Managing
Object Oriented Projects

Dr. Jim Arlow, Senior Consultant

LogOn Technology Transfer

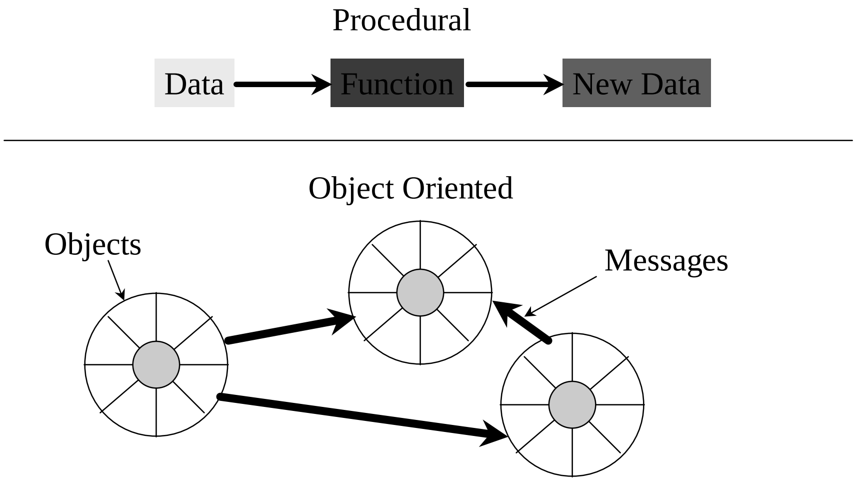
Contents

- Why is an OO project Different?
- What do you need to know?
  - – OO Principles
  - – Lifecycle
  - – Deliverables
  - – Team Structure
  - – Reuse (come to my talk tomorrow!)

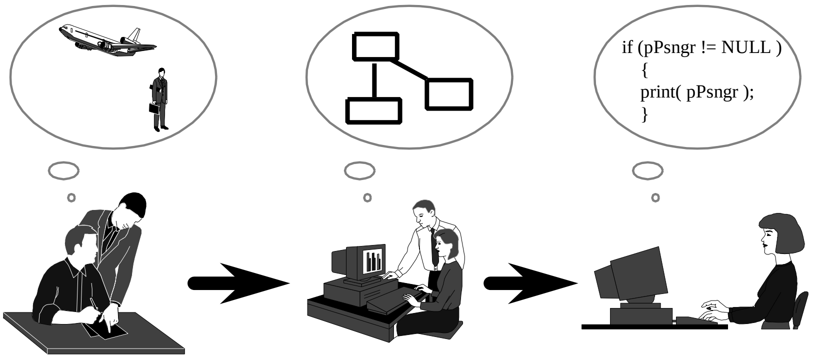
Basic Concepts

Managing OO Projects Part 1

What is OO?



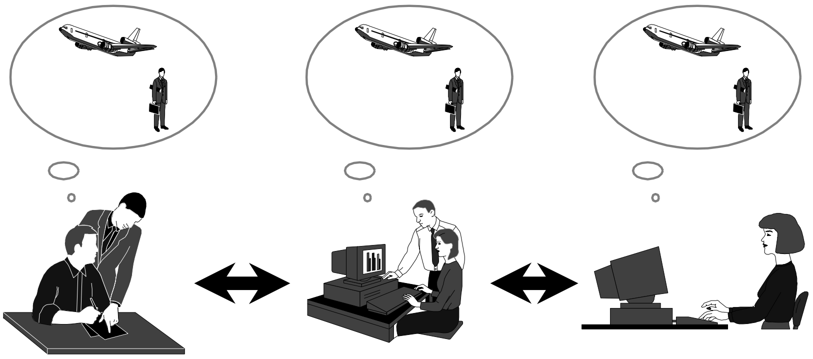
Conventional SE



```
if (pPsngr != NULL )
    {
    print( pPsngr );
    }
```

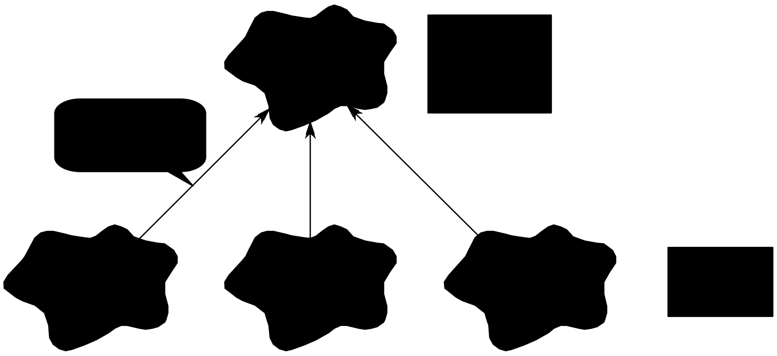
Users Analyst/Designers Programmers

Object Oriented SE



Users Analyst/Designers Programmers

Abstraction, Inheritance and Polymorphism



Advantages of OO

- Conceptual continuity
- Managing Complexity
  - Semantic richness
  - Objects allow partial systems to work
- Quality
- Maintenance - resilience to change
- Time to market
- Reuse

What is an OO Project?

- An OO project is a *sequence* of *unique, complex* and *connected activities* having one *goal* or *purpose* that must be completed by a *specific time*, *within budget* and *according to specification,* that uses *Object Technology* to help reach its goal.
- Clarity of purpose
  - Conditions of Satisfaction

Why is an OO Project Different

- Lifecycle
  - Scheduling
- Skills (manager and programmer)
- Deliverables
  - Classes
- Team Structure
  - Small teams

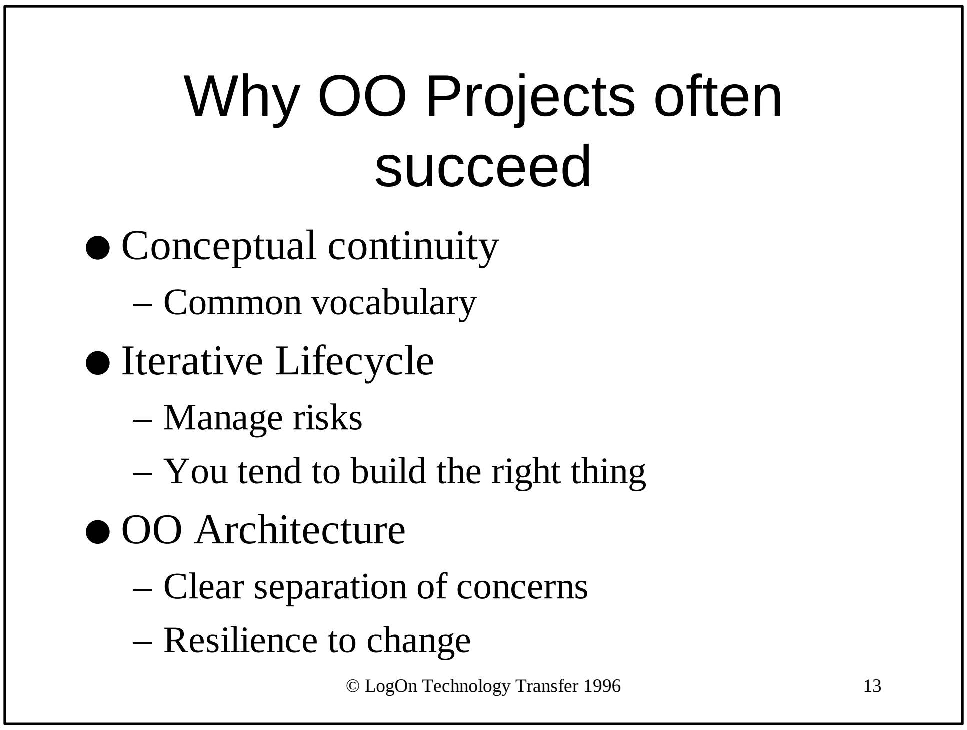
OOPM and Corporate Culture

- “The system structure reflects the culture and organisation of the group that creates it”
- Wide-scale introduction of OO can *change* Corporate Culture
- There is no single strategy for managing an OO project

Reasons why Projects fail

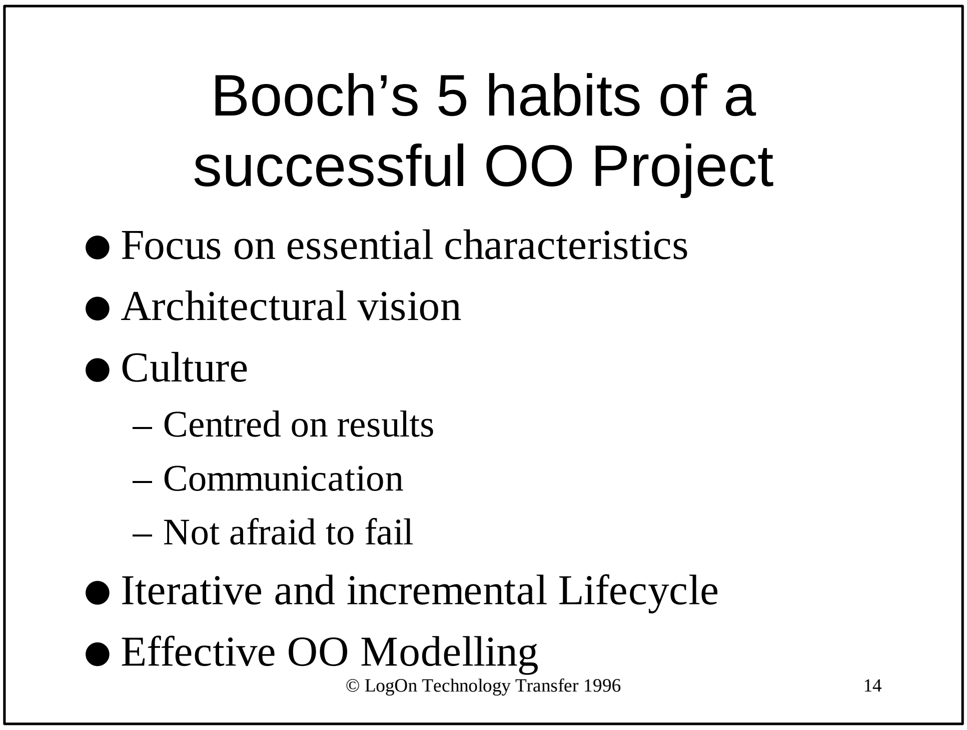
- Don’t manage risks
  - “ Management must actively attack a project’s risks, otherwise they will actively attack you”[1]
  - Build the wrong thing
  - Don’t involve users at all stages
- Technology fails
  - Don’t use system architecture to reduce impact of technology failure

[1] [Gilb 1988]



Why OO Projects often succeed

- Conceptual continuity
  - Common vocabulary
- Iterative Lifecycle
  - Manage risks
  - You tend to build the right thing
- OO Architecture
  - Clear separation of concerns
  - Resilience to change



Booch’s 5 habits of a successful OO Project

- Focus on essential characteristics
- Architectural vision
- Culture
  - Centred on results
  - Communication
  - Not afraid to fail
- Iterative and incremental Lifecycle
- Effective OO Modelling

LogOn's habits for a successful OO Project

- Simplify
- Generalise
- Plan for reuse (if you want it)
- Do the high risk parts first
- Involve end-users as much as possible
- If it's broken, fix it

The bottom line...

- Delivering a system that meets users' present requirements and that can be easily extended to meet users' future requirements on time and within budget
- A Quality System requires a Quality System Architecture

System Architecture

- Physical Architecture
  - Machines
  - Networks
- Logical OO Architecture
  - Classes
  - Class Libraries
  - Frameworks
  - Design patterns

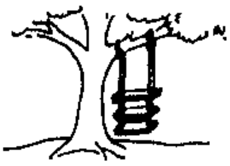
Always focus on “minimal characteristics”

- Project team must have a clear *shared vision* of the desired characteristics
- All decisions must contribute to achieving this vision
- Any decision counter to the vision must be abandoned
- All neutral decisions are luxuries

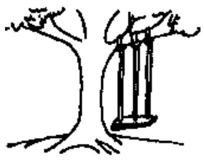
Deliverables

Managing OO Projects Part 2

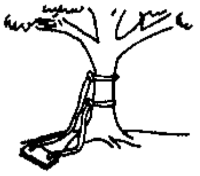
What the Users Wanted...



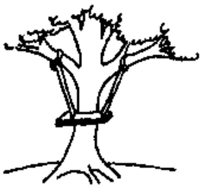
1. As proposed by Project Sponsor



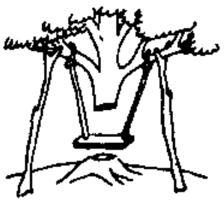
2. As specified in Project Requirements



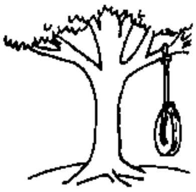
3. As designed by System Architect



4. As produced by Programmers



5. As installed



What the Users wanted

Deliverables of OO Projects

- Managers need to understand exactly what it is that their team is going to deliver!
- Many of the Deliverables of an OO project are significantly different to those of a conventional project
- Ultimately we deliver a working system

Classes

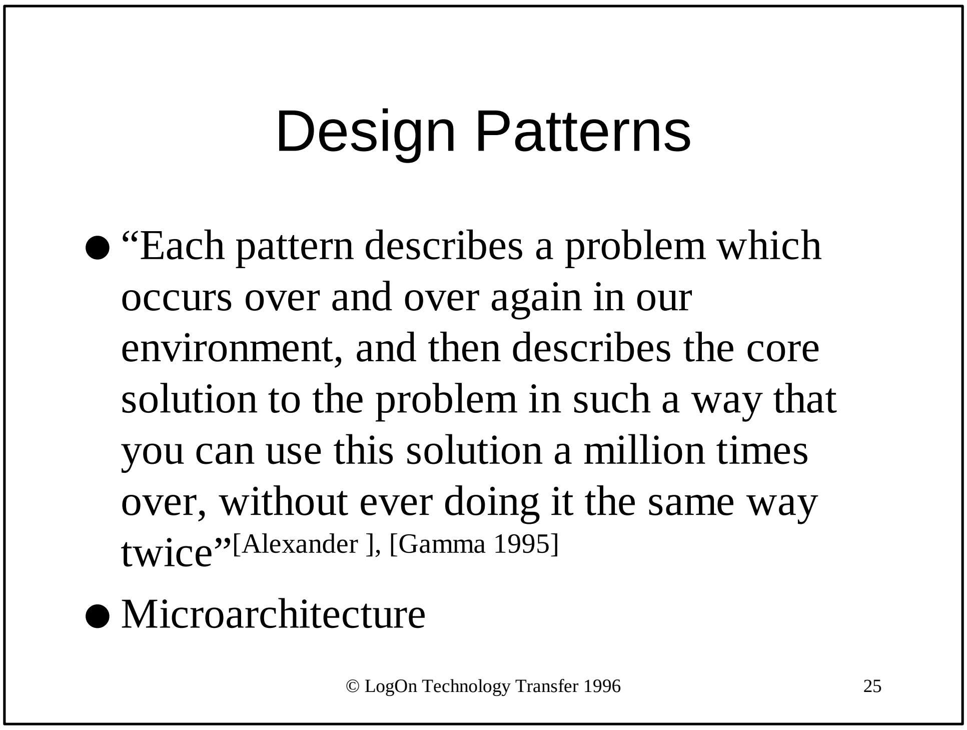
- The Class is the most basic unit of decomposition in an OO project
  - Embodies one abstraction in problem or solution domain
  - Small number of well-defined responsibilities
  - Separates interface from implementation
  - Simple and extendible
  - Generic as possible

Class Rules of Thumb

- Each class should have about 3 to 5 responsibilities ✓
- No class stands alone ✓
- Beware many very small classes ☠
- Beware few but very large classes ☠
- Beware of “functoids” ☠
- Beware of “God” classes ☠
- Avoid deep inheritance trees (C++) ☠

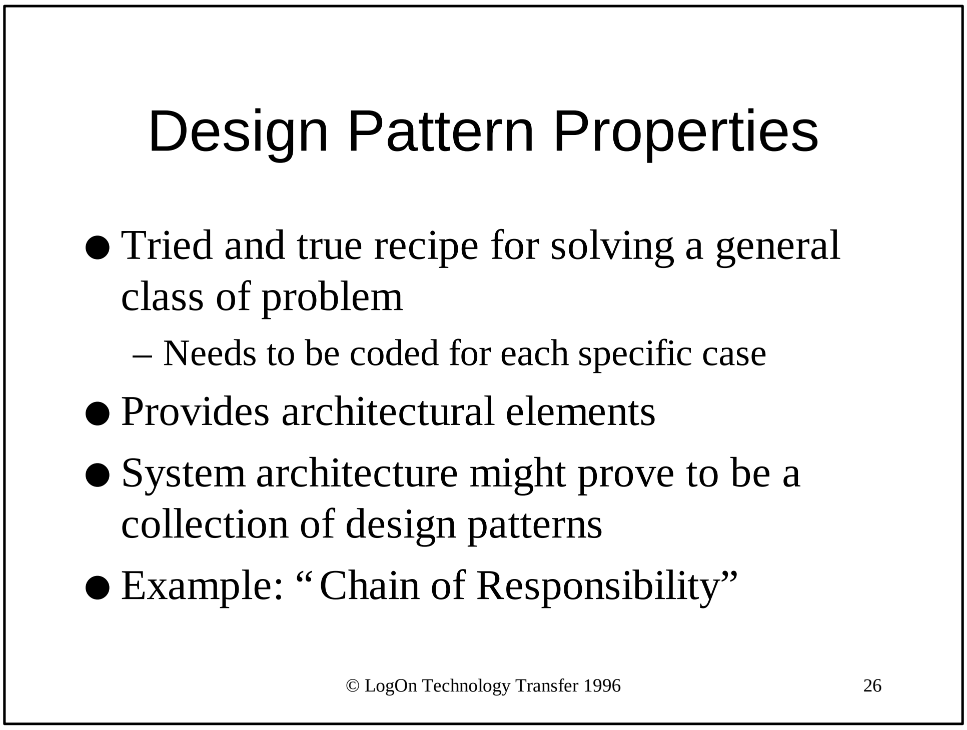
Case Study: Cohesion and Coupling

- A project we reviewed had a very large number of classes, and most of these classes did not embody a crisp abstraction. Furthermore most classes were coupled to most other classes in an ad-hoc manner. Even a small change to this system often necessitated a complete recompilation - a task that took 24 hours to complete.



Design Patterns

- “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to the problem in such a way that you can use this solution a million times over, without ever doing it the same way twice”[Alexander], [Gamma 1995]
- Microarchitecture



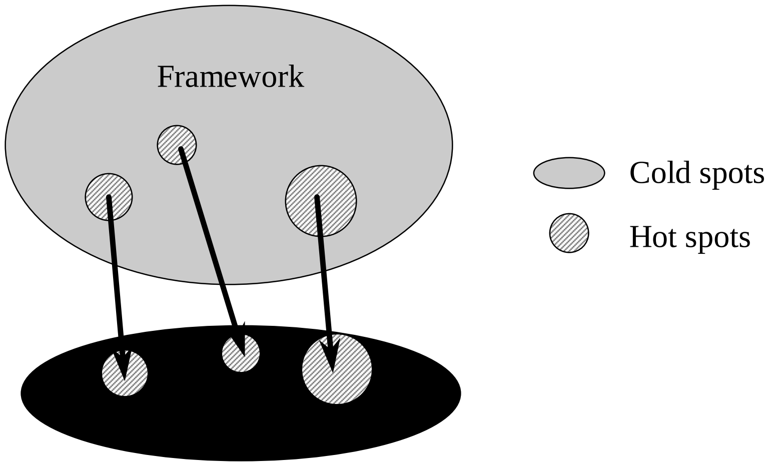
Design Pattern Properties

- Tried and true recipe for solving a general class of problem
  – Needs to be coded for each specific case
- Provides architectural elements
- System architecture might prove to be a collection of design patterns
- Example: “Chain of Responsibility”

Frameworks

- A collection of classes that co-operate together to provide an architectural unit
- A framework
  - – is more than the sum of its parts
  - – embodies an element of system architecture
- e.g.
  - – Model View Controller architecture
  - – Taligent Document Framework

Using a Framework



Framework Properties

- For a powerful framework there should be many cold spots and few hot spots
- The “Hollywood Principle”
  - – Don’t call us, we’ll call you!
- Mechanism for reusing system architecture

Project Culture

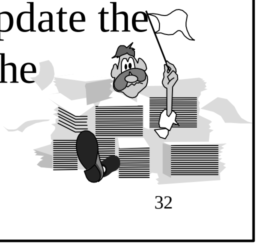
Managing OO Projects Part 3

Drivers for OO projects

- Architecture ☑
  - Maturity and reuse ☑
- Requirements ☑
- Quality
  - Metrics, expense
- Calendar
- Documentation
  - “Paper envy”, [Booch 1995]

Case study: Documentation

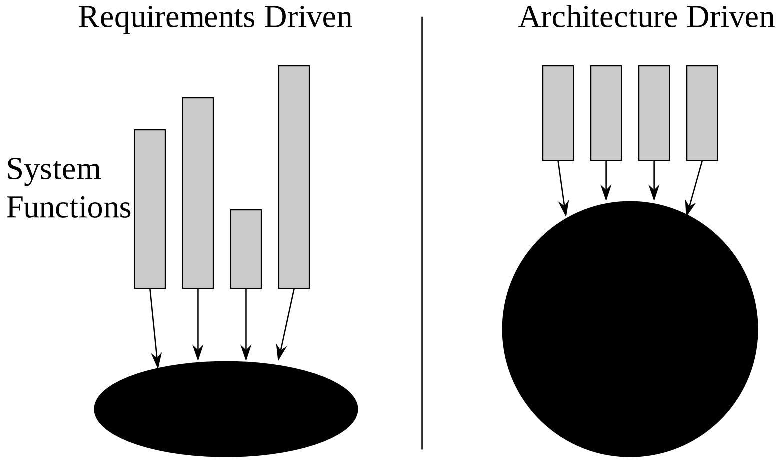
- We once recommended a small change to the code of a system that would have significantly increased its overall quality. The change was estimated at 2 man-hours work. It was rejected as it was estimated that it would take *2 man-days* to update the systems documentation to reflect the change.



Case Study: Requirements

- A project to *display* a chart for planning purposes met all user requirements and was well-received and liked. However, when the users asked for a modification so that they could *edit* the charts, the system architecture could not absorb the change, and the system had to be rewritten.

Requirements vs. Architecture

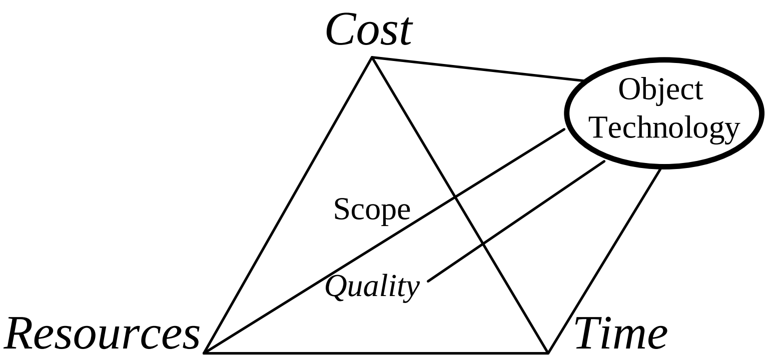


Requirements Driven

System Functions

Architecture Driven

OT Impact on Project



Cost

Object Technology

Scope

Quality

Resources

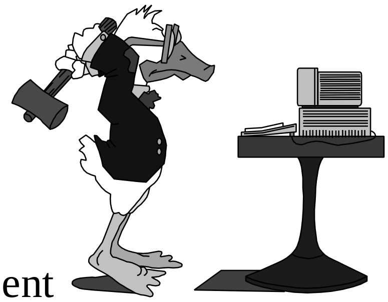
Time

The OO Project Lifecycle

Managing OO Projects Part 4

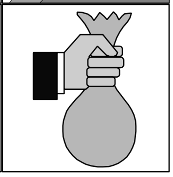
The Six Phases of a Project

- Enthusiasm
- Disillusionment
- Panic
- Search for the Guilty
- Punishment of the Innocent
- Praise and Honours for the non-participants



Four Types of Project

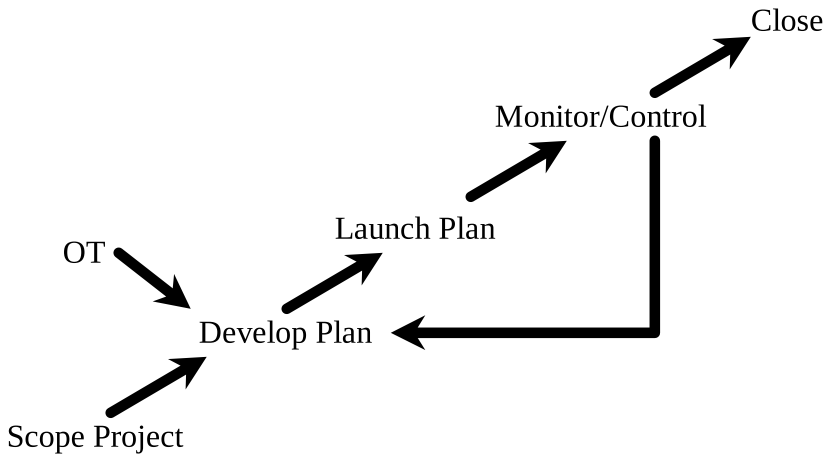
Business Value

		Low	High
Risk	High		
	Low	Who cares?	

OT

can lower risk and increase value

Project Management Lifecycle



Close

Monitor/Control

Launch Plan

OT

Develop Plan

Scope Project

Conventional Lifecycle

Requirements

Analysis

Design

Build

Deliver

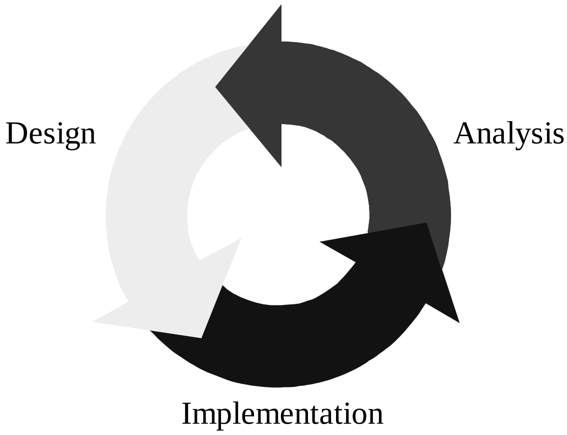
Cascade or Waterfall approach

Cascade Lifecycle Properties

- Linear
  - Assumes initial requirements are correct, complete and do not change
  - Responds poorly to changing business needs
- Big projects mean long time to delivery
  - Can deliver a system no one wants to use
  - Long time before *any* business advantage
- Only works well when we know *exactly* what we want and this *does not change*

Whirlpool or Spiral Lifecycle

Keep iterating around the loop until you’ve had enough!

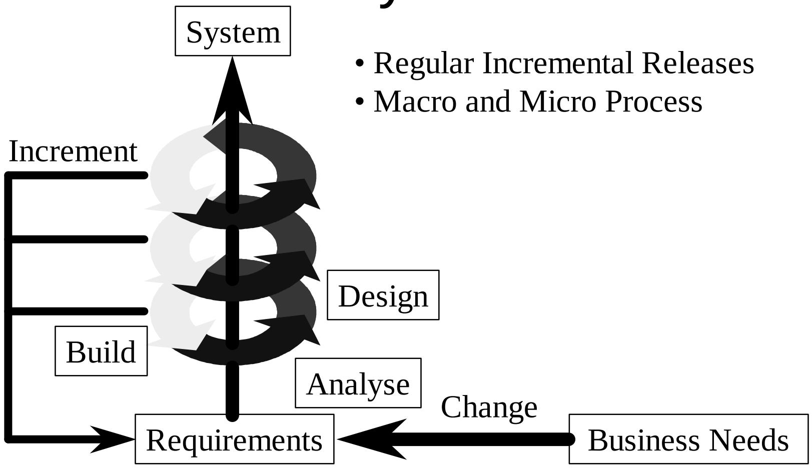


Spiral Lifecycle Properties

- Good for small projects
- Difficult to know when to stop!

The Iterative and Incremental Lifecycle

- Regular Incremental Releases
- Macro and Micro Process



Iterative & Incremental Lifecycle Properties

- Responds well to changing business needs
- Easier to monitor and control
- Greater probability that delivered system will match users’ requirements
- Can more easily manage user expectations
- “Homes in” on desired outcome
  - Outcomes are often moving targets

Impact on Deliverables

- Iterative Lifecycle
  - Incremental delivery
  - Step-wise refinement to final system
  - Mitigate risk at each increment

- Conventional Lifecycle
  - Big-bang approach
  - All or nothing
  - High risk

Iterations and Increments

- Iteration
  - – One pass around the ADB loop
  - – 2 to 5 iterations generally constitute an increment
  - – *Not* a milestone
- Increment
  - – A deliverable which is a useable piece of functionality
  - – Increments are milestones

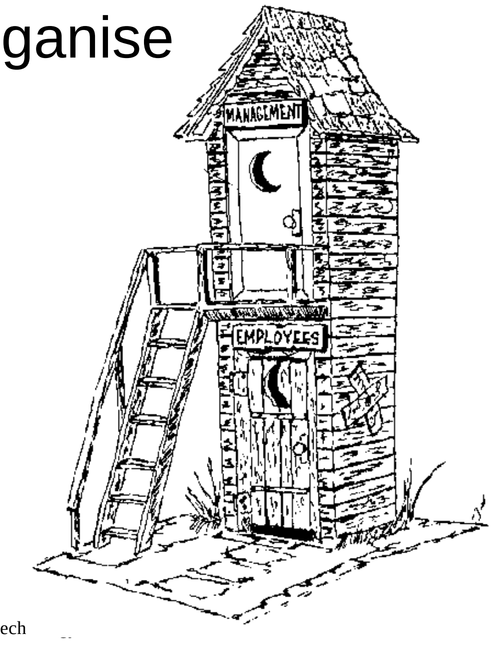
Summary: Axioms for the OO Lifecycle

- Know your purpose
- “Test your theories against reality at the earliest possible opportunity”
- Monitor your goals, and modify your actions to achieve those goals
- Actions affect outcomes

Team Building for OO Projects

Managing OO Projects Part 5

How *not* to organise your team...

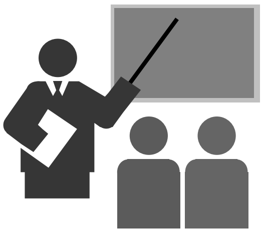


OO Learning Curves

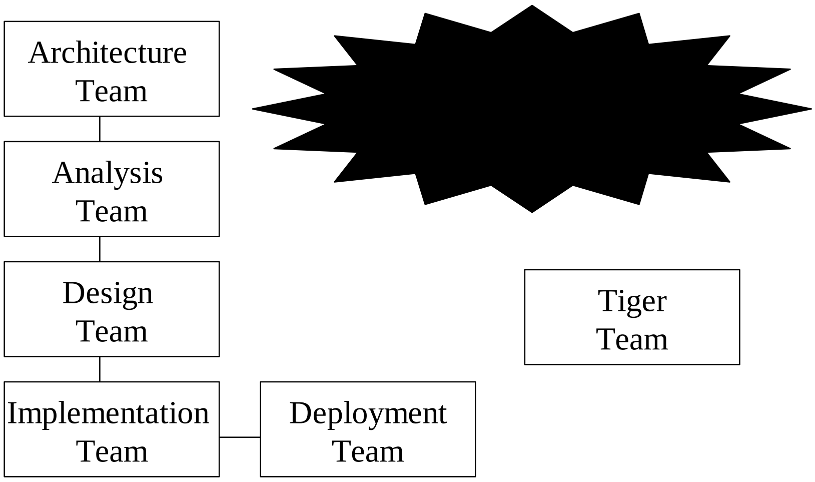
- Programmer
  - – 1 month to learn C++ language syntax
  - – 6 to 9 months to become proficient
- Analysts/Designers
  - – 12 to 18 months to become proficient
  - – OO Design is hard
  - – No substitute for experience

How to Kick-Start an OO Project

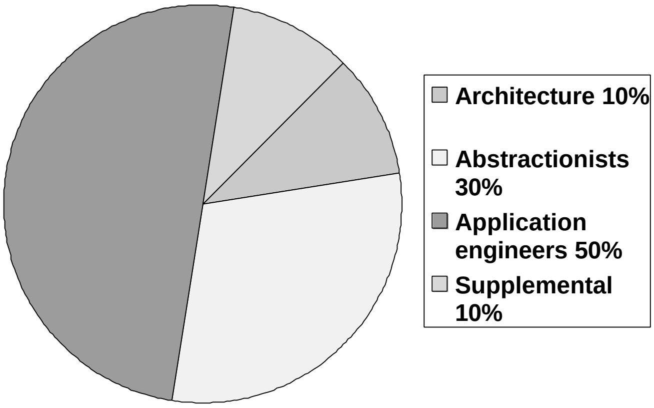
- Mentoring!
- Seed project with experienced people
- External/internal consultants at key stages
  - – Planning
  - – Project start up
  - – Regular reviews
    - both design and code
  - – Post-project review



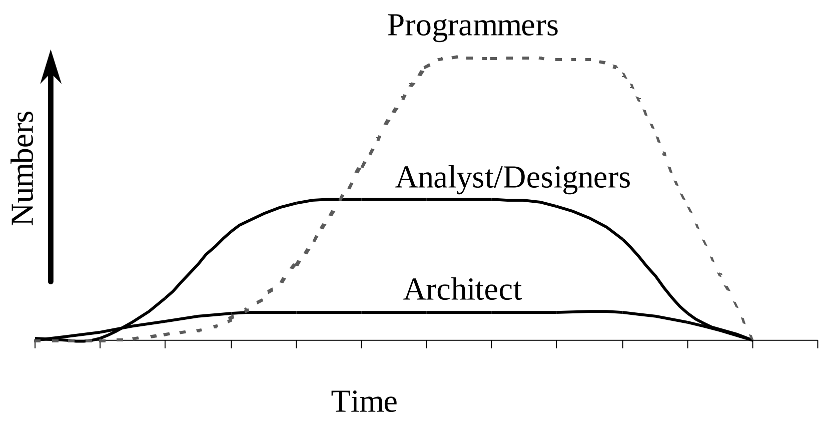
Team Structure: Sub-teams



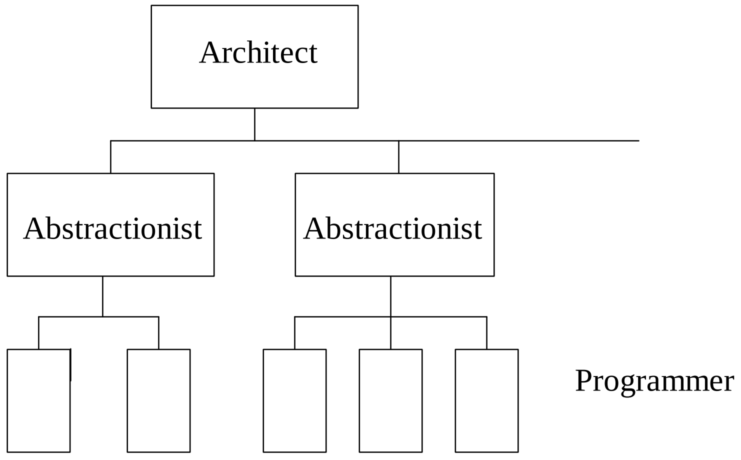
Staffing



Staffing Profiles



Team Structure


Architect

Abstractionist

Abstractionist

Programmer

Roles

- Architect
  - – System architecture and vision
- Abstractionist
  - – Micro-architectures
  - – One Abstractionist per class category
- Programmer
  - – Implementing abstractions

Architect: Responsibilities

- System Architecture
- Assess technical risks
- Define content of successive iterations
  - – Help in planning
- Consultancy
- Marketing
  - – Future product definition

Architect: Skills

- Experience
  - Problem domain and general software engineering
- Vision
- Leadership
- Communication
- Proactive and goal-oriented
- Risk taker

Myth of the replaceable programmer

- Some Project Managers view programmers as the “lowest form of life”. They are just replaceable parts
- This ignores the fact that a good programmer may be up to 10 times more productive than a bad programmer
- Good programmers are very valuable and need to be encouraged and rewarded

OO as an Amplifier

- Object orientation acts like an amplifier - it makes the best programmers much better, and the worse programmers much worse!
- The same is true for Analyst/Designers !

Case Study: Team Building

- A company took a group of non OO programmers and over a period of one month trained them in C++ and an OO methodology. They then launched them straight into a full-blown OO project. Naturally the project failed badly. How did this happen? Management did not understand that OT is different to conventional software development.

Management Strategies for Reuse (Introduction)

Managing OO Projects Part 6

Reuse Myths...

- We are doing OO therefore we get reuse
- Reuse is free
- We don’t need to organise for reuse
- A Library Tool will give us reuse
- All the programmers put reusable code in a shared LAN directory (BPS strategy)

Reuse is a cultural issue

- Your project/organisation must decide whether it is serious about reuse or not
- If you want reuse you must make it a deliverable
  - Schedule for reuse
  - Organise the team for reuse
  - Create new roles

Types of reuse

- Reuse of Architecture
- Reuse of Analysis
- Reuse of Patterns
- Reuse of Designs
- Reuse of Documentation
- Reuse of Code
  - N.B. Cut and Paste is NOT reuse

Managing reuse

- Building *with* reuse
  - – Reuse existing components
    - Class libraries
  - – Increases productivity
- Building *for* reuse
  - – Creating libraries of new reusable components
  - – Need new roles and responsibilities
  - – Initial decrease in productivity

Building for reuse

- Creating reusable components is expensive
- Productisation
  - – Quality
    - Testing
  - – Completeness
  - – Documentation
    - Example programs
  - – Support
    - Service level agreement

Reusable components cost more

- Need to put in extra effort to make the component generic
- Don’t know how or where the component will be used - needs to be more complete
- As component may impact important projects it needs thorough testing
- Must be well documented in order to be reusable

Creating reusable components: Staffing

- Requires best and most creative people
- They must have a clear mandate to create the components
- Should be well rewarded as they can have a major impact on the project
- Staff for the maintenance phase
- Staff for support

Summary

- Managing an OO project is significantly different to managing a conventional project
- Understanding OT and understanding the different management strategies required for the OO project will lead to success

Bibliography

- [Booch 1996] Booch, G. *Object Solutions Managing the Object-Oriented Project*, Addison-Wesley 1996, ISBN 0-8053-0594-7
- [Meyer 1995] Meyer, B. Object Success, Prentice Hall 1995, ISBN 0-13-192833-3
- [DeMarco 1982] DeMarco, T. *Controlling Software Projects: Management, Measurement and Estimation*, Yourdon Press, Edgewood Cliffs, NJ 1982
- [Booch 1994] Booch, G. *Object Oriented Analysis and Design with Applications* (second edition), Benjamin/Cummings 1994, ISBN 0-8053-5340-2
- [Wysocki 1995] Wysocki, R. *Effective Project Management*, Wiley 1995, ISBN 0-471-11521-5
- [Lorenz 1994] Lorenz, L. Kidd, J. *Object Oriented Software Metrics*, Prentice Hall 1994, ISBN 0-13-179292-X
- [Gamma 1995] Gamma, E. *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley 1994, ISBN 0-201-63361-2
- [Brooks 1995] Brooks, P. *The Mythical man-Month*, Addison-Wesley 1995, ISBN 0-201-83595-9
- [Alexander 1977] Alexander, C. *A Pattern Language*, Oxford University Press, 1977
- [Parnas 1986] Parnas, D. *A Rational Design Process: How and Why to Fake It*, IEEE Transactions on Software Engineering vol. SE-12(2) 1986
- [Gilb 1988] Gilb, T. *Principles of Software Engineering Management*, Addison-Wesley, 1988