5 Object Oriented Analysis

5.1 What is OOA?

5.2 Analysis Techniques

5.3 Booch's Criteria for Quality Classes

5.4 Project Management and Iterative OOAD

5.1 What is OOA?

- How to get understanding of what we want to build
- Many definitions try to distinguish analysis from design
  - Discovery - Invention
  - What? - How?
  - Physical - Logical
  - Analyst - Designer

5.1 Discovery vs Invention

- Discovery (Analysis)
  - Requirements
  - Physical Objects
  - Terminology
  - Constraints
  - User expectations
  - System boundaries
- Invention (Design)
  - Logical design
  - Data model
  - User interface
  - Control structure
  - Object definition
  - Algorithms
  - Interface to platform

Discovery: find the things which are fixed
Invention: find a possible solution

5.1 What? vs How?

- What? (Analysis)
  - Requirements
  - Terminology
  - System boundaries
  - Constraints
  - User expectations
  - User interface
  - Object definition
- How? (Design)
  - Logical design
  - Data model
  - Control structure
  - Algorithms

Not clear where to put e.g. User interface or object definitions

5.1 Logical vs Physical

- Logical (Analysis)
  - Requirements
  - Terminology
  - Constraints
  - Logical design
  - Data model
  - Control structure
  - Algorithms
- Physical (Design)
  - Platform API
  - User interface
  - Physical objects
  - Hardware API
  - Data storage API

Many "design" activities are "analysis" in this scheme

5.1 Analyst vs Designer

- Analyst
  - Gather requirements
  - Design solutions
  - Implement
  - Test
- Designer
  - Gather requirements
  - Design solutions
  - Implement
  - Test

When the analyst does it its analysis
When the designer does it its design
Hierarchical

5.2 Analysis Techniques

- Ad-hoc
- Noun lists
- CRC cards
- Use cases

5.2 Ad-hoc Analysis

- Analysis on-the-fly while implementing
  - Simple problems
  - Objects, methods and behaviour obvious
- Probably the only analysis method in HEP?
- Works well with a good "analyst/designer"
- Works miserably when the problem is too difficult for the "analyst"
- Hard to do in collaboration

5.2 Noun List Analysis

- Identify nouns, adjectives, verbs from e.g. requirements documents
  - nouns → objects?
  - Verbs → methods?
  - adjectives → object variations? → abstractions?
- Fight blank page syndrome
- Depends on quality of existing documentation
- Too concrete, difficult in large projects

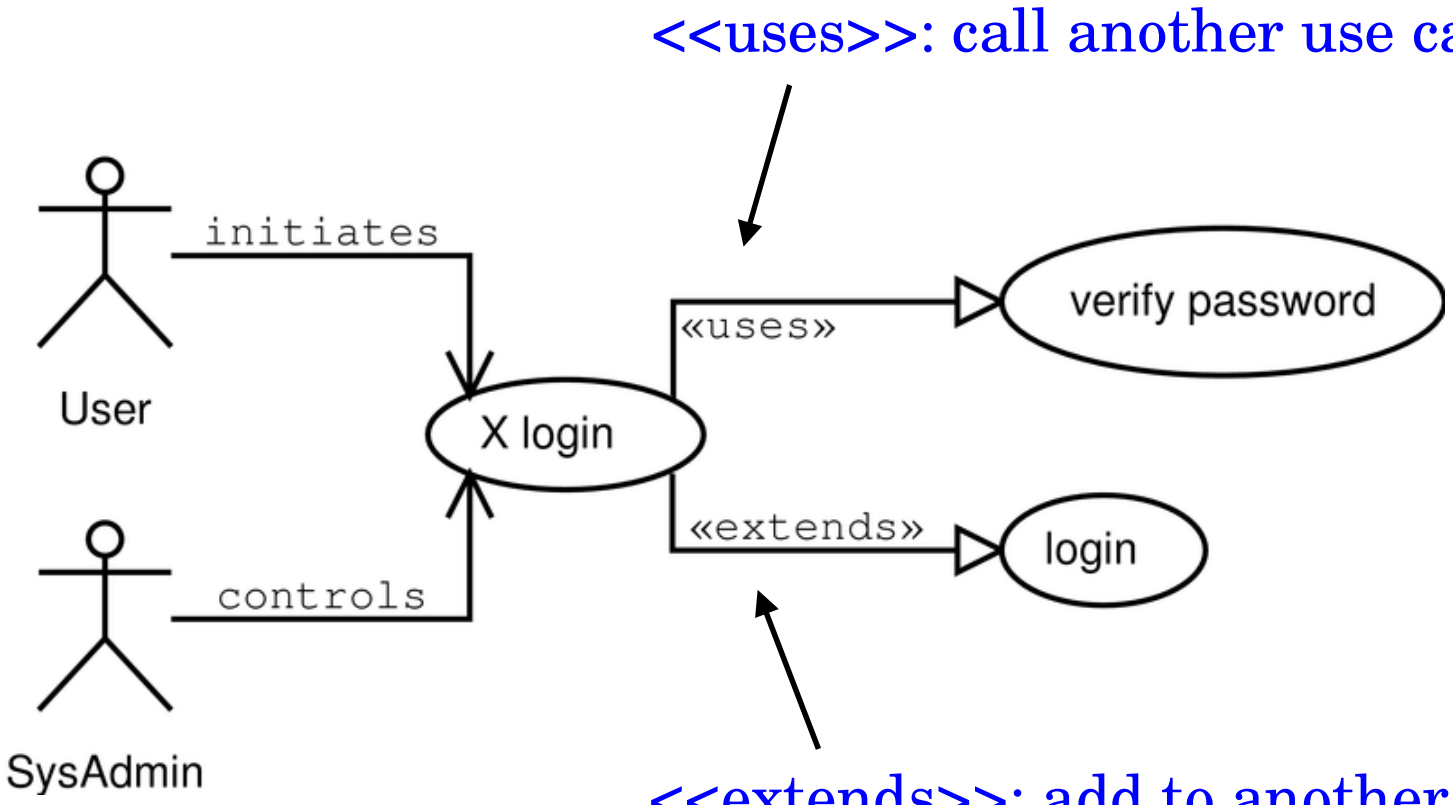
5.2 Use Case Analysis

- Start from requirements
- Describe response of system to events
  - Normal flow of action
  - Error and exception handling
- Can implement tests to check use cases
- Can be quite formal
  - UML diagrams
  - Nested use cases

5.2 Use Case Template

Use Case:	The name
Actors:	User or other systems which trigger an event
Summary:	Abstract
Pre-conditions:	What must be true before the use case can be considered; possibly other use cases
Description:	Interaction between actors and system, normal and errors or exceptions
Post-conditions:	What is true after the use case is done, i.e. the state of the system
Related:	List other related use cases

5.2 Use Case in UML



Notation similar to class inheritance, but meaning is different

5.2 Use Case Summary

- Create use cases from requirements
  - Response of system to events
  - Normal and errors/exceptions
- Leads to tests
  - Map use cases to tests
- Use cases are not designs
  - That's how you manage to satisfy the tests derived from use cases

5.2 CRC cards

- CRC = Class Responsibilities Collaborators
- Aids brainstorming to find classes/objects
- Index cards note in pencil
  - Front: class name, responsibilities
  - Back: collaborators, variables, techniques
- Group discussion
  - Find or move responsibilities, find/rename/split classes, identify collaborators and techniques

5.2 CRC Cards

- What do we get? Better understanding of
  - classes and collaboration
  - class interfaces
  - message flow
  - implementation ideas
  - common view of project in the group
- Results will need verification and reworking
  - Code and tests

5.3 Booch's Criteria for Quality Classes

- When is an class/object well designed?
- Booch says look for
  - Coupling
  - Cohesion
  - Sufficiency
  - Completeness
  - Primitiveness

5.3 Class Coupling

- "Strength" of associations between classes
  - strong coupling → individual classes hard to understand, correct or change
  - tension with inheritance which couples classes
  - tension with complexity of a class
- Relation with other principles
  - couplings within or across packages different

5.3 Class Cohesion

- Connections between elements of a class
  - elements, i.e. class methods, work together to provide well-defined behaviour
  - no unrelated elements or "coincidental cohesion"
- Examples:
  - ThreeVector and transformations (rotation, boost, translation) are separate classes
  - data handling and algorithms in Athena separate

5.3 Class Sufficiency

- Class provides enough characteristics of an abstraction to allow meaningful and efficient interaction
  - Its about modelling some concept via a class
- Example
  - Particle: has many aspects
    - 4vector, charge, spin, other quantum numbers

5.3 Class Completeness

- Interface of class captures all meaningful characteristics of an abstraction
  - Sufficiency → minimal useful interface
  - Now want to cover all aspects of a concept
  - Class should be widely useable
- Example
  - Particle again:
    - relations with other particles, combinations
    - vertices, production, decay, operations

5.3 Class Primitiveness

- Primitive operations efficiently implemented only with access to representation of abstraction, i.e. the class
- Should only provide primitive operations
  - keeps the interface clean+tidy
- Example
  - ThreeVector provides operations +, -, * etc.
  - but no operations with collections, these are left to the users/clients to implement

5.3 In Different Words ...

- Reuseablity
  - behaviour useful in many contexts?
- Complexity
  - difficulty of implementation?
- Applicability
  - is behaviour relevant to the class it is part of?
- Implementation Knowledge
  - implementation depends on class details?

5.3 Object and Class Naming

- Objects → proper noun phrases:
  - vector, theVector, dstarVector
- Classes → common noun phrases:
  - ThreeVector, Particle, LorentzRotation
- Modifier operations → active verbs
  - draw, add, rotate, setXX
- Selector operations → verbs imply query
  - getXX, isOpen

5.4 Iterative OO Analysis and Design

- The development process → project management
  - Ad-hoc
  - Milestones
  - Iterative
- There is always a development process
  - If not explicit probably ad-hoc random walk
  - OOAD leads to an explicit development process

5.4 Ad-hoc Project Management

- Small projects
  - Little requirements gathering
  - Quick coding
  - Frequent problems, but fixed quickly too
- Doesn't scale well to larger projects
  - Need coordination between several (many) people
  - Need realistic schedules
  - Need reliable estimators of project progress

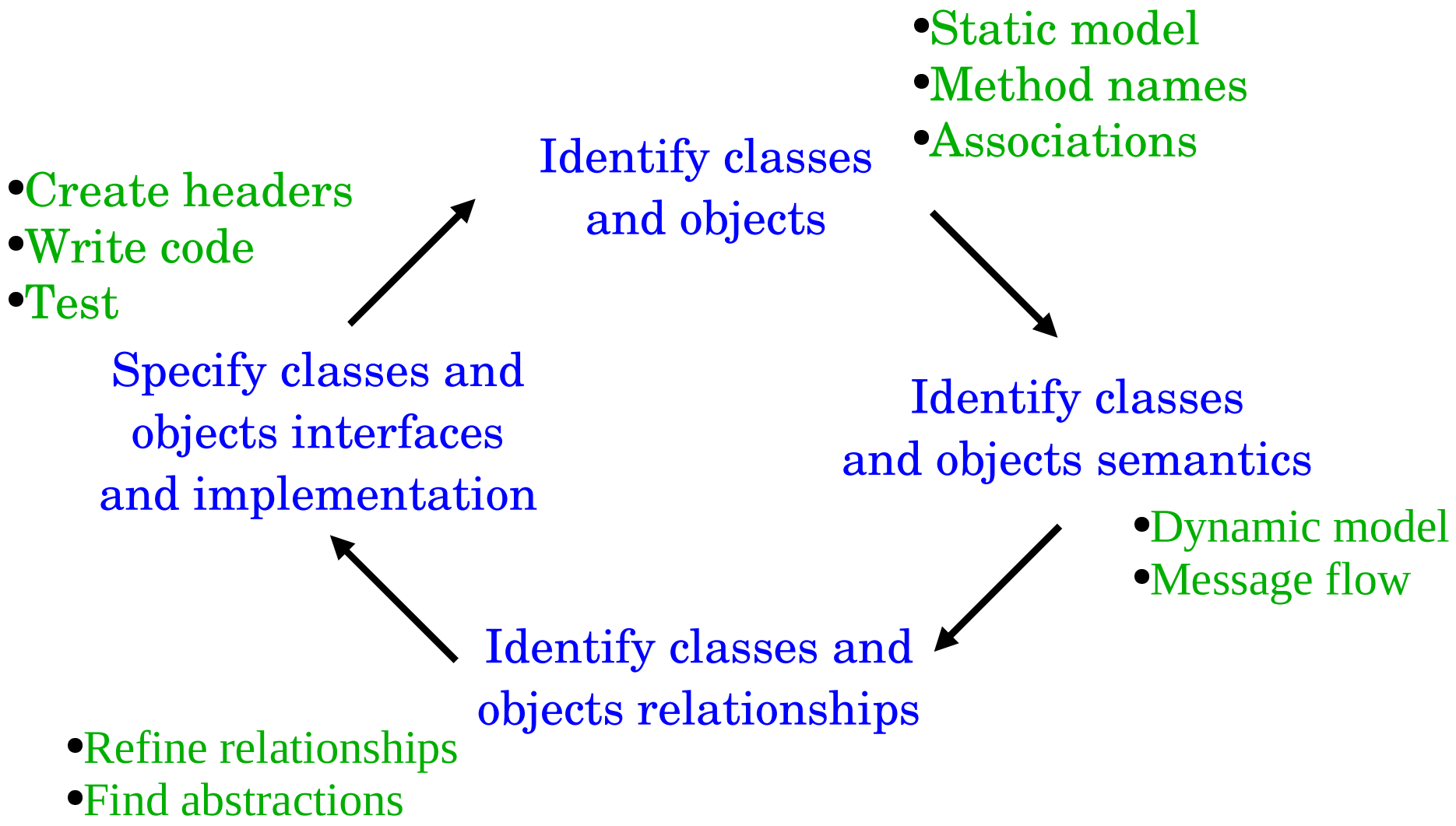
5.4 Milestones

- Milestones (delivery dates) for
  - Requirements documents
  - Design documents
  - Implementation
  - Documentation
- Problematic
  - Hard to predict progress to completion
  - Earlier documentation becomes obsolete

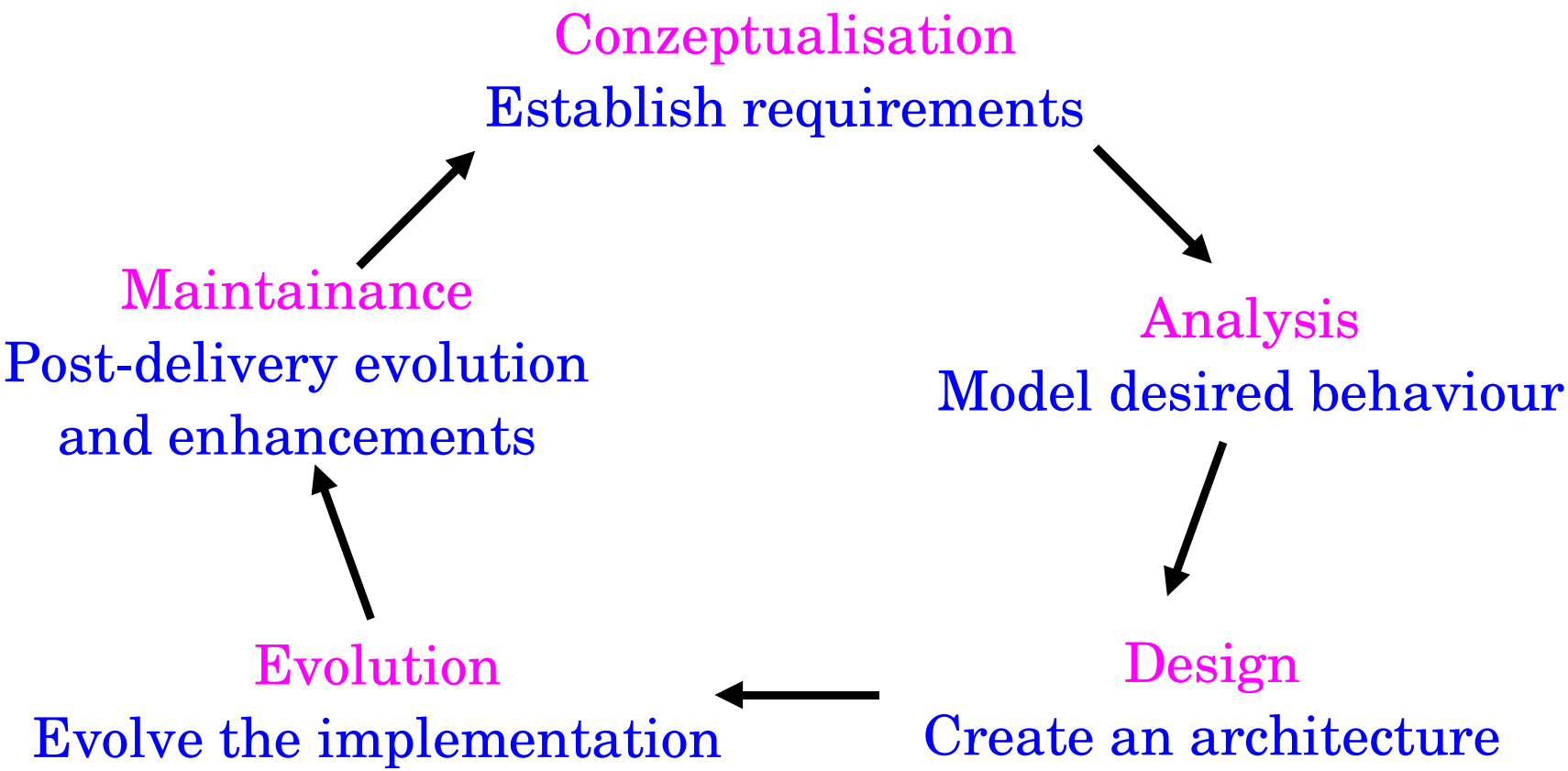
5.4 Iterative Project Management

- 1 Analysis and design to split project into functional components and *slices*
- 2 For each component determine what is needed first (next) → the *slice*
- 3 Develop slices until it works
- Repeat 1 to 3
- Can evaluate effort needed in each cycle
  - Can predict time to completion more reliably
  - Can react when problems appear

5.4 The Booch Micro Cycle



5.4 The Booch Macro Process



5.4 Iterative OOAD Summary

- The OO development process is iterative
  - Anlysis, design, coding, test in small steps
  - More consistency between analysis, design and product
  - Can react early when problems appear
- Feedback from coding to analysis and design
  - spot and correct errors
  - don't be afraid to reconsider analysis and design decisions

5.5 Agile/XP Process

- An overview based on R.C. Martins book
  - What is it? Can we profit from it?
- Observation of process inflation vicious circle
- Need to break this circle
  - Agile or XP
- Emphasis on creative processes, coding and the final product, lightweight on formal steps

5.5 Agile Values

- Individuals & interactions over processes & tools
  - Real people create the code
- Working software over total documentation
  - No document unless immediate and real need
- Customer collaboration over contract negotiation
  - Frequent feedback based on experience
- Responding to change over following a plan
  - Controlled present, fuzzy future

5.5 Agile Principles

- Early and continous delivery of working systems
- Welcome changing requirements
- Stakeholders and developers collaborate daily
  - Users, collaboration managment, developers
- Projects around motivated individuals
  - Support and trust them
- Information in team flows through talking

5.5 Agile Principles

- Progress measured by working software
- Sustainable development
  - Long-distance run, not a sprint
- Attention to technical detail and excellence
  - High quality code
- Simplicity: no unneccessary work
- Self-organising teams
  - Solve problems together

5.5 Agile Practices

- Customer team member
- User/usage stories
- Short cycles
  - 2 weeks iteration, release plan covering 6 iterations
- Acceptance tests provided by customers
  - Need test environment to allow easy tests
- Pair programming of production code
- Test-driven development: test-first programming

5.5 Agile Practices

- Collective code ownership
- Frequent integration
- No overtime, mandatory 40 h week
- Open workspaces
- Planning game with every iteration
- Simple design
  - Most simple solution, complexity only when economical
- Refactoring frequently
  - Adiabatic code changes towards better design

5.5 Agile Summary

- Contains a lot of reasonable suggestions
- Some of them may be applicable to us
- Problems
  - Clearly identified teams?
  - Clearly identified customers/users/ stakeholders?
- Emphasis on code and working product
  - Essential documentation still produced
  - Clean+tidy code, not quick+dirty hacking