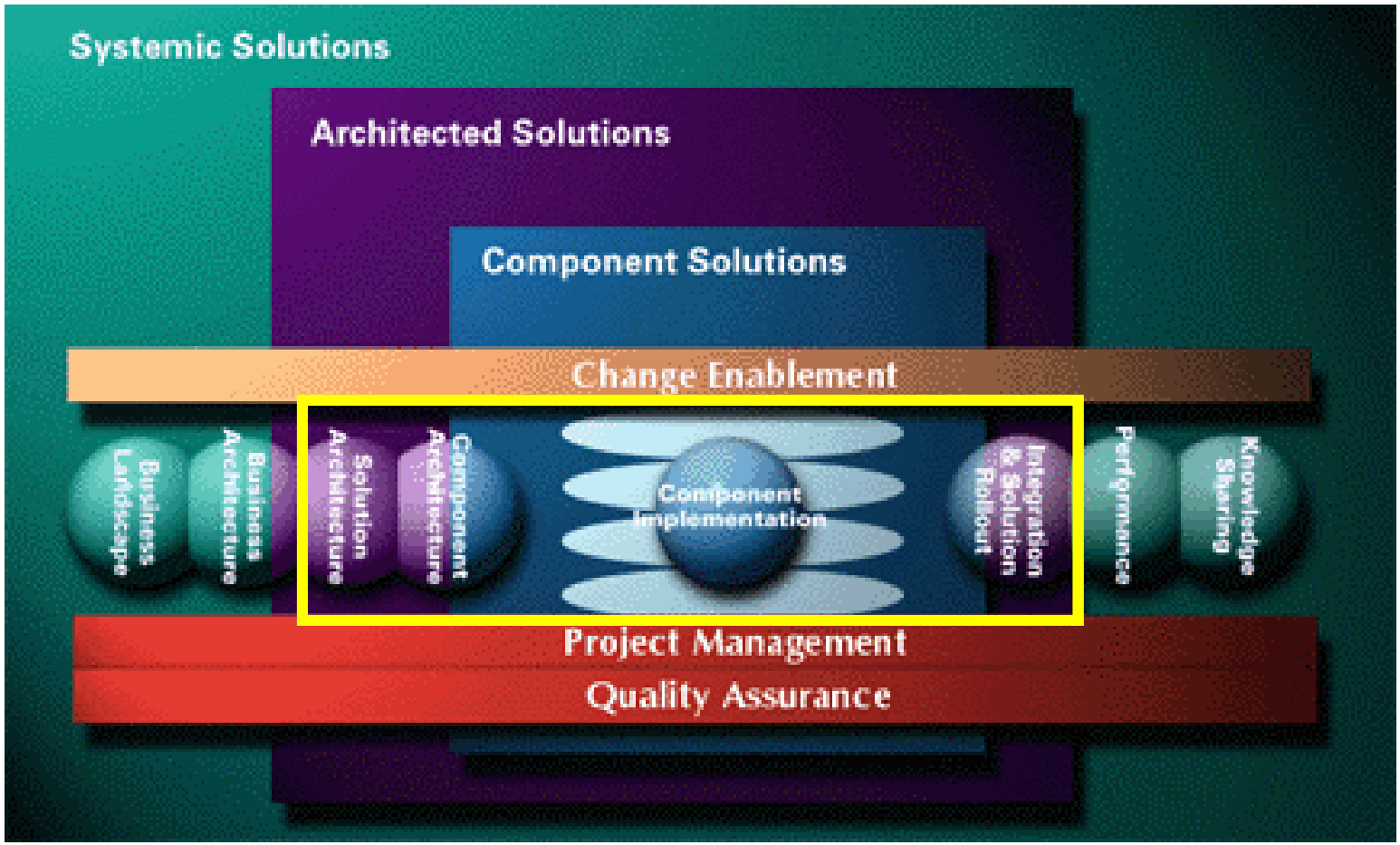


Run vertically and equate to the major activities of the project, with defined checkpoints for each phase

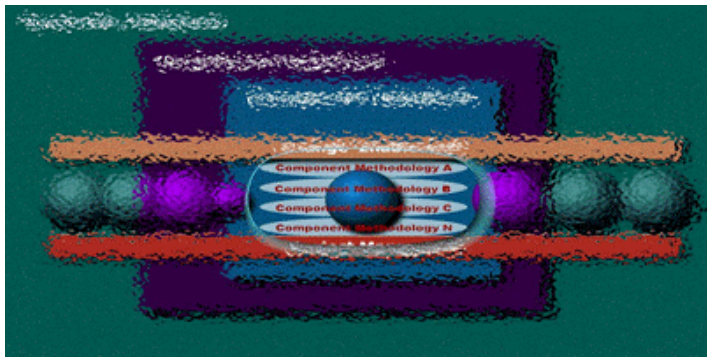


Denotes the iterative nature of the process, i.e. the trial, evaluate, and improve characteristics of RAD development

A³S Compliant



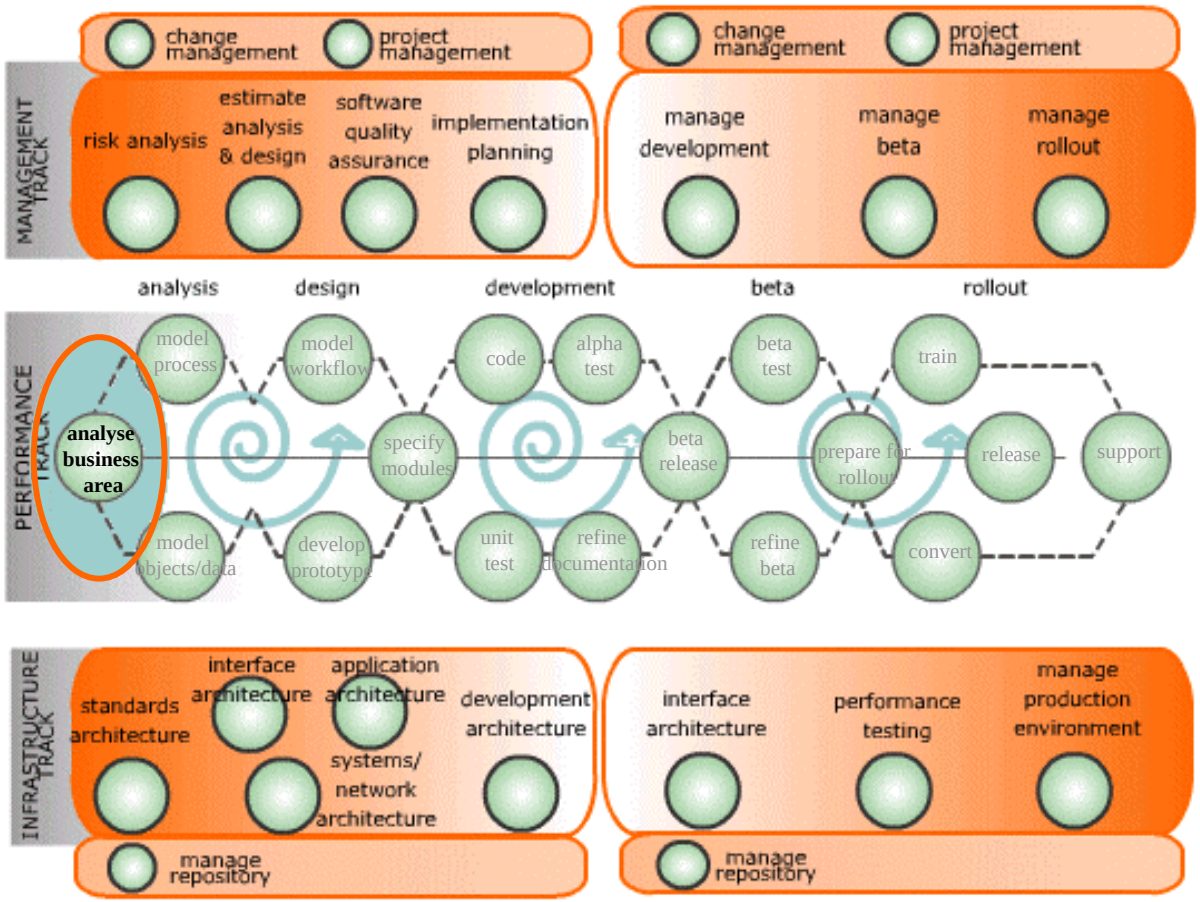
A³S Compliant



Phases are aligned with several of the A3S phases

RAD*frame*	Architected Solutions
Analysis and Design Management	Project Management, Change Enablement
Implementation Management	Project Management, Change Enablement
Analysis	Solution Architecture
Design	Component Architecture
Development	Component Implementation
Beta	Component Implementation, Integration and Solution Rollout
Analysis and Design Infrastructure	Solution Architecture, Component Architecture
Implementation Infrastructure	Component Architecture, Component Implementation, Integration and Solution Rollout

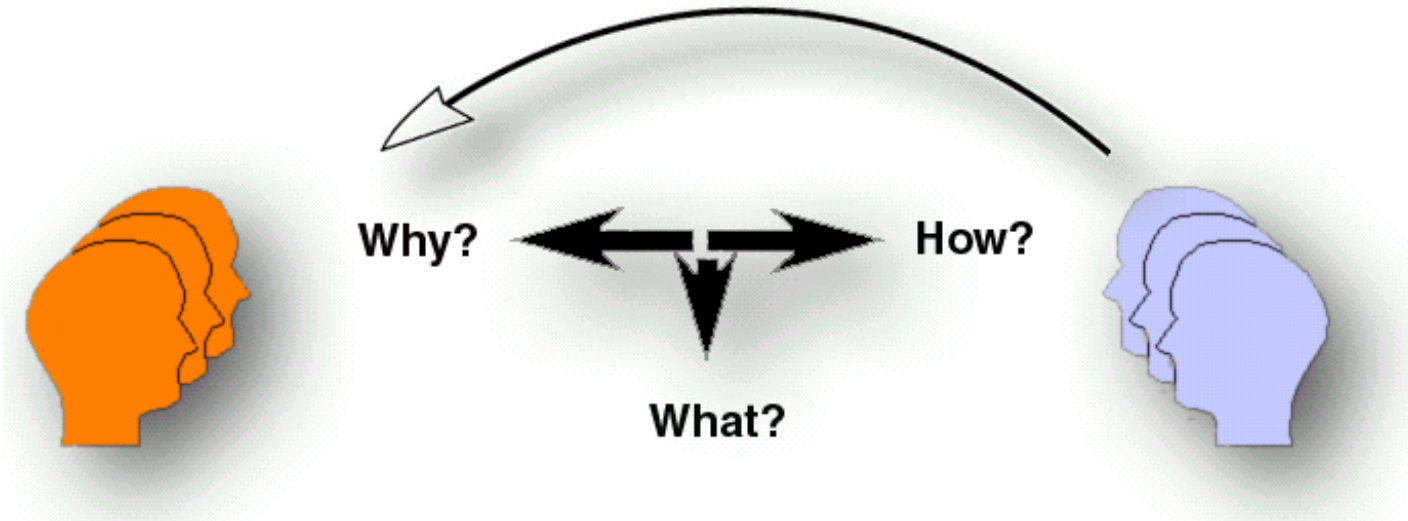
Analyse Business Area



Analyse Business Area - Overview

- A number of questions need to be answered:
  - What needs to be done?
  - What fundamental elements of the business are we focusing on?
  - What are the constituent parts of this aspect of the business?
  - What is integral to the business?
  - What principal data/information must the company work with?
  - What events trigger the start of the related business operations?
- Deliverables:
  - Business Requirements
  - Business Case
- Focus: Business perspective, not Systems

Hyperknowledge is about knowledge



Captures what individuals know

Publishes that knowledge so others can exploit it

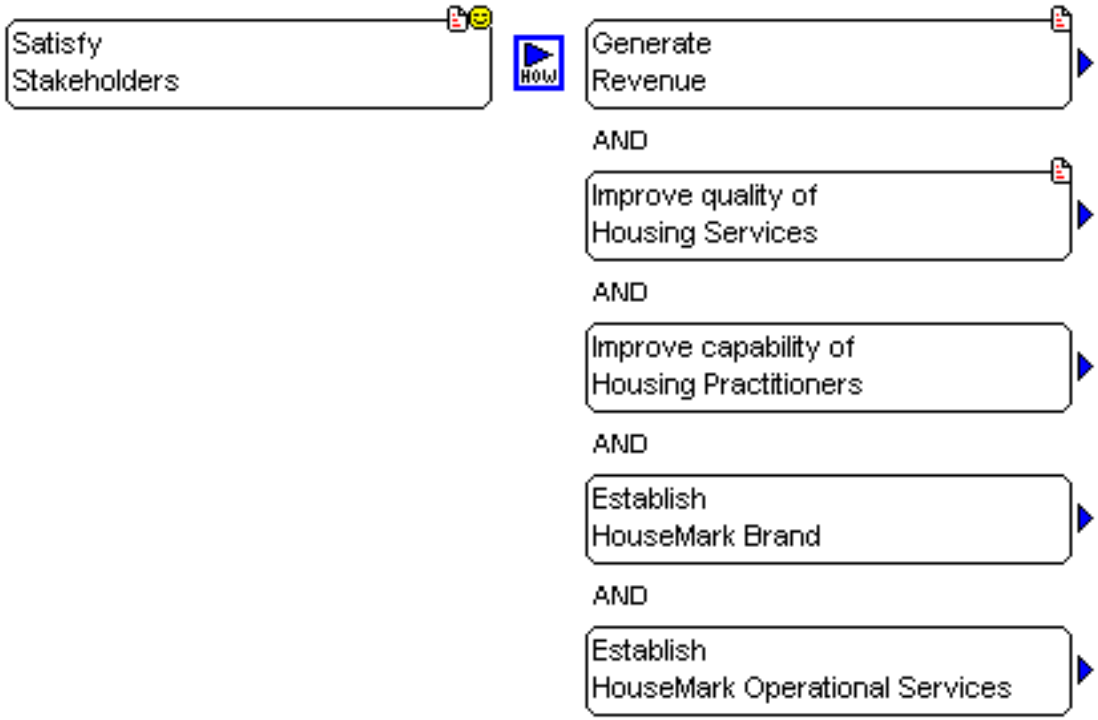
The How? Question

- Decomposition into detail
- (just as a book is broken down into a title, headings, sub-headings, etc)

Sequence is not implied by the How? direction.

Real-world example - Hyperknowledge model

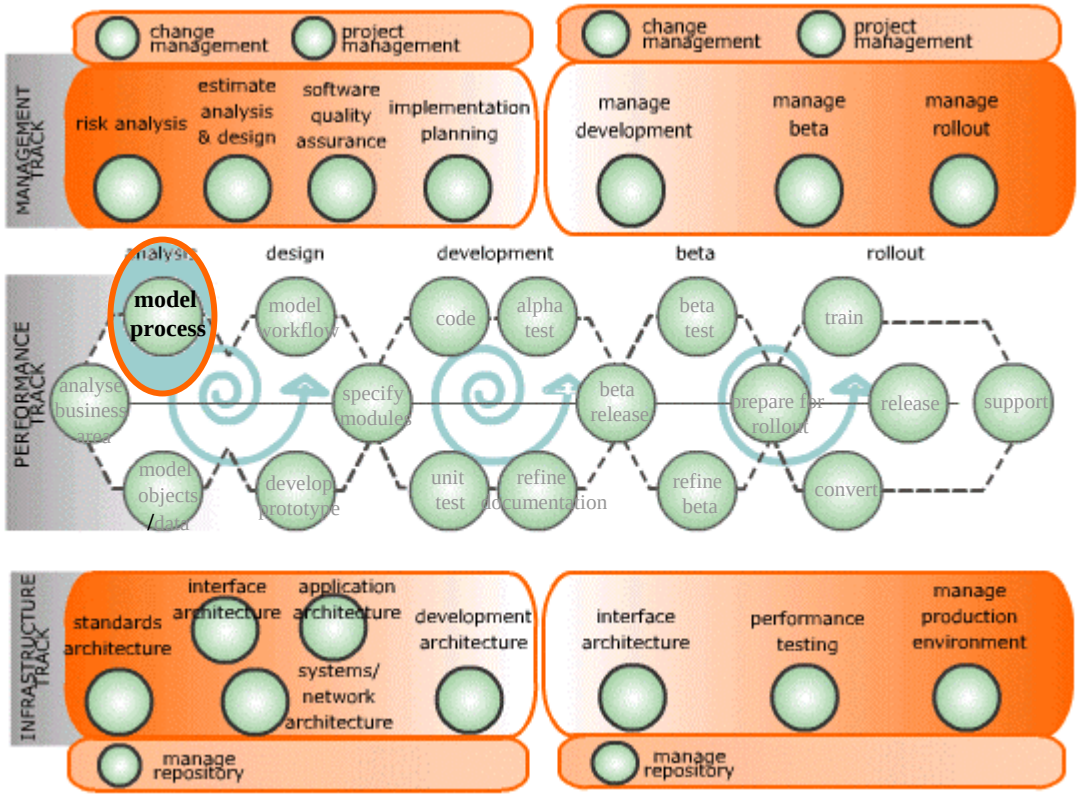
Model Overview



What lists Used

Stakeholders
Revenue
Housing Services

Model Process



Use Case Diagrams - Standards

- Actors

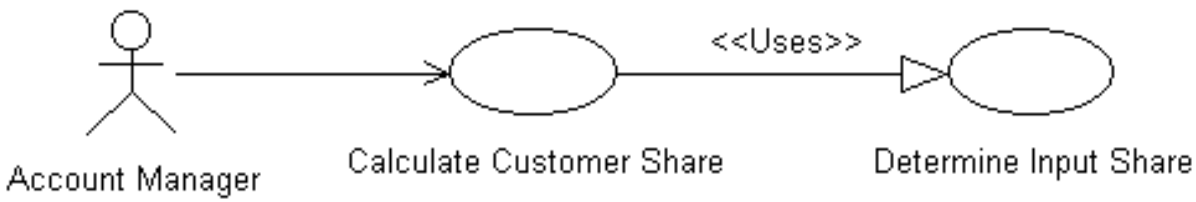
  Used to represent external systems that interface with the target system, or users of the system

- Ellipses

  Used to represent the high level screens within a system, or some complex processing used by a screen.

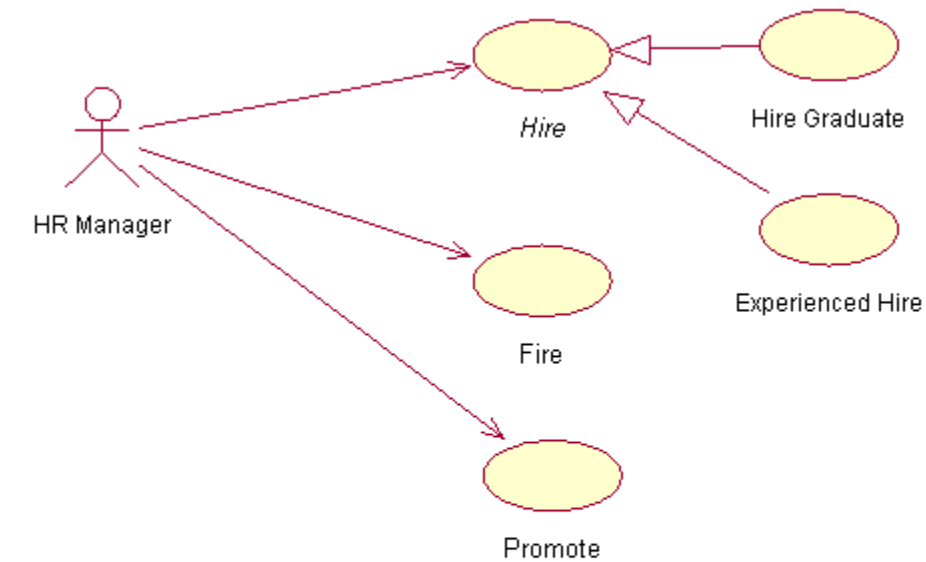
- Relationships

  Used to show how an actor interacts with a system, or a screen uses some functionality represented within the system.

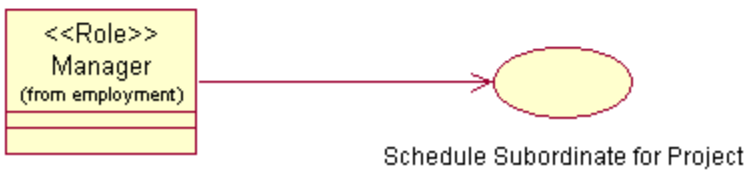


Use Case Diagrams

- Example

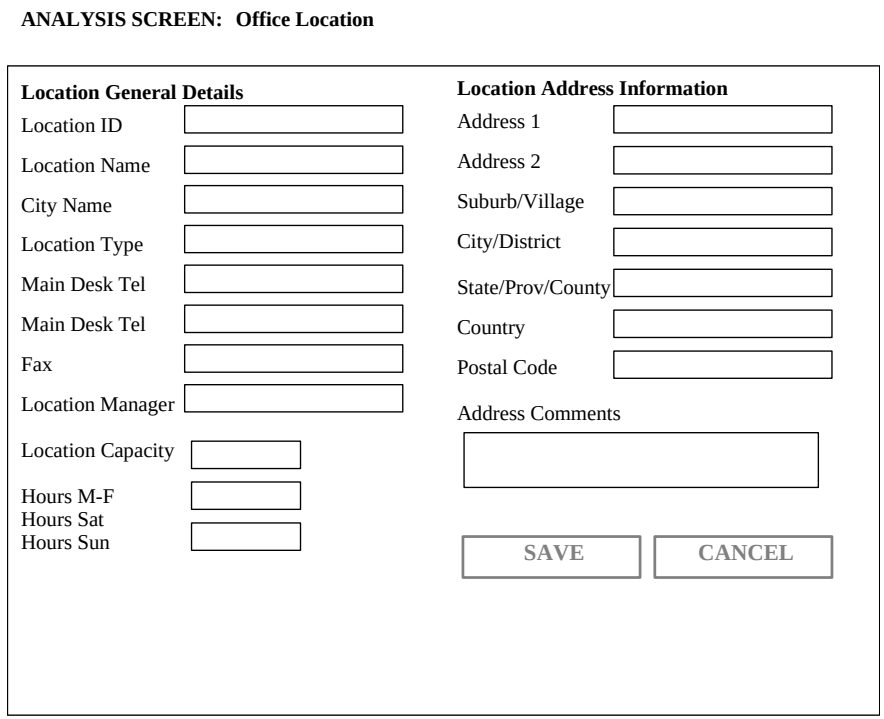


- The HR Manager can Hire, Fire and Promote staff
- Hire Graduate and Experienced Hire are specialisations of the Hire Use Case

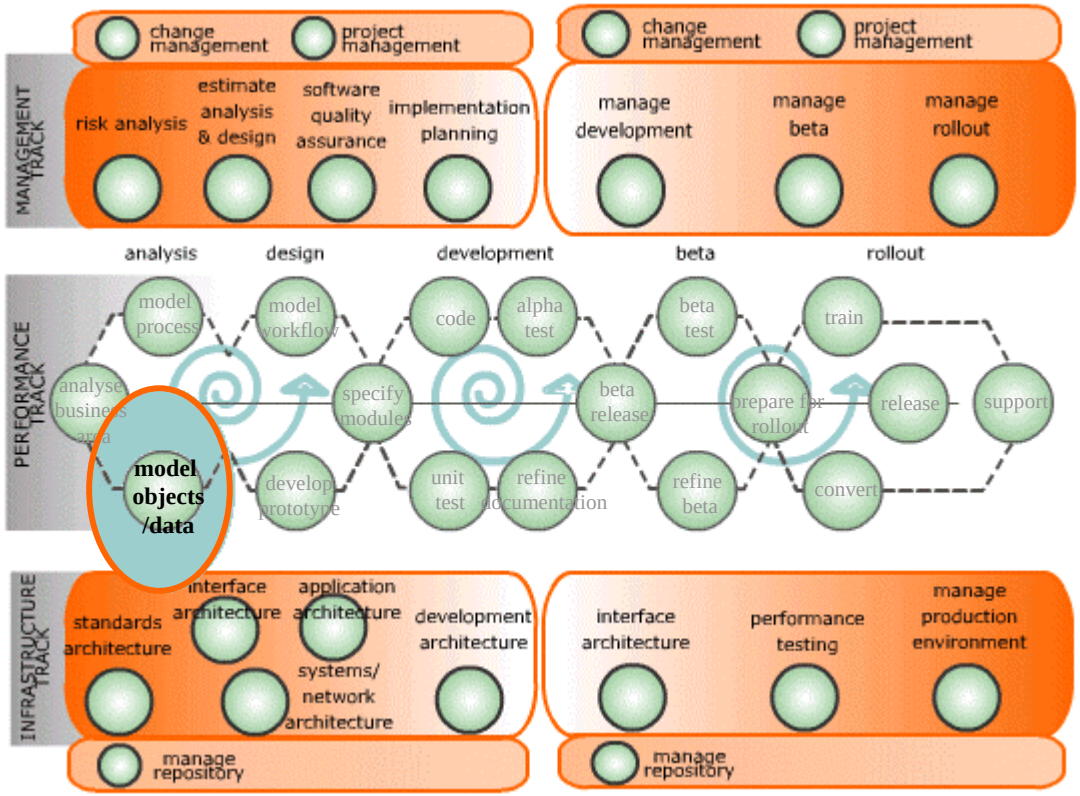


Analysis screen prototype - Example

- High Level Screen Design
  - Visually lay out key interactions
  - Clarify “whole system” behaviours
  - Do not develop detailed screens
  - Do not create ‘pleasing’ GUIs
    - The aim is to confirm behaviour, not to seek input of colours, placement of fields, or navigation preferences
  - Target Audience same as before:
    - Key stakeholders and team members
  - Feeds in to the Proof of Concept



Model Objects/Data



Object modelling - What?

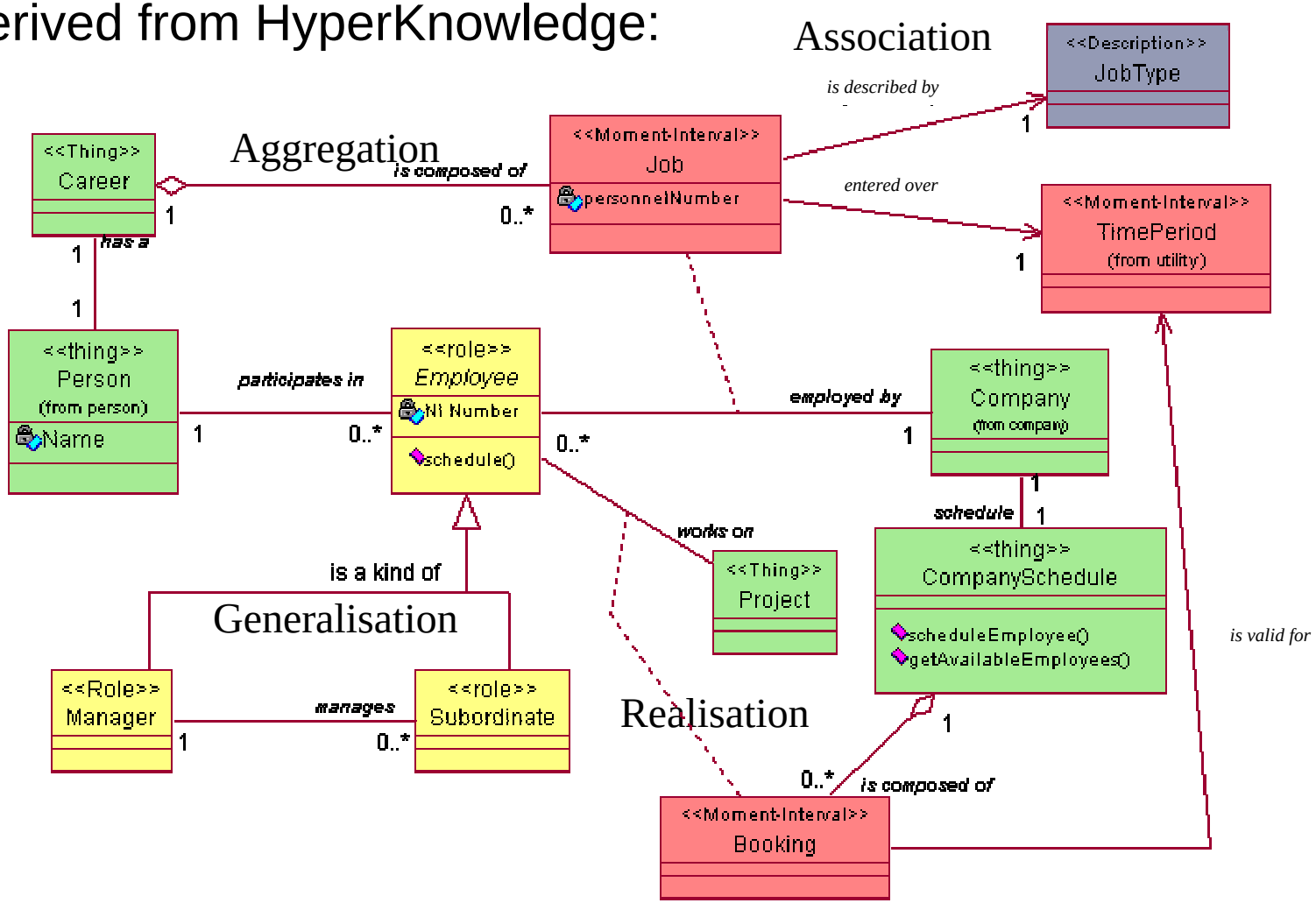
- Make a conceptual model based on the detailed requirements captured in the process modelling
  - We used HyperKnowledge to capture these business requirements
- Use the “Unified Modelling Language (UML)”
  - to express objects in the systems and the relations between them;
  - to express the dynamic behaviour of the system

Object modelling - Why?

- Object modelling is used to:
  - Utilise an object oriented view of the problem/solution domain
  - Develop a coherent, complete description of the ‘TO BE’ system
  - Create a reusable, stable description of data and behaviour in the system
- If not done:
  - There is no object oriented base for object programming
  - It is more difficult to iterate and reuse, since we have no docuented starting point for each object’s existence
  - We have no formal link between the code and the Business Requirements

Analysis of Candidate Classes

- Derived from HyperKnowledge:



“First Cut” Class Diagram

- The Class Diagram describes:
  - Types of objects in the system
  - Static relationships between objects
- Inputs to the process:
  - Hyperknowledge Models
  - Use Cases
    - Developed previously in the Analysis Phase
- Initial classes are predominantly "Entity" type classes
  - i.e. they represent tangible real-world objects
    - Use the nouns of high level nodes in the Hyperknowledge model
    - Other, "Control" type classes tend to emerge later in the process

“First Cut” Class Diagram

Classes vs Objects

- Objects of the class have a real-world equivalent
  - Objects represent a real-world object in the problem domain.
    - Example: This car, That parking lot, The clerk named “Tom”,
  - Classes describe these objects
    - Example: Cars all have number plates, colours, are left- or right-hand drive
- Objects of the class have identity
  - An object is uniquely identified merely by its existence.
    - Example two different locations in the same city would still be two unique locations.
    - **Question** - Do two car parks at the same location also have unique identities?
  - A class defines what kind of existence the objects have
    - Example: a car is different to a minibus - a minibus usually has specific licence regulations

“First Cut” Class Diagram

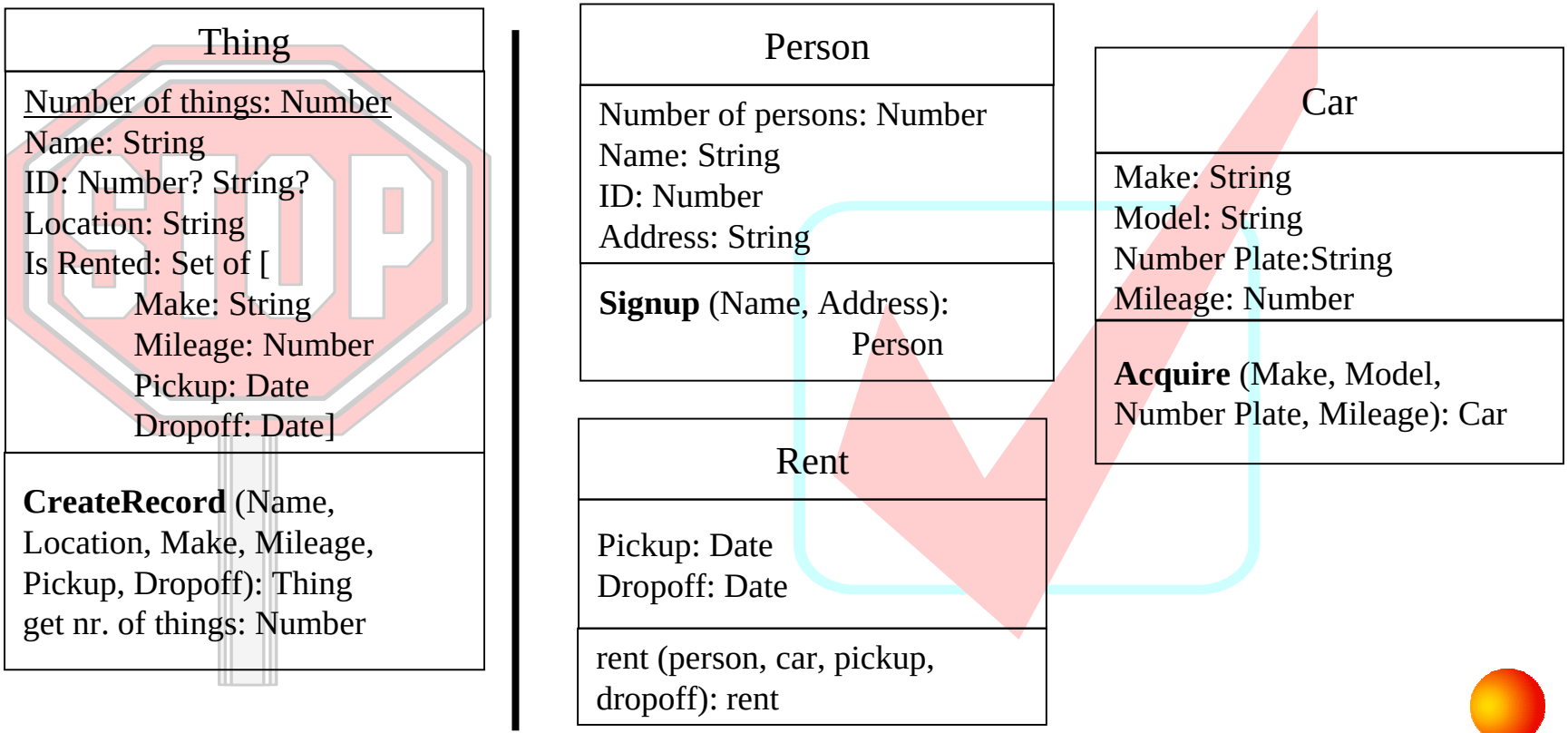
What information to include about each class

- Objects of the class have a certain state and behaviour
  - Having a state means that objects maintain some kind of information.
    - Example for a Car: Number Plate, Make, Model, repair history, etc
  - Having behaviours means that an object must be able to offer services to other objects.
    - Example for Car: Rent, Change Location, etc.
- Objects of the class have a well defined life cycle
  - There is a certain moment when the object becomes to existence in the system.
    - Example: A Car is bought by the company.
  - There is a certain moment when an object disappears from the system.
    - Example: A Car is sold or written off (wrecked).

“First Cut” Class Diagram

Define the class boundaries

- Classes should have well-defined boundaries

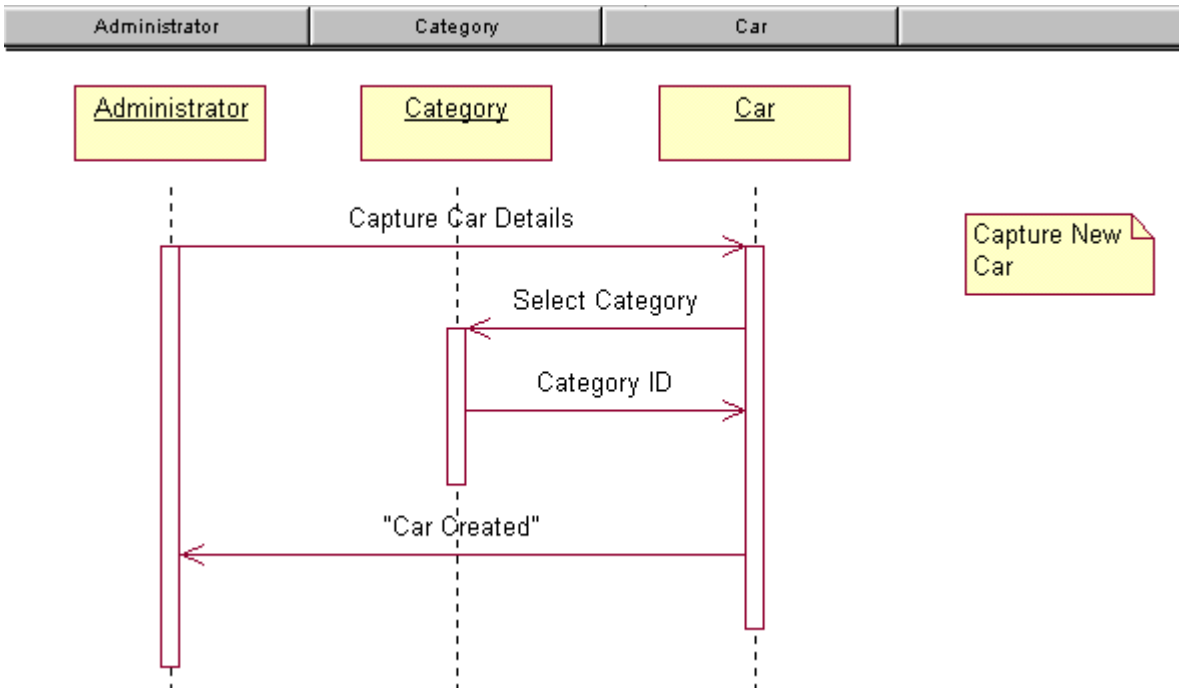


Sequence Diagram

- Graphical view of a scenario that shows object interaction in a time-based sequence
- Establishes the roles of objects and help provide essential information to determine class responsibilities and interfaces
- Closely related to Collaboration Diagrams and both are alternate representations of an interaction
- Shows time-based object interaction while Collaboration Diagrams show how objects associate with each other

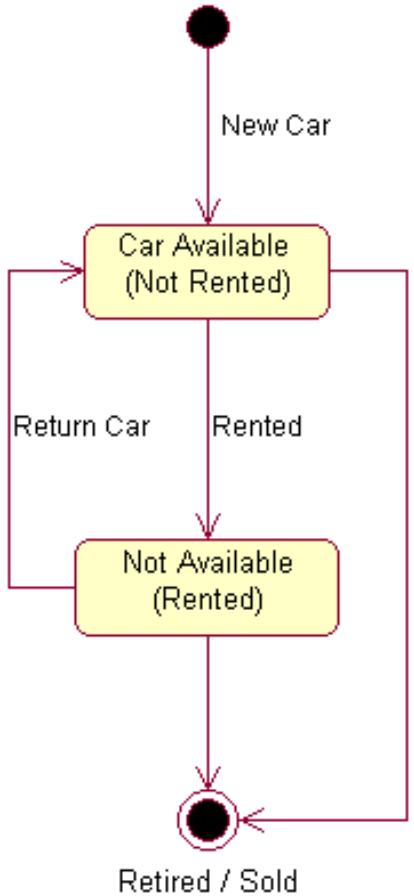
Sequence Diagram

- Two dimensions
  - Vertical placement represents time
  - Horizontal placement represents different objects



State Transition Diagram

- Formal specifications of the states of a particular class
  - Including how transitions between states occur
  - This Diagram describes the states of a car
- Arrows are transitions from the start state to the end state
- Complex behaviour should be modelled using a State Transition Diagram
- Derived from the Sequence Diagram

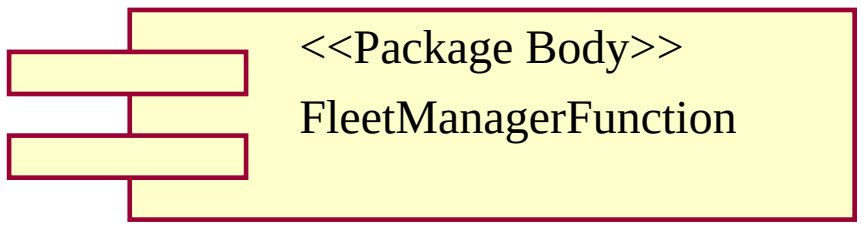


Component Diagram

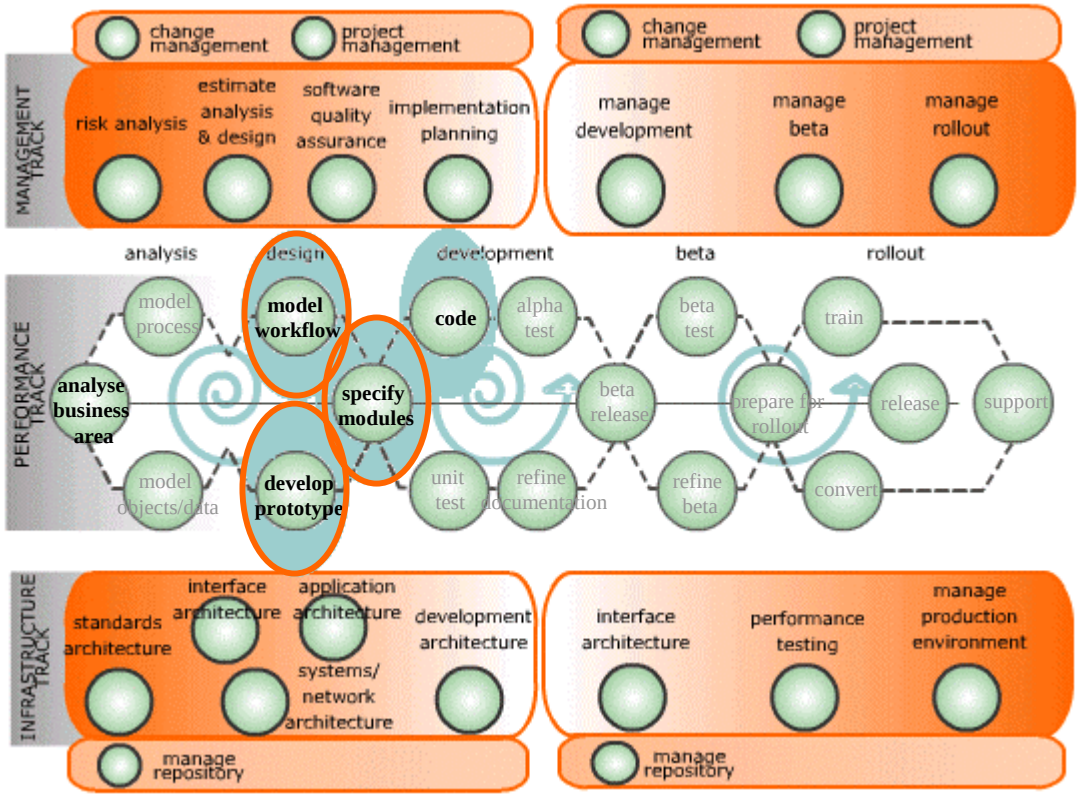
- Shows the organisations and dependencies among software components
- Show the externally visible behaviour of the components by displaying the interfaces of the components
- Calling dependencies among components are shown as dependency relationships between components and interfaces on other components
- Not always used. Simple systems may not need to create a component diagram

Component Diagram

- Component
  - Represents a module with a well-defined interface
  - Used to show interface and calling dependencies among software modules
  - Also show which components implement a specific class
  - Each software module is represented by a Component in the model



Model workflow

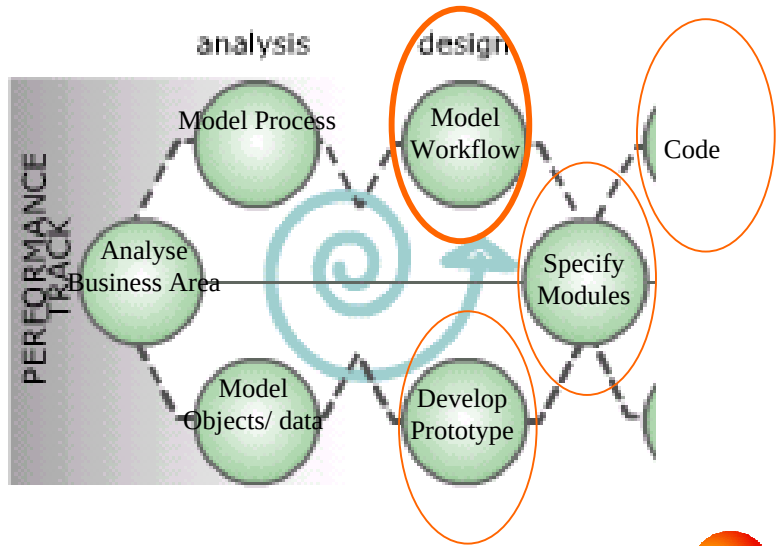


Workflow Modelling and User Interface Design

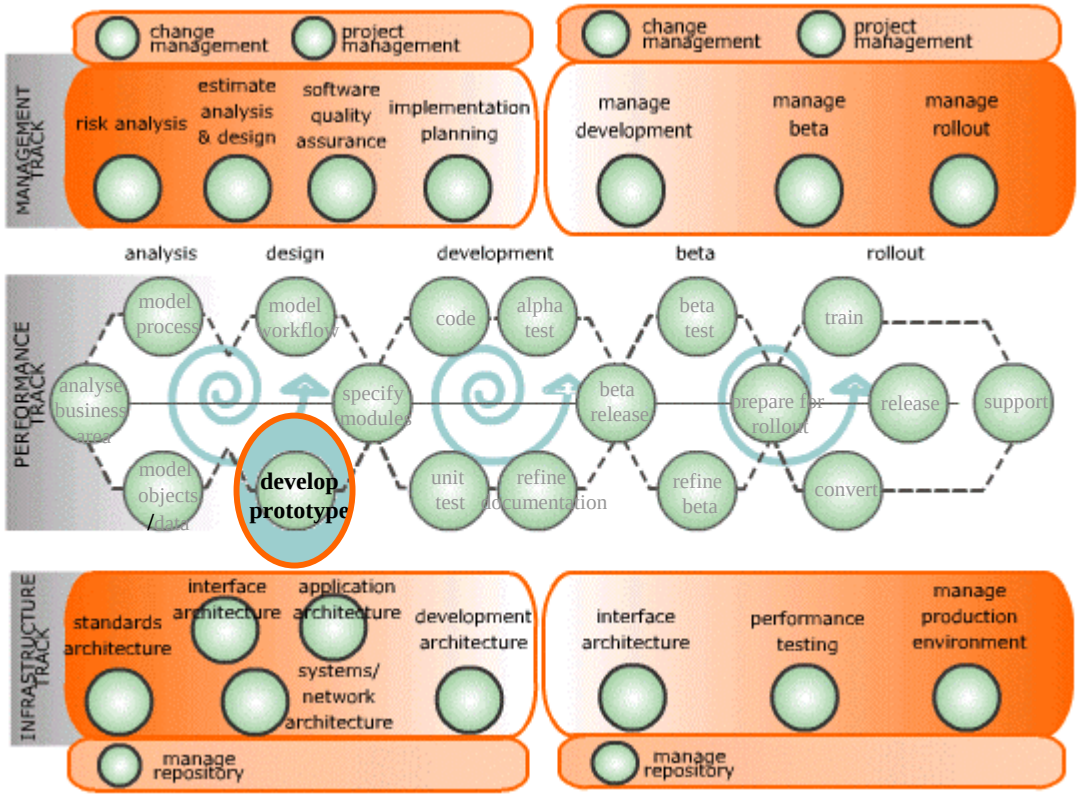
- Workflow modelling is:
  - necessary to define a complete, effective, and efficient support mechanism for the interactions of the system including the target end-users
  - used to detail the various available paths between interactions
  - done in a Object Oriented environment through User Interface Design
- User interface modelling will
    - help to drive the design
    - provide validation of design decisions

User Interface Design - What

- Interface Design is part of Model Workflow
- Four areas of design result in a holistic solution:
  - **Navigation**: How the user navigates around the application
  - **Screen Design**: The actual appearance of the screen
  - **Experiential Design**: Considers the whole experience of interacting
  - **Usability Testing**: Is the application usable and easily understood.



Develop Prototype



Why do Prototyping?

- A model of the application is created for the purposes of
  - Further eliciting and understanding requirements for the application
  - Evaluating design alternatives
  - Simulating performance
  - Demonstrating product features or project progress to the user
- Prototyping *is not* the same as Rapid Application Development

What is a Prototype?

- The prototype may simply consist of a series of non-functional screen designs, or it may be a working application
- Implementation of the prototype varies considerably from project to project due to:
  - **Software Architecture**. A new, untried architecture will have a number of unknowns that require testing
  - **Client factors**. A working prototype is an extremely effective way of internally selling a project
  - **System domain**. An interactive on-line system provides more 'show" for the time invested than a batch processing system with few screen
  - **System scale**. A small single user system is unlikely to require a working prototype

Prototype: Deliverables

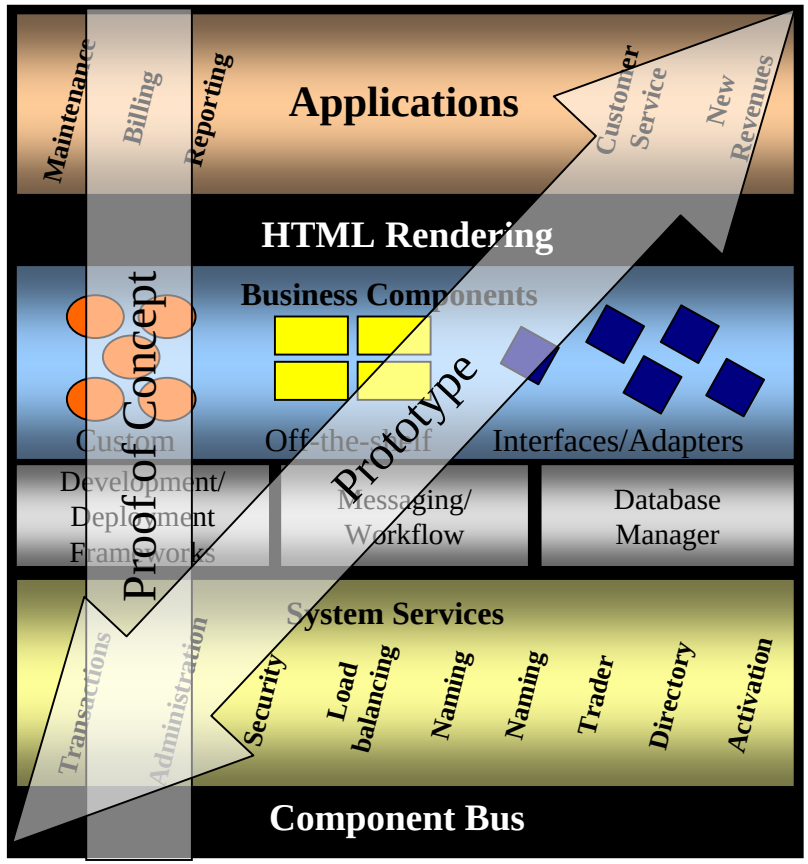
- A prototype should
  - Give the target audience a realistic feel of the intended application
  - Help the team learn more about the final application through developing the prototype
  - Sell the project to the Client
  - Offer similar functionality to the final application
- The benefits of developing a prototype are:
  - Development problems can be caught early on in the project.
  - The application architecture can be extensively tested prior to implementation.
  - The client sees a “product” earlier in the development lifecycle.
  - A successful prototype can get buy-in to the project.
  - Any changes required by the client can be made early on in the project.

Technical Prototype

- A technical prototype is usually limited to a small piece of the functionality of the full system implementation
- The aim is not to create reusable classes, but to measure the viability of the architecture against the requirements
- A technical prototype naturally follows on from the Architecture Selection and Proof-Of-Concept work carried out in Analysis and Design Infrastructure

Technical Prototype

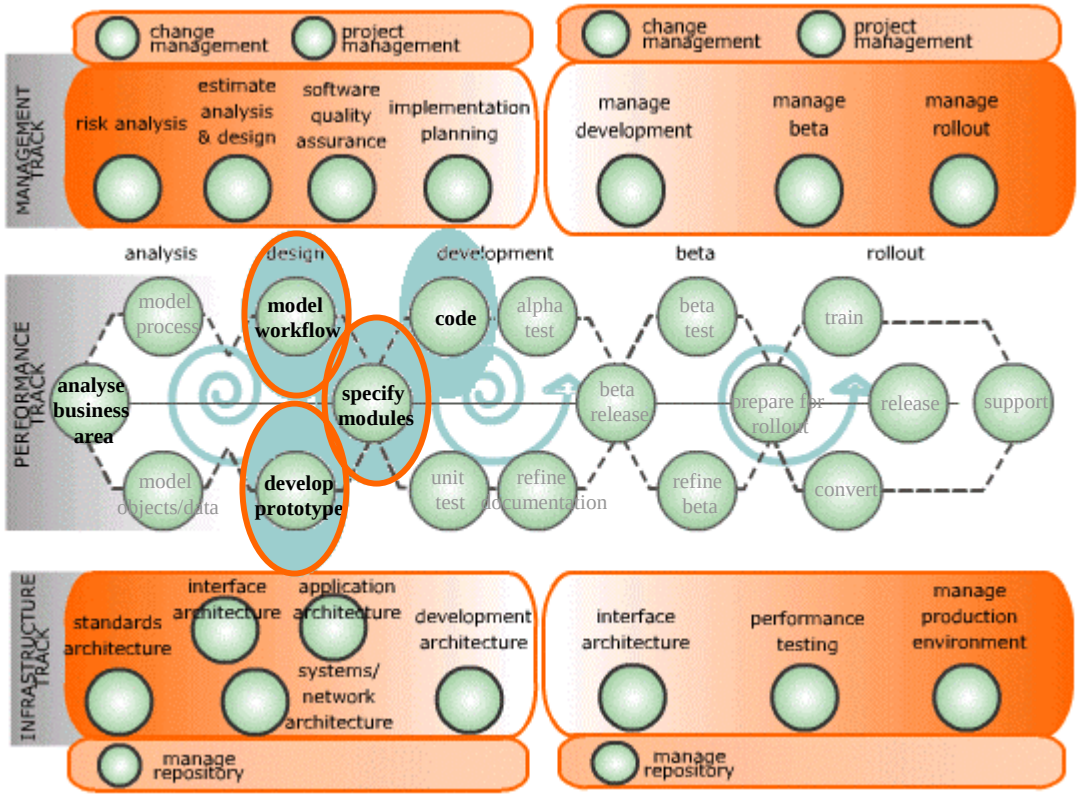
- Demonstrates all aspects of the architecture
  - Slices through all layers
  - Follows data through layers
- A technical prototype should provide answers for the following questions:
  - Does the proposed application architecture satisfy the functional and non-functional requirements?
  - What skills should be strengthened on the engagement team prior to system build?
  - What changes can be made to the development environment to facilitate the system build?



Determine Prototype Scope

- The scope of the prototype may depend on several factors:
  - Is it a technical prototype?
  - Will the prototype require any experiential or GUI design?
  - Will the prototype form the basis of further development work (e.g. is it evolutionary, or throw away)?
- A reduced and simplified version of the Hyperknowledge model can be used to scope the domain of the prototype
- In the case of a technical prototype, a design specification should also be developed
- A demonstration of the prototype must be scripted

Specify Modules



Specify Modules

- Create a 'blueprint' which will communicate all the specifications to:
  - Users: For Approval
  - Data Management or Information Management group for planning
  - Project Implementation team for construction, testing and installation
- Outputs of this segment:
  - Technical Detailed System Specification
  - Test Strategy
  - Rollout Strategy

Detailed System Specification

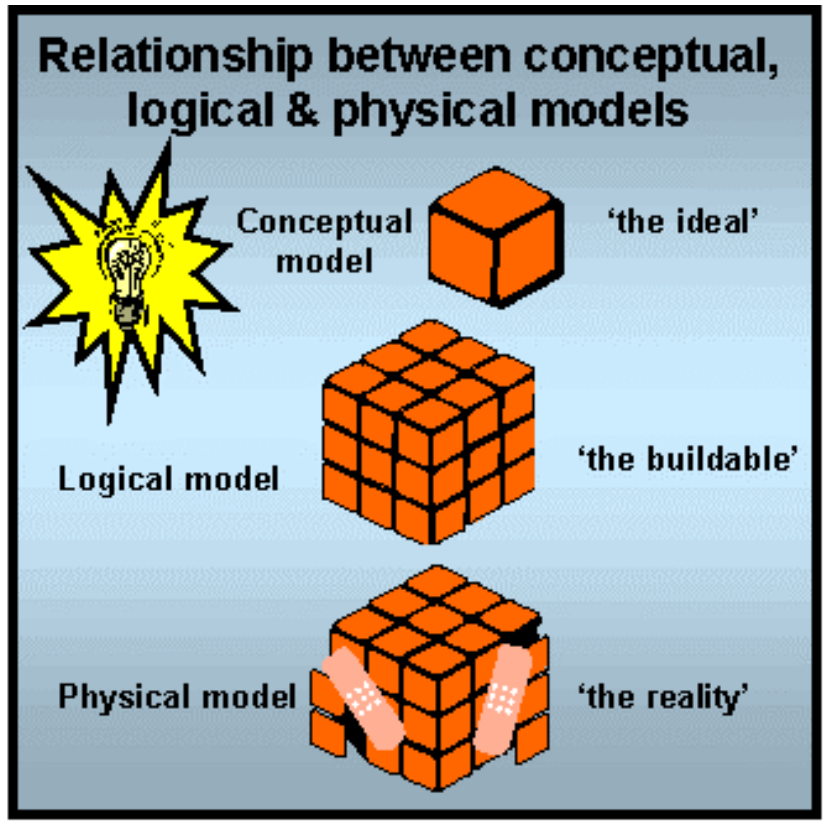
- Collection of documentation that describes fully the application to be coded
- The documents that make up the System Specification include:
  - Physical Data Model
  - User Interface Specification
  - System Controls Documentation
- A Detailed System Specification should:
  - Accurately describe the application
  - Completely describe the application
  - Be presented in a standard format that is interpretable by a trained software developer

Detailed System Specification

Specs

Construct Physical Data Model

- The data models are refined into a precise specification for development
- Describes precisely the tables (rows, columns), referential integrity and domains of the target system database
- Might also contain customisation that enables the software to run better on the chosen platform

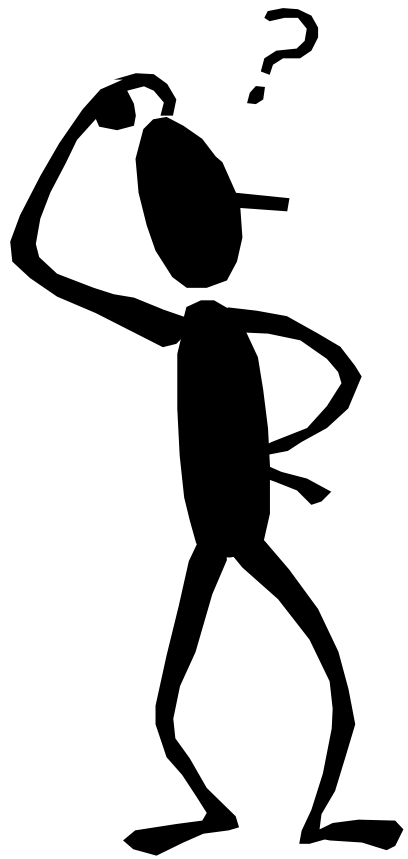


Testing

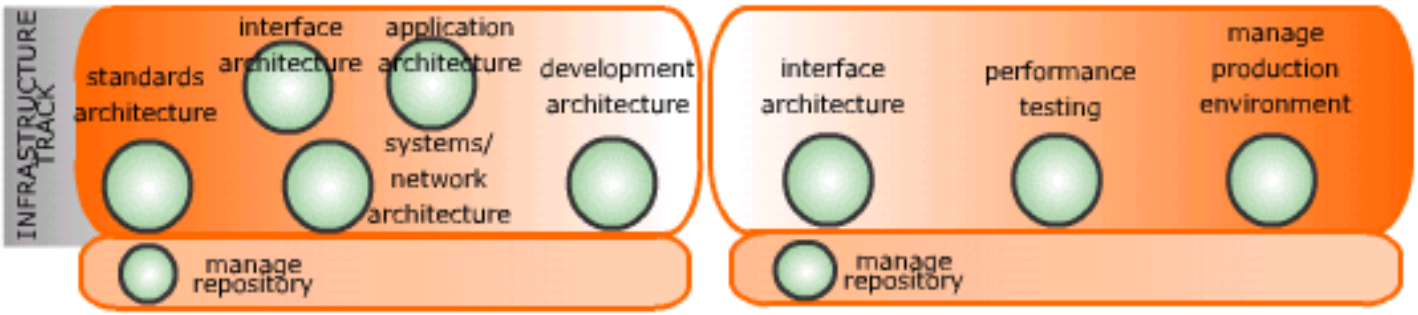
Rollout

ARTHURANDERSEN

Questions?

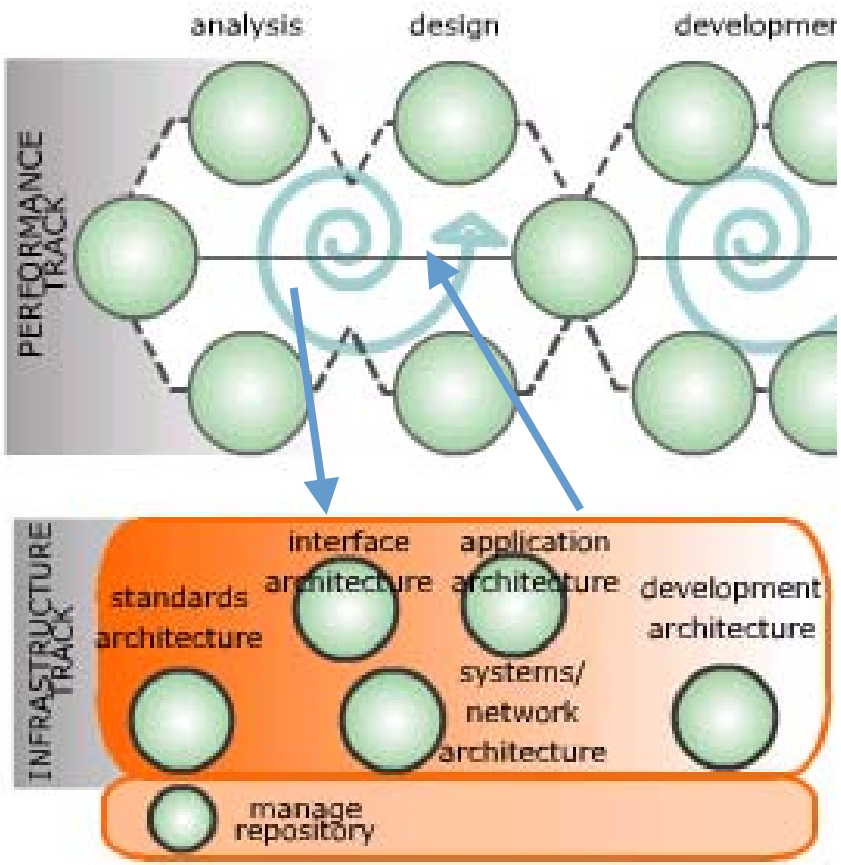


What is the Infrastructure Track?



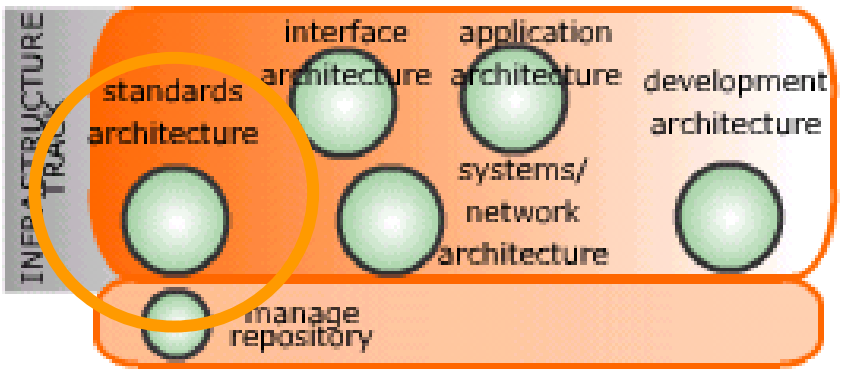
- Infrastructure Track activities are the responsibility of Infrastructure Team members
- Provides a grouping of activities that will be undertaken by the infrastructure team of developers, network administrators, database administrators, etc
- Does not preclude the same person or team having involvement in other track

Analysis and Design Infrastructure



- These activities are crucial to:
  - The quality of the finished product
  - The reduction of risk throughout the process
  - The creation of an application that fulfils the non-functional requirements of the system
  - The identification and development of components
- They directly impact the details of the design itself
  - (iterative process)

Standards Architecture

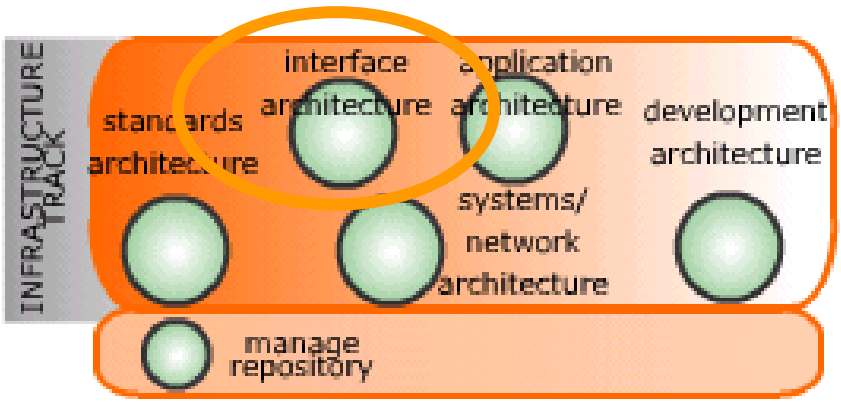


- Enforces consistency and common understanding
- Conventions for team members
  - Naming Standards and Templates
  - Notation Standards for diagramming
  - Coding Standards
  - Reuse Standards
  - Tools Standards
  - Business Terminology Standards

Standards Architecture: why?

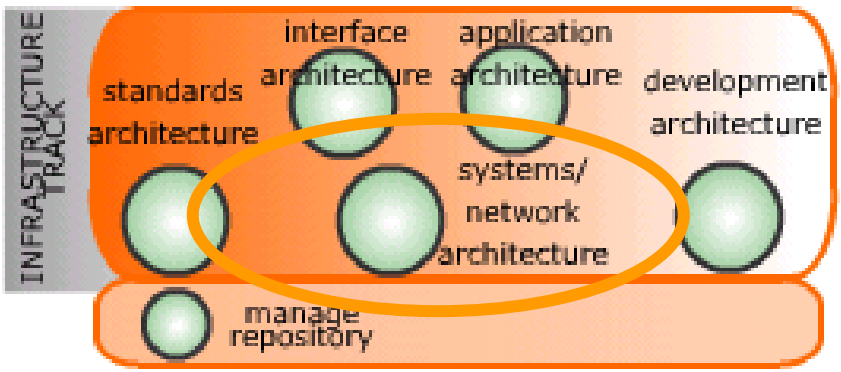
- The Project Standards deliverable should detail the standards approach for all areas of the project, and should take into account the working practises of the client
- Activities:
  - Identify Client Standards
  - Define System Component Naming Standards
  - Define Documentation Standards
  - Define Business Terminology Standards
  - Define Development Standards
    - Maybe bind into one “Developer’s handbook”?
    - Programming Style
    - Code Documentation
    - User Interface
    - Report Formatting
  - Define Build Environment Standards

Interface Architecture



- Data Exchange Strategy
- Current network infrastructure
- Design, build, and test
- Minimum information to be captured on each interface
  - Source system
  - Target system
  - Frequency
  - Duration
  - Time
  - Responsibility for interface support
  - Network requirements
  - Version

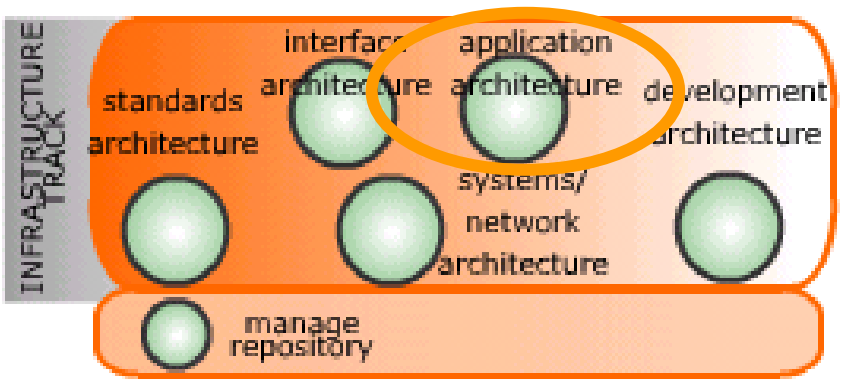
Systems/Network Architecture



Systems/Network Architecture

- Current network architecture
- Technical requirements
- It is required to:
  - Enable users to connect to the system
  - Enable systems to connect to the system, and the system to connect to other systems
- Activities
  - Reference Architecture
  - Define Security Measures

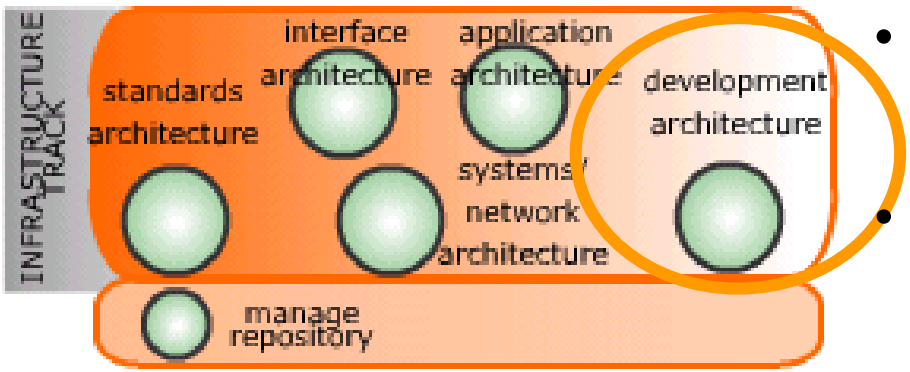
Application Architecture



Application Architecture

- Identification of subsystems on services
- Well defined public interfaces
  - weak coupling - information hiding
  - client-server versus peer-to-peer
- Layered construction
  - strong versus weak levels
  - open versus closed levels
- Covered in more detail later

Development Architecture



Development Architecture

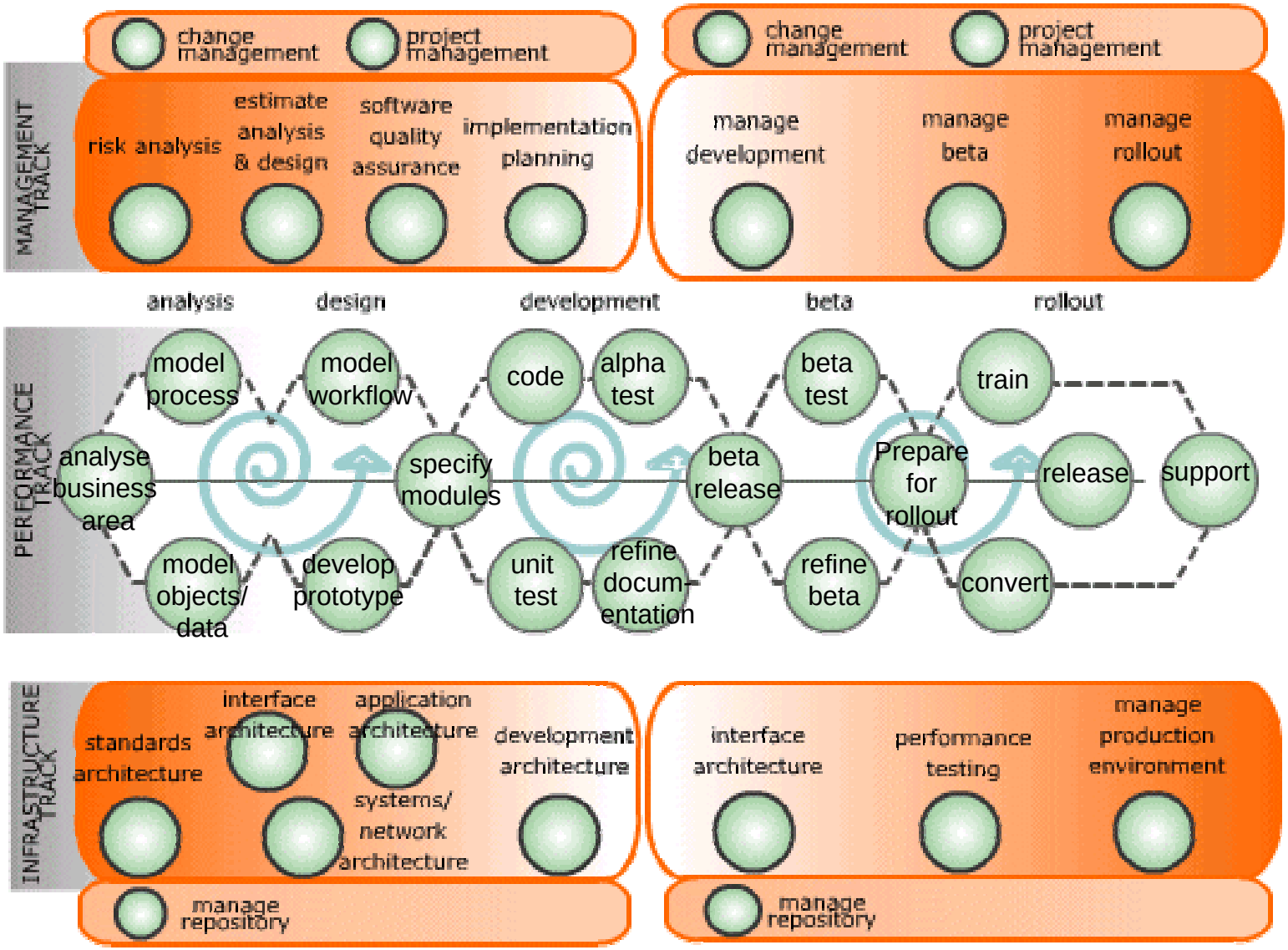
- Select and implement development tool(s)
- Select and implement additional object/class/module libraries
- Configure the project version control system for development
- Select, implement, and configure other development/ management tools.

Development Architecture

Development Architecture

- The development architecture refers to the development tools, configuration management tools, object libraries etc. needed to construct the software
- Activities
  - Identify Development Tools
  - Obtain Development Tools
  - Train Developers

RAD*Frame*™ - recap



RAD*Frame*™ - recap

- Rapid Application Development
- A Framework, not a Methodology
- A Synthesis of Proven Methods
- Evolution, not Revolution
- Software Engineering, not Hacking
- Integration of Process and Technology

RAD*Frame*™ - recap

- Production of an engineered information system integral to the needs of the organisation
- Delivery of an end product which exceeds user expectations, performs above the required level, and finishes within budget
- Integrate Process Technology perspectives, and workflow to provide an innovative solution
- Capture and use knowledge to optimise the current development and improve future development
- Rapidly design, develop, and deliver an application without sacrificing reliability, reuse, or maintainability

ARTHURANDERSEN

Questions?