Object Discovery

A process for developing medium-sized object-oriented applications

Part One

History and Principles behind the Discovery Method

Background to Discovery

- **Third-Generation**
    - **1st - “naive” generation, circa 1990**
    - **2nd - “eclectic” generation, circa 1994**
    - **3rd - “selective” generation, circa 1998**
- **Influences**
    - **OBA/Graham/Jacobson/Gilb analysis**
    - **RDD/OOSE object modelling**
    - **JSP/Harel control, ERM data modelling**
    - **RDD/Design Patterns system architecture**
    - **Fusion/Z detailed design and specification**

Background to Discovery

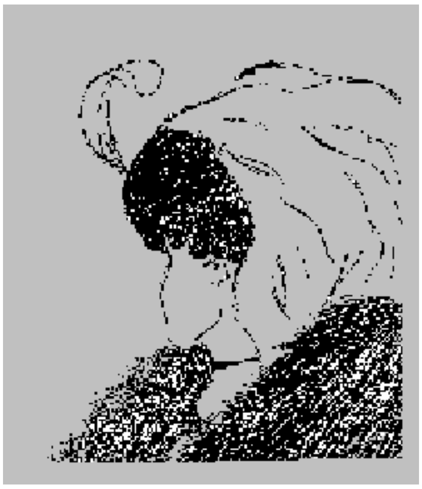
- **Case Studies**
    - **Reengineer 10-year old CAD**
    - **Glass gob manufacture**
    - **Personal and business loans**
    - **In-flight shopping and entertainment**
    - **Academic registration and tracking**
    - **Course options organiser**
- **Affiliation**
    - **OPEN Consortium (30+)**
    - **OPEN Working Groups (~15)**
    - **OPEN Process and Metamodel**

Principles of Discovery

- **Transformational**
  - **...versus elaborational**
  - **current into required (cf SSADM)**
  - **analysis into design (cf SSADM)**
  - **seamlessness versus traceability**
- **Cognitive Focus**
  - **power of abstractions (cf Gestalt)**
  - **plasticity of early models**
  - **selective use of techniques**
  - **“discovery procedures”** ★

★ indicates some reinforcement technique

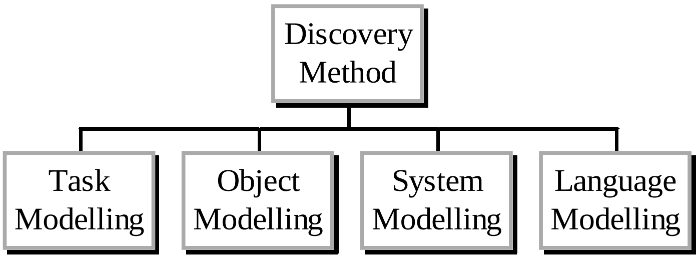
Power of Abstraction



Principles of Discovery

- **Technical Process**
  - **respect focus of techniques**
  - **sequence inputs and outputs**
  - **cross-check overlapping models**
  - **provide systematic guidelines**
- **Communication**
  - **continuous client involvement**
  - **uncluttered visual models**
  - **presentation and feedback**
  - **setting development priorities**

Phases of Discovery


Discovery Method

Task Modelling

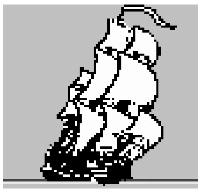
Object Modelling

System Modelling

Language Modelling

➡ 9 ➡ 42 ➡ 70 ➡ 95

Part Two

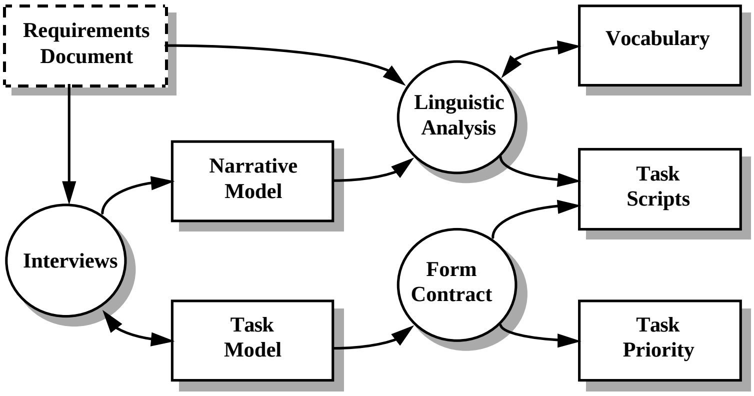


Discovery Task Modelling Phase: Requirements Engineering

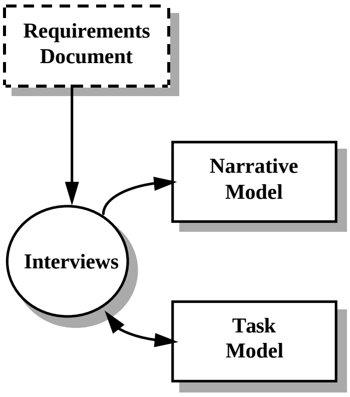
Task Modelling Activities

- ❍ **interviewing techniques** ➜ 12
- ❍ **system task identification** ➜ 20
- ❍ **narrative modelling** ➜ 24
- ❍ **construction of vocabularies** ➜ 28
- ❍ **scripting in restricted grammar** ➜ 31
- ❍ **task priority matrix** ➜ 37

Discovery: Task Modelling



Interviews



Goals of Interviewing

- **Establish Business Model**
  - **whole business context**
  - **human and machine interactions**
  - **natural business constraints**
- **Capture Client Concerns**
  - **important make-or-break**
  - **social, political factors**
- **Establish Feasible System**
  - **automation boundary**
  - **costs and benefits**

Interviewing Bias

- **The Client**
  - **total understanding of problem**
    - **implicit, compiled expertise**
  - **limited explicit formulation:**
    - **single or narrow goals**
    - **existing procedures, interfaces**
    - **“requirements” unreliable**
- **The Developer**
  - **limited understanding of problem**
  - **early explicit incomplete formulation:**
    - **competence in systems**
    - **aware of rationalisations**

Non-directive Interviewing

- **Modelling Assumptions**
  - **Don’t impose your own object-model**
  - **Let client express their own business model**
- **Question Presuppositions**
  - **Don’t lead with “what X do you do/have?”**
  - **Use open-ended questions “tell me about...”**
  - **Don’t use multiple-choice, multi-part**
- **Active Listening**
  - **clarification - “let me see if I got this right...”**
  - **summarisation - “first, you do X, then ...”**

Directed Interviewing

- **Stakeholders** ★
  - **viewpoint analysis by stakeholder**
  - **bluesky, win-win scenarios**
- **Task-centred** ★
  - **natural focus of client**
  - **mission-critical task(s)**
  - **task dependency:**
    - **independent, concurrent (thread)**
    - **precursor, consequent (logical)**
    - **upstream, downstream (time-order)**

Directed Interviewing

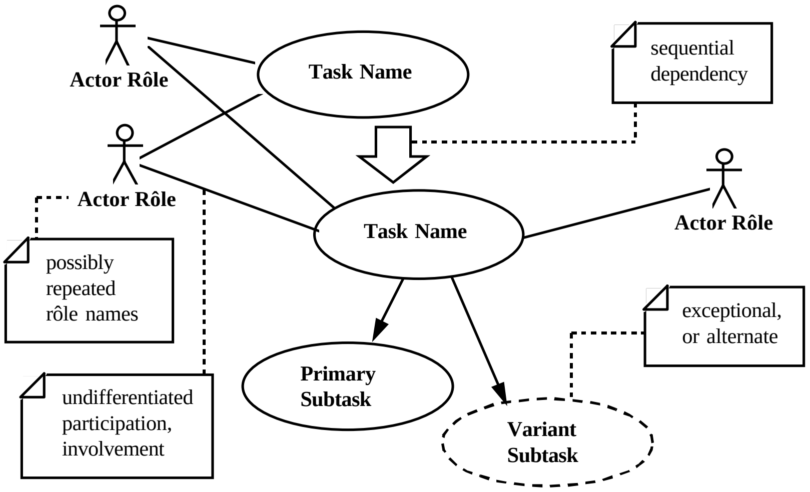
- “Wh”-questions
  - who - stakeholders, actor-rôles
  - what - primary tasks, by rôle
  - how - business process, subtasks
  - why - logical justification, dependent tasks
  - when - time constraints, dependent tasks
  - where - impl. constraints, boundaries

  - subtasks reveal new actor-rôles
  - continue until no new tasks revealed

Build the Task Model

- Rationale
  - high-level model of business
  - communication - visual impact
  - client feedback (corrections)
  - developer feedback (rationalisations)
- Technique
  - sketch tasks during interviews → 19
  - structure task model → 20
  - present to client, revise → 23
  - suggest optimisations, revise

Task Sketch Syntax

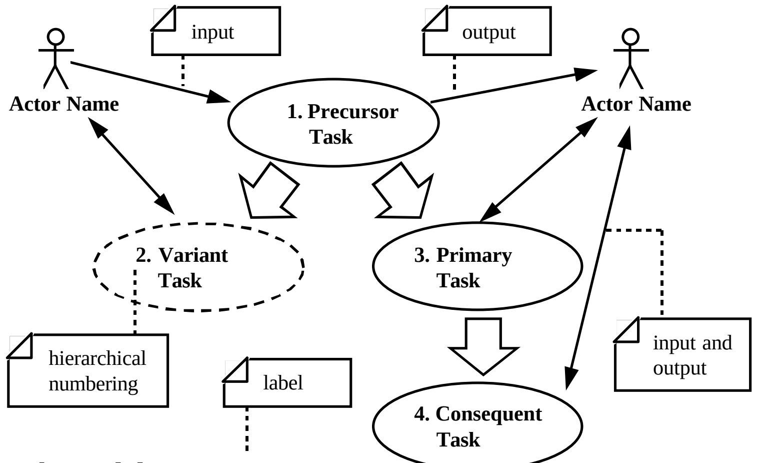


Structure Task Model

- **Rationale**
  - **captured n tasks in a flat sketch**
  - **variable granularity, crossed lines**
  - **compose hierarchy by rules**
- **Technique**
  - **cluster tasks by focus, goal, purpose**
  - **preserve contiguous boundaries**
  - **aim for 2-5 tasks per level**
  - **prefer to avoid crossed lines**
  - **isolate actor participation types**

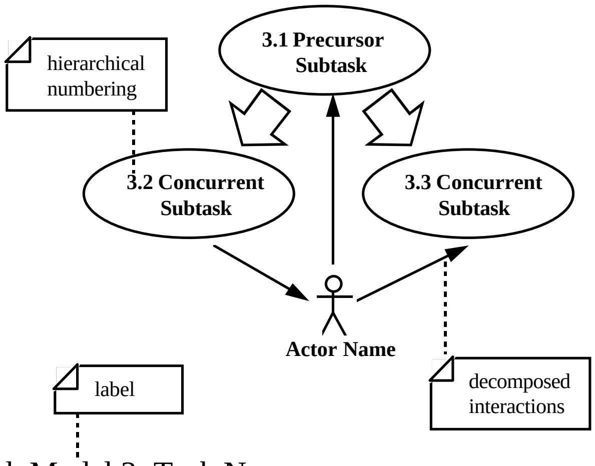
by actor, direction

Task Model Syntax



Task Model 0: System Name

Task Model Syntax



Task Model 3: Task Name

Task Model Checks

- **Client revision**
  - **check structure is meaningful**
  - **check participants all present**
- **Developer revision**
  - **isolate actor interfaces**
  - **propose task migration**
- **Cross-checks**
  - **ensure contiguous clusters**
  - **propagate participation types by level** → **21-22**
  - **constraints on model size, crossed lines**

Build the Narrative Model

- **Rationale**
  - **storytelling cognitively relevant**
  - **natural business process paradigm**
  - **client’s natural language**
- **Technique**
  - **complete for each task, subtask**
  - **identify actor rôles, materials** → **25**
  - **identify prerequisites, postrequisites**
  - **describe tasks, including:**
    - **subtask dispatch points** → **26**
    - **alternate task branch points**
    - **exception task break points**

Narrative Model Syntax

Narrative Model 0.0: Title of Task	**Author**	**Date**	**Revision**
Purpose:	summary of task goals		
Actor Rôles:	participants at this level (may elide)		
Materials:	documents and artefacts used (may elide)		
Prerequisites:	material constraints/availability, actor states/preparedness, logical dependency, sequential dependency.		
Description:	Natural-language description of task, which may include dispatch points for sequential or concurrent subtasks, branch points for alternate tasks and break points for exceptional tasks. Any rules, conditions, events recorded in the most natural way.		
Exceptions:	Natural-language description of abnormal termination cases.		
Postrequisites:	revised material constraints and actor states, validation statement of completeness, e.g. with respect to dependent tasks.		

Narrative Model Syntax

- **Presentation options**
  - **embed dispatch points in a single paragraph**
  - **organise by subtask, summarise under subheadings**
  - **organise by business rule, summarise under subheadings**
  - **include exceptions at end of description**
  - **dispatch to exception tasks at break points**

Task 1.1: Name
Summary of subtask

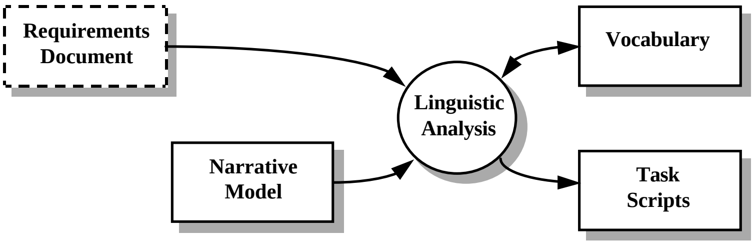
Task 1.2: Name
Summary of subtask

Summary of first subtask **[Task 1.1]**. Summary of second subtask **[Task 1.2]**.

Narrative Model Checks

- ❍ **Client Revision**
  - ❍ **check completeness of detail**
  - ❍ **check business logic makes sense**
- ❍ **Cross-Checks**
  - ❍ **narrative produced for each task**
  - ❍ **uses only those actors, materials listed**
  - ❍ **prerequisites enabled by precursors**
  - ❍ **postrequisites enable successors**

Linguistic Analysis



Build the Vocabulary

- **Rationale**
  - **client vocabulary is rich, context-dependent**
  - **possible repeated, ambiguous concepts**
  - **“boil down” the domain vocabulary**
- **Technique**
  - **identify primary, secondary NPs, Vs**
  - **identify AdjPs, AdvPs, PPs**
  - **attach AdjPs, secondary NPs to NPs** ➡ **30**
  - **attach AdvPs, (phrasal) Preps to Vs**
  - **cluster concepts, reduce** ➡ **30**
  - **validate vocabulary with client**

Vocabulary Clustering

Item	**Part**	**Master**	**Description**
degree	N		study programme
module	N		degree course unit
student	N		registered for degree
level	N	student	year of study
option	N	degree	optional course unit
core	Adj	module	obligatory for degree
approved	Adj	module	optional for degree
free	Adj	module	arbitrary course unit
select	V		prioritise approved module choices
allocate	V		assign students to modules

Build the Scripts

- **Rationale**
  - **re-express tasks in core vocabulary**
  - **abstract from physical context**
  - **abstract from user interface**
- **Technique**
  - **use only the core vocabulary**
  - **express in restricted grammar**
  - **complete 1+ scripts for each narrative**
  - **formalise pre-, postconditions**
  - **adjust vocabulary**
  - **adjust task decomposition**

➡ 32
➡ 35

Restricted Grammar

- **Graham's SVDPI**

  Subject Verb DirectObject
  [Preposition IndirectObject]
- **Simons' TCA**

  Test Condition Action

```
Test ::= IF Condition THEN Action [ELSE Action]
          | WHILE Condition DO Action
          | WHEN Condition DO Action
          | FOR Subject IN IndirectObject DO Action;
Condition ::= Subject  Verb  [Complement];
Action ::= SVDPI;
```

Restricted Grammar

- **Simplify Narrative**
  - **reduce full NL to simple active sentences**
  - **may yield object-stimulus-object patterns**
- **Adjust Vocabulary**
  - **productive in forcing generalisations**
- **Adjust Tasks**
  - **productive in simplifying control structures**
  - **express logic for single item transactions**
  - **allow iteration row/column transformation**

Task Scripts

Script 2.1: Loan Application	**Author**	**Date**	**Version**
PRE true;			

the **customer** *completes* a **loan application**;
IF the **manager** *approves* the **loan application**
THEN
    the **clerk** *sends* an **acceptance letter** to the **customer**
ELSE
    the **clerk** *sends* a **rejection letter** to the **customer;**			

POST
    IF **loan application** *is approved*
    THEN **customer** *has* **acceptance letter**
    ELSE **customer** *has* **rejection letter**;			

Task Scripts

Script 2.2: Loan Financing	**Author**	**Date**	**Version**

PRE **loan application** *is approved*;

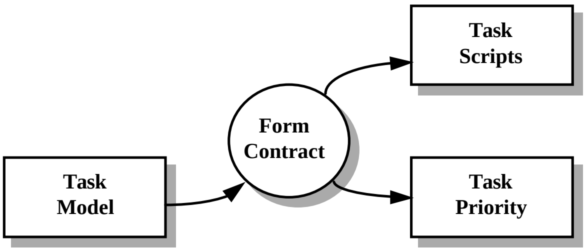
the **manager** *forwards* the **loan application** to the **banker;**
the **clerk** *opens* a new **loan account** for the **customer**;
WHEN
  the **manager** *receives* a **finance note** from the **banker**
DO
  the **clerk** *sends* a **repayment schedule** to the **customer;**

POST
  the **banker** *has* the **loan application;**
  the **loan account** *is* open;
  the **customer** *has* a **repayment schedule;**

Script Checks

- **Client revision**
  - **vocabulary is adequately subtle**
  - **stylised scripts retain business process**
- **Cross-checks**
  - **every narrative has one or more scripts**
  - **each NP, V, ... defined in vocabulary**
  - **narrative rôles, materials preserved**
  - **narrative business logic preserved**
    - **postconditions entail postrequisites**
  - **inception and completion formally defined**
    - **concrete, testable pre-, postconditions**

Form Contract



Form Contract

- **Rationale**
  - **select system components**
  - **agree system specification**
  - **agree delivery schedule**
  - **agree pricing strategy**
- **Technique**
  - **construct priority matrix**
  - **add client, developer factors**
  - **prioritise tasks by category**
  - **agree specification, schedules**
  - **sign off contract**

➡ 39
➡ 41

Task Priority Matrix

Task	OrdId	CliPri	DevPri	EasDev	Priority
1 Registration					18
1.1 UGrad Pre-reg.	5	5	5	3	18
1.2 UGrad Registration	5	5	5	4	19
1.3 PGrad Registration	4	5	4	4	17
2 Academic Progress					16.66
2.1 UGrad Re-reg.	5	5	5	4	19
2.2 Course Changes					15.50
2.2.1 Module Add/Drop	4	5	4	3	16
2.2.2 Degree Change	4	4	3	4	15
3 Alumni Tracking	3	3	2	4	12

Priority Factors

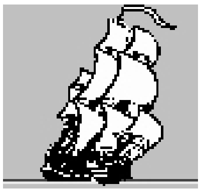
- **Order of Identification**
  - **high score for mission-critical task**
  - **middle score for client-identified task**
  - **low score for developer-suggested task**
- **Client and Developer Priority**
  - **client’s sense of task urgency**
  - **developer’s sense of task dependency**
  - **invert cost, difficulty: ease of development**
- **Score to Chosen Depth**
  - **add equal-scaled factors for leaf nodes**
  - **average all *leaf* scores for other nodes**

Rank Tasks

- **incremental delivery**
- **guaranteed and run-on pricing**
- **future extensions**

1.2 UGrad Registration	19	Essential
2.1 UGrad Re-reg.	19	
1.1 UGrad Pre-reg.	18	Necessary
1.3 PGrad Registration	17	
2.2.1 Module Add/Drop	16	
2.2.2 Degree Change	15	Desirable
3 Alumni Tracking	12	Optional

Part Three

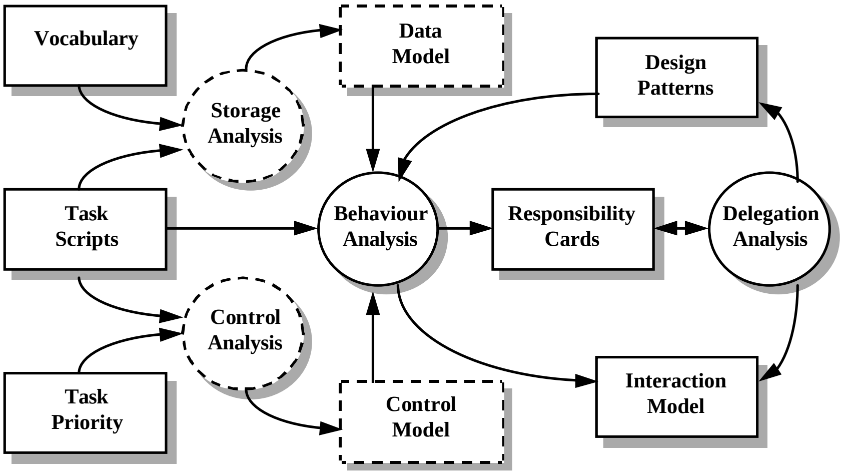


Discovery Object Modelling Phase: a Responsibility-Driven Approach

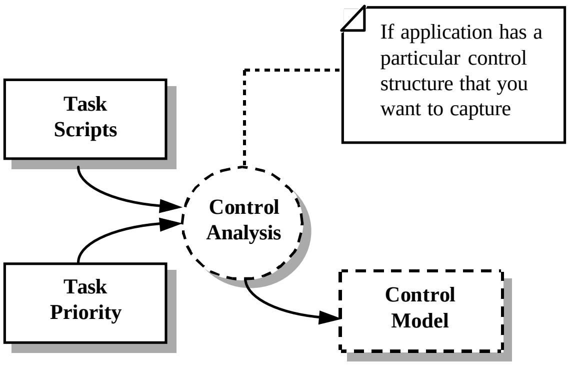
Object Modelling Activities

- ❍ **object identification by behaviour** ➡ 54
- ❍ **responsibility/collaboration analysis** ➡ 59
- ❍ **delegation using Design Patterns** ➡ 56
- ❍ **interaction diagram (object messaging)** ➡ 66
- ❍ **data modelling (back-end)** ➡ 49
- ❍ **control structures (front-end)** ➡ 45

Discovery: Object Modelling



Control Analysis

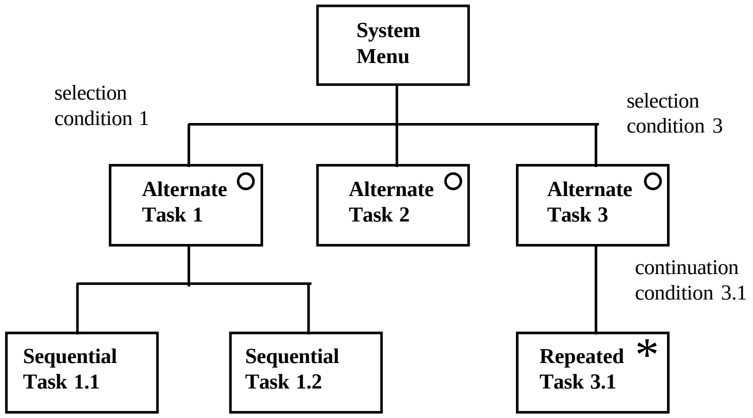


If application has a particular control structure that you want to capture

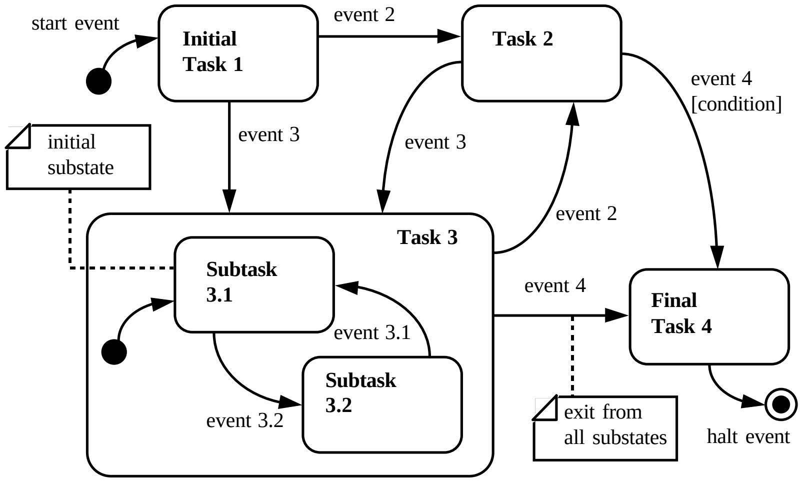
Control Model ★

- **Hierarchical**
  - **free selection of tasks**
  - **impose menu structure**
  - **use e.g. JSP-style Structure Chart** ➡ 47
- **Event-Driven**
  - **tasks enabled, disabled**
  - **impose abstract state model**
  - **use e.g. Harel-style, UML State Diagram** ➡ 48
- **Asynchronous**
  - **shared access to tasks**
  - **use e.g. Petri Net, UML Activity Diagram**

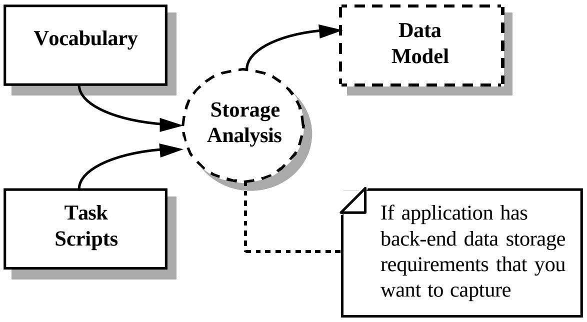
Hierarchical Control Model



Event-Driven Control Model



Storage Analysis

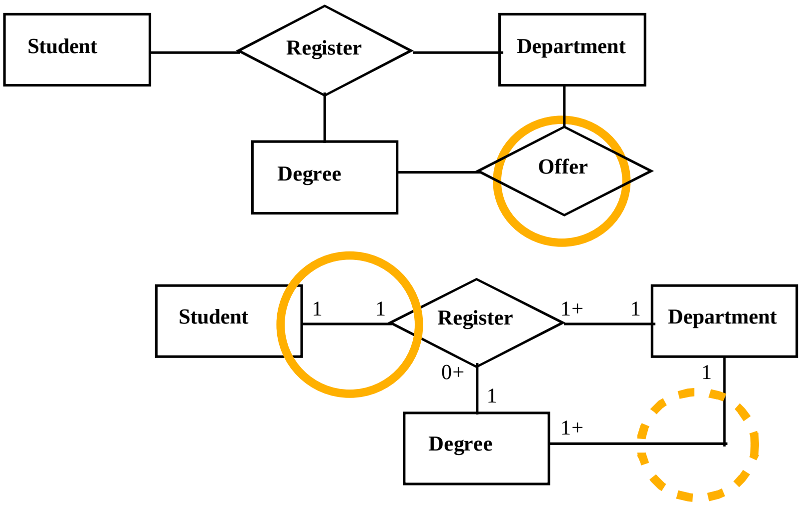


If application has back-end data storage requirements that you want to capture

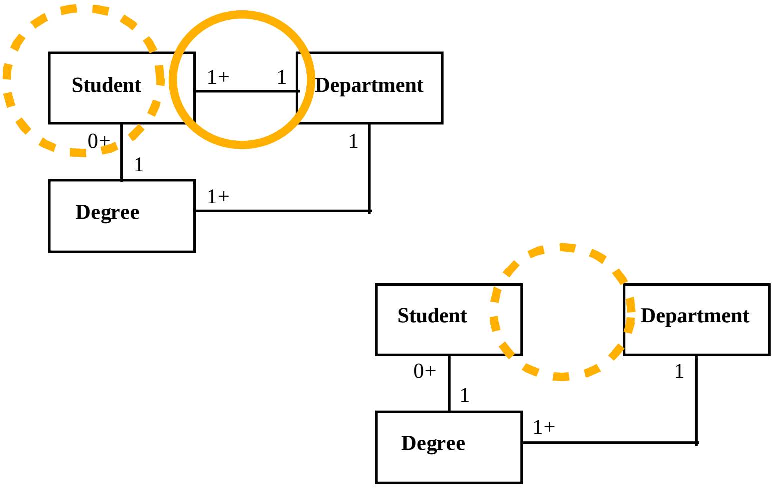
Data Model

- **Rationale**
  - **optimise data storage model**
  - **interface with relational DBMS**
  - **no overlaid “object model” semantics**
- **Technique** ★
  - **entities from NPs + storage Vs**
  - **build ERM using associations** ➡ 51
  - **eliminate redundant paths** ➡ 52
  - **encapsulate M:N, ternary, and higher** ➡ 51
  - **link dependent to master entities** ➡ 53

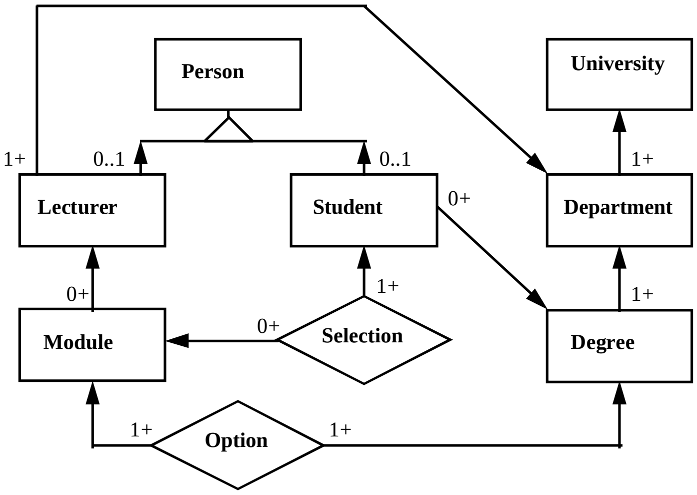
Entity-Relationship Transformations



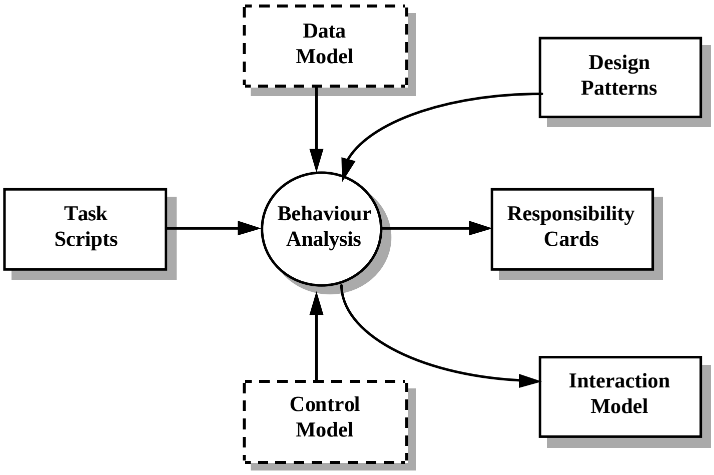
Entity-Relationship Transformations



Data Model



Behaviour Analysis



Behaviour Analysis

- **Rationale**
  - **objects are agents with a purpose**
  - **classify by external behaviour**
  - **distribute system functionality**
- **Technique** ★
  - **use catalogue of rôles, design patterns** ➡ 56
  - **identify responsible agents from scripts** ➡ 57
  - **include any data-managers, controllers**
  - **write responsibility cards** ➡ 59
  - **build interaction model** ➡ 66

Pattern Catalogues

- **Categories**
  - **model, support, viewer, controller, stream**
  - **business objects, system objects, data objects, event handlers, command interfaces**
- **Design Patterns**
  - **STL, Booch:** ***Bridge***
  - **ERM:** ***Observer, Mediator, Bridge***
  - **RDD:** ***Mediator, Template Method, Command, Chain of Responsibility***
  - **EDD:** ***Command, Composite, Chain of Responsibility***

Objects from Scripts

- **Responsibility**
  - **who is responsible for the VP activity?**
  - **could be S, D, I or other**
- **Examples**
  - **subject responsibility:**

    the **manager** *forwards* the **loan application** to the **banker;**
    the **clerk** *opens* a new **loan account** for the **customer**;

  - **object/other responsibility:**

    the **manager** *asks* the **banker** for the **funds**;
    the **student** *supplies* his/her **student details** for the **registration form**;
    the **borrower** *borrows* the **book;**

Verb Phrase Indicators

- **act upon/request**
  - **direct object responsible for handling;**
  - **subject responsible for initiating**
- **transmit/send**
  - **subject responsible for sending;**
  - **indirect object responsible for receiving/handling**
- **provide/supply**
  - **neither actor-subject nor form-object**
  - **usually, some other clerk or manager**

Responsibility Cards

- **Rationale**
  - **generalisation of CRC cards**
  - **identify objects by responsible rôles**
  - **keep active agent perspective**
- **Technique** ☆
  - **group script clauses by agent responsible**
  - **cluster VP actions by goals**
  - **2-5 coarse-grained responsibilities** ➡ **60**
  - **identify collaborators**
  - **assign data on need-to-know basis** ➡ **61**
  - **assert invariants for concept**

Front of Card

Name: object name	**Author**	**Date**	**Version**
Purpose: Statement of purpose, no more than two lines of text, describing the overall rôle or goal of the object-concept identified above			
Responsibilities: Coarse descriptions of purpose, breaking down the overall statement above; The individual goals of the object, rather than each proposed method; Responsibilities for *knowing about*, and for *doing* things; Enumerate 2-5 (general) statements.	**Collaborators:** The objects involved in helping this one to achieve its purpose; The objects to which this object delegates some of its responsibility; Index each sub-contractor object; List participating collaborators, by index, against each responsibility statement.		

Reverse of Card

Classification: Suggested classification of the object concept, such as: unique instance, rôle in a pattern, or class-level concept. Relationship to other concepts, such as is-a, a-kind-of, part-of.

Attributes:

Data attributes discovered for this object, part of its concrete state;

Data assigned on a need-to-know basis to the object which must use it;

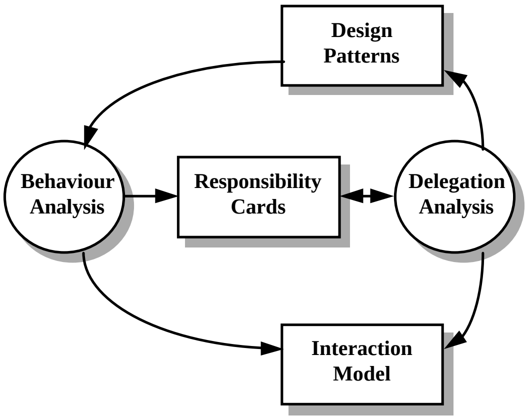
Data assigned to the object which manages and protects the information.

Invariants:

Constraint rules relating the different attributes’ values to each other;

Conditions describing states which this object may not legally enter;

Delegation Analysis



Delegation Analyis

- **Rationale**
  - **second part of behaviour analysis**
  - **break down over-burdened objects**
  - **force decentralisation of functionality**
- **Technique**
  - **focus on levels of responsibility**
  - **apply RDD decomposition rules**
  - **continue until atomic message level**
  - ***Chain of Responsibility* patterns**
  - **build interaction model**

64

66

Decomposition Rules

- **Card Size**
  - **refine to 2-12 atomic services**
  - **over 6 consider, over 12 mandate decomposition**
- **Delegates**
  - **responsibilities are cohesive**
  - **delegate to subcontractors**
- **Peers**
  - **responsibilities may be partitioned**
  - **split into two or more peers**
- **Parents**
  - **responsibilities overlap - hive off to parent**

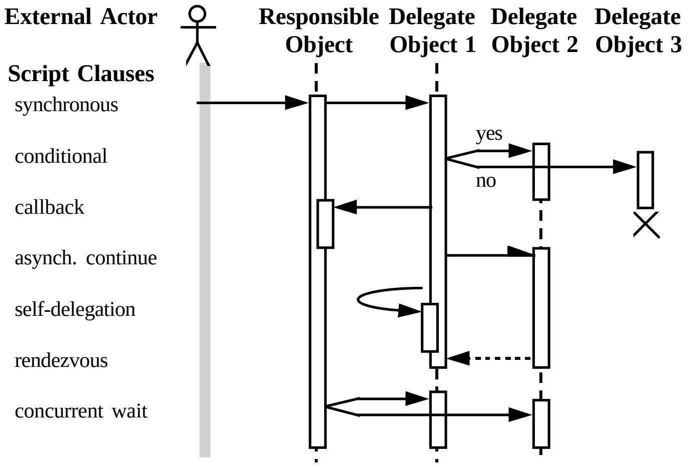
Responsibility Card Checks

- **Client/developer review**
  - **coherent object abstractions**
  - **plausible ownership of responsibility**
- **Cross-checks**
  - **scripts:**
    - **VPs map to atomic services,**
    - **external NPs removed as actors, but tasks managed**
    - **passive NPs demoted to attributes**
    - **active NPs map to objects**
  - **storage objects: data assigned to data-users**
  - **interface/controller objects: logic captured**

Build the Interaction Model

- **Rationale**
  - **complement responsibility cards**
  - **check scripts against responsible objects**
  - **provide message sequencing**
- **Technique**
  - **distinguish internal/external**
  - **list script clauses on LHS**
  - **responsible objects as timelines**
  - **each clause is one stimulus**
  - **recipient is responsible object**

Interaction Model Syntax



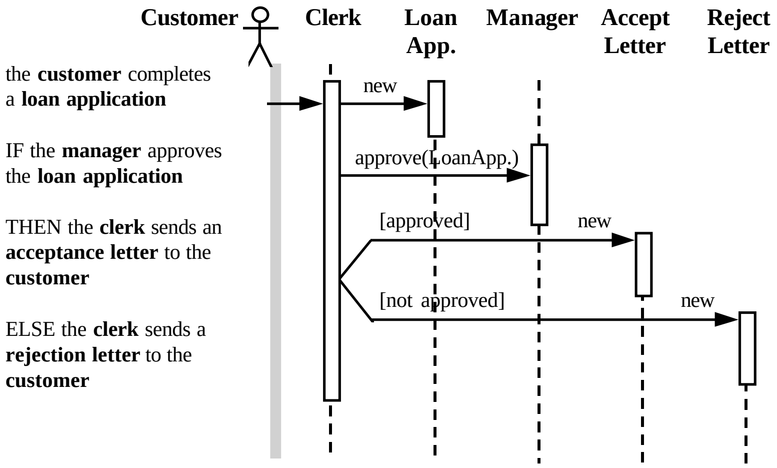
External Actor

Responsible Object | Delegate Object 1 | Delegate Object 2 | Delegate Object 3

Script Clauses

- synchronous
- conditional
- callback
- asynch. continue
- self-delegation
- rendezvous
- concurrent wait

Interaction Model



Customer | Clerk | Loan App. | Manager | Accept Letter | Reject Letter

the **customer** completes a **loan application**

IF the **manager** approves the **loan application**

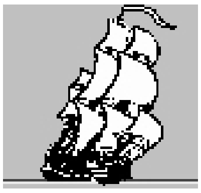
THEN the **clerk** sends an **acceptance letter** to the **customer**

ELSE the **clerk** sends a **rejection letter** to the **customer**

Interaction Model Checks

- **Consistency checks**
  - **Single threads:**
    - **every case managed by one object**
    - **comb or ladder, no returns**
  - **Multiple threads:**
    - **focus of control shifts**
    - **returns to wake up**
- **Cross-checks**
  - **script coverage - VPs map to stimuli**
  - **top timeline maps to main responsibility card**
  - **collaborators map to delegated timelines**

Part Four



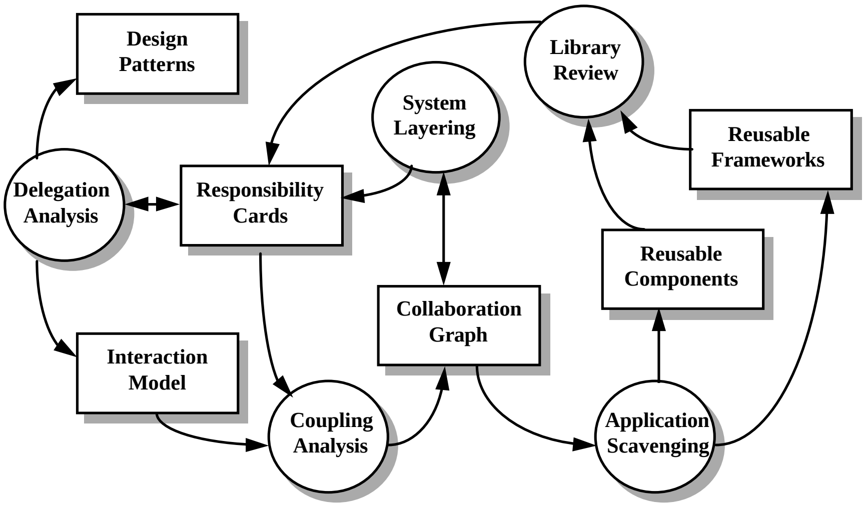
Discovery System Modelling Phase:

Optimising the Architecture in the Context of Frameworks

System Modelling Activities

- ❍ **collaboration diagram (connections)** ➡ 75
- ❍ **cohesion/coupling analysis** ➡ 73, 77
- ❍ **system layering using aggregation** ➡ 78
- ❍ **system layering using generalisation** ➡ 81
- ❍ **feedback from frameworks and components** ➡ 86
- ❍ **application scavenging** ➡ 91

Discovery: System Modelling



Design Patterns

Library Review

System Layering

Reusable Frameworks

Delegation Analysis

Responsibility Cards

Reusable Components

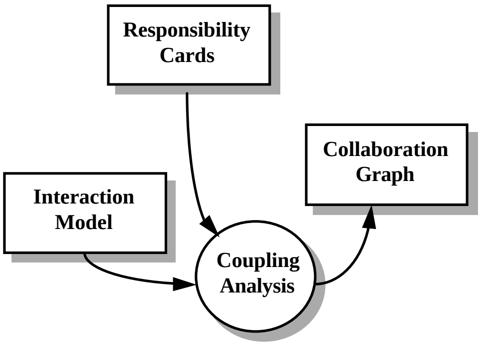
Collaboration Graph

Interaction Model

Coupling Analysis

Application Scavenging

Coupling Analysis



Responsibility Cards

Collaboration Graph

Interaction Model

Coupling Analysis

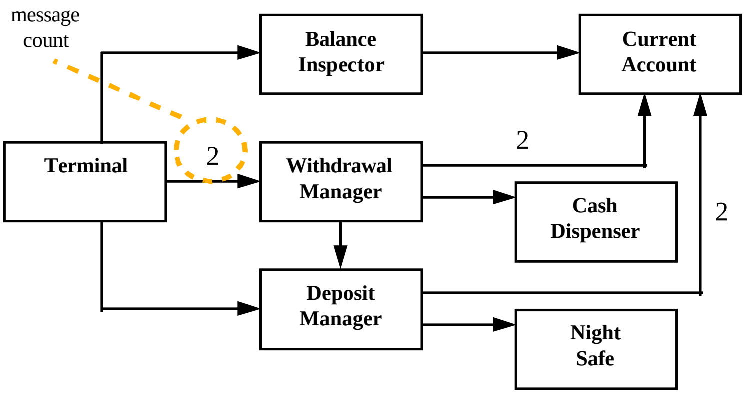
Coupling Analysis

- **Rationale**
  - **group together client-server messages**
  - **overview of total connections by usage**
  - **viewpoint needed for system layering**
- **Technique**
  - **walk through responsibility cards** ➡ 59
  - **trace through interaction model** ➡ 66
  - **build collaboration graph:**
    - **add new arcs for new client-server connections**
    - **increment usage counts on old connections**

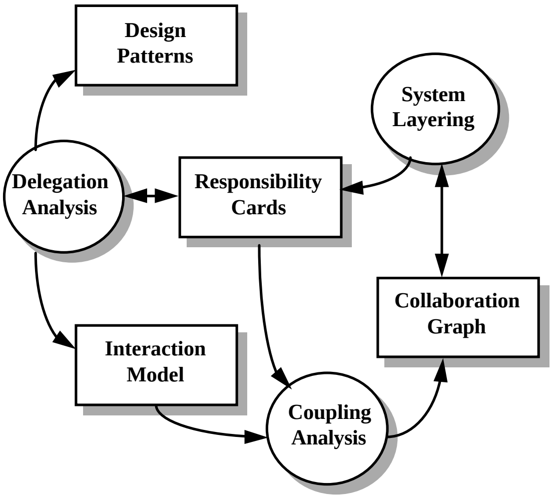
➡ 75

Collaboration Graph

- ❍ **overview of client-server connnections**
- ❍ **NOT snapshot of instance messaging**



System Layering



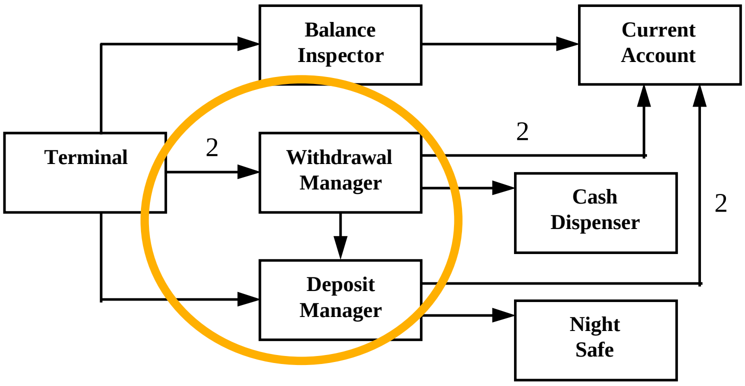
System Layering

- **Rationale**
  - **introduce modular hierarchy**
  - **reduce inter-module coupling**
  - **low complexity = easy implementation**
  - **low dependency = more reuse**
- **Technique**
  - **aggregate over tightly-coupled subsystems** ➡ 78
  - **generalise over commonly-invoked services** ➡ 81
  - **generate new coordinating objects**
  - **update responsibility cards, interactions**
  - **harvest new Design Pattern examples** ➡ 91

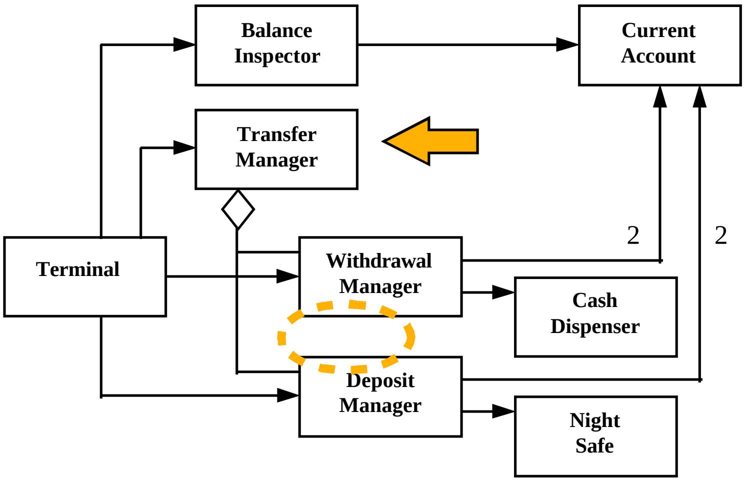
Aggregate over Coupled Subsystems

- **What to look for**
  - **closed loops, linked rings, doubly-linked rings in the collaboration graph**
  - **choose to enclose the collaborations that you wish to eliminiate (eg with high per-service counts)**
- **What to do**
  - **invent new object abstraction - *Mediator* - that aggregates over components**
  - **remove inter-component collaborations**
  - **subsystem services migrate up to *Mediator***
  - ***Mediator* communicates with components**

Aggregate over Coupled Subsystems



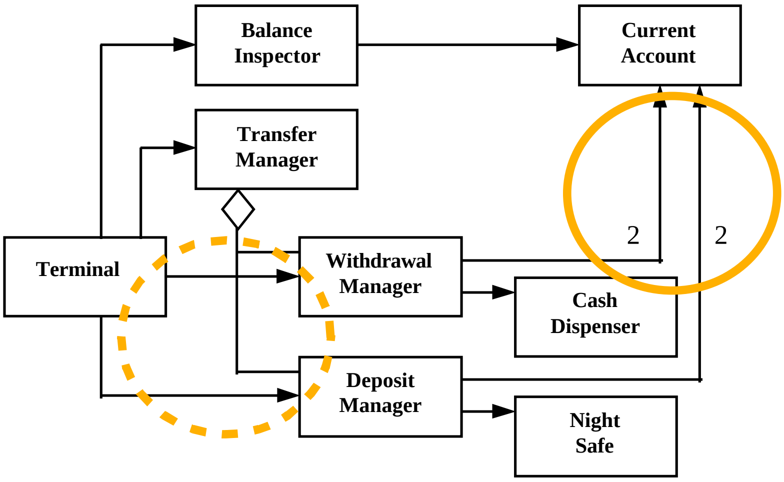
Aggregate over Coupled Subsystems



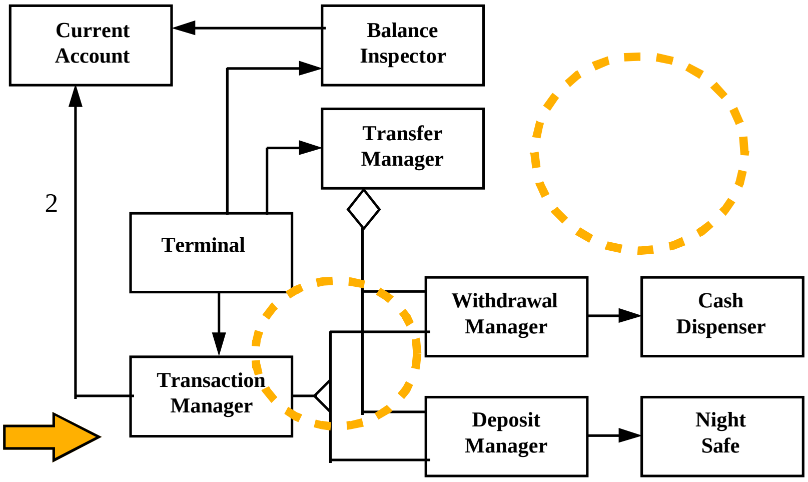
Generalise over Common Services

- **What to look for**
  - **objects whose interfaces overlap (server-end)**
  - **objects that invoke overlapping services (client-end)**
- **What to do**
  - **invent new abstract parent, often a *Command*, that generalises over the components**
  - **remove component client/server collaborations**
  - **component collaborations migrate up to parent, become part of a *Template Method***
  - **Components redefine parts of the *Template Method* (eventually, using dynamic binding)**

Generalise over Common Services



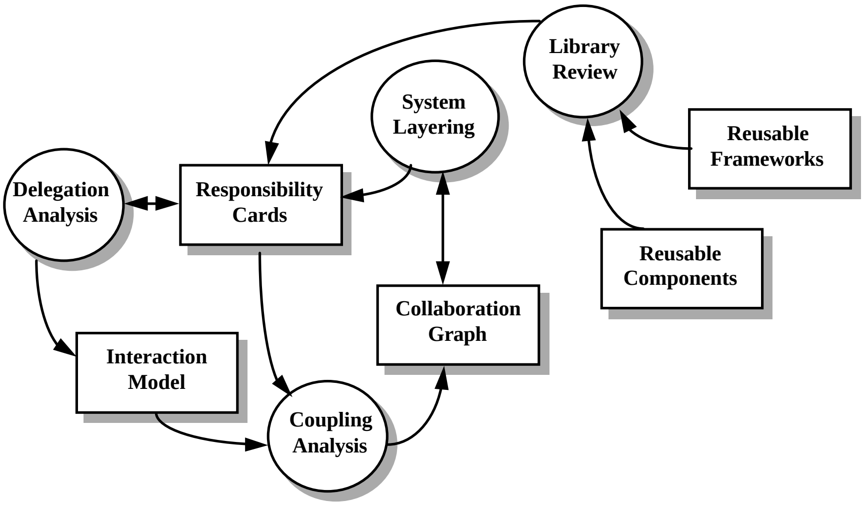
Generalise over Common Services



System Layering Checks

- ❍ **Client reviews**
  - ❍ **new abstractions may be meaningful**
  - ❍ **may have data, states, life history**
- ❍ **Cross-checks**
  - ❍ **equivalence of services, before and after**
    - ❍ **check Mediator interface, collaborations**
    - ❍ **check Template Method hot-spots, binding**
  - ❍ **subsystem objects have responsibility cards**
  - ❍ **lifelines modified to reflect new pattern**

Library Review



Components

- ❍ **Definition**
  - ❍ **cohesive software element**
  - ❍ **reusable part (dual of framework)**
  - ❍ **directly implemented, available**
- ❍ **Coverage**
  - ❍ **domain-dependent (business object)**
  - ❍ **domain-independent (GUI, basic object)**

Frameworks

- **Definition**
  - **domain-specific harness or shell**
  - **reusable whole (dual of component)**
  - **implementation (documented by Patterns)**
- **Coverage**
  - **horizontal: common layer or substrate**
  - **vertical: narrow business domain**
- **Architectures**
  - **white-box: inheritance, effective subclasses**
  - **black-box: composition, components/interfaces**

➡ 92

Library Review

- **Rationale**
  - **maximise reuse of designs**
  - **plan for component integration**
  - **plan for framework integration**
- **Technique**
  - **examine application control structure** ★
  - **compare with existing framework(s)**
  - **good match: import framework, rework the current design (cards, interactions, collaborations)**
  - **poor match: schedule framework(s) for reengineering, deliver to existing design**

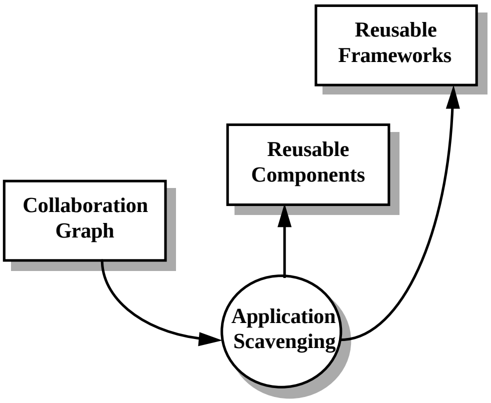
Business Decisions

- **Alternatives**
  - **import frameworks before system layering stage**
  - **layer system before considering frameworks**
- **Decision criteria**
  - **cognitive: framework may bias perception of current system (eg OO-toaster)**
  - **historical: maturity of existing framework**
  - **pragmatic:**
    - **real needs of the current system**
    - **cost of importing existing framework**
    - **cost of reworking existing framework**

Library Reuse Checks

- **Cross-checks**
  - **equivalence of services, before and after**
    - **no application service omitted**
    - **no dangling hot-spots/missing methods**
  - **control structure preserved (white-box)**
    - **only hot-spots have been specialised**
    - **no internal dispatch points overwritten**
    - **state machines of subclasses only introduce substate machines, or orthogonal states (no overlapping)**
  - **control structure preserved (black-box)**
    - **components supply expected interface**
    - **component methods pre-/postconditions satisfy “programming by contract” rules**

Application Scavenging



Application Scavenging

- **Rationale**
  - **harvest new components**
  - **refactor existing framework(s)**
- **Technique**
  - **proceed after 3+ applications**
  - **generalise on business objects, control strategy**
  - **white-box during evolutionary phase**
    - **new *Command, Template Method***
    - **multiple levels of generalisation**
  - **black-box when control stabilised**
    - **refactor layer of controller objects**
    - **plug-in points for business objects**

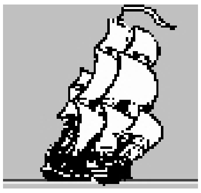
Costs of Refactoring

- **Refactor Class**
  - **O(1): add method, add attribute**
  - **O(n^2): reorganise internal dependency**
- **Refactor Inheritance**
  - **O(n): add extended class - specialise**
  - **O(n^2): add overlapping class - generalise, migrate**
- **Refactor Collaboration**
  - **O(x^n): too many specialisation dimensions - extract and encapsulate one degree of variation**
  - **O(x^n): new focus of control - different patterns of collaboration**

Developer Rôles

- **Application Developer**
  - **focus: current application**
- **Reuse Manager**
  - **focus: mapping onto components**
- **Librarian**
  - **focus: defence of frameworks**
- **Application Scavenger**
  - **focus: extraction of components**

Part Five

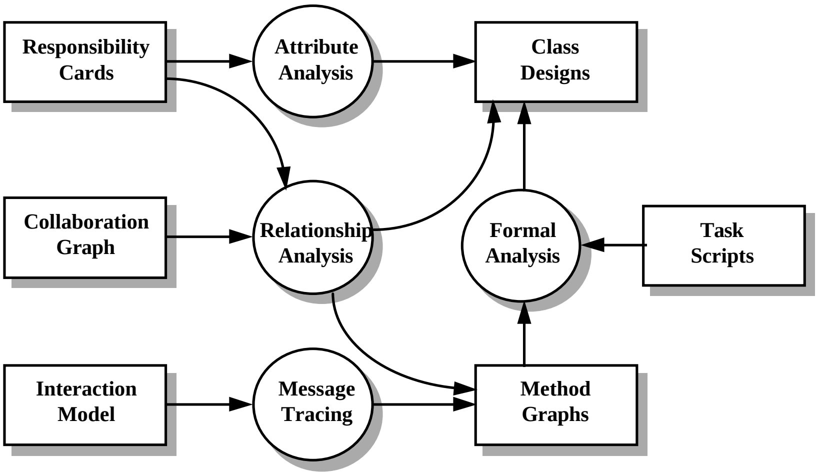


Discovery Language Modelling Phase: Detailed Design for Implementation

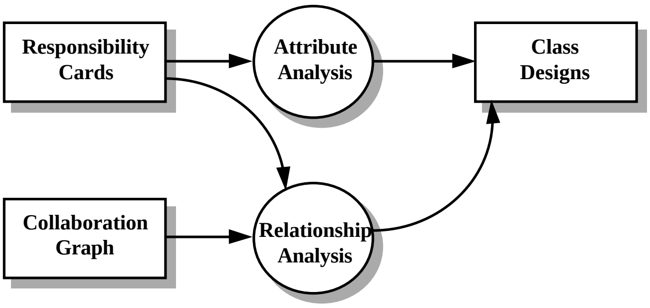
Language Modelling Activities

- ❍ **lifetime and visibility analysis** ➡ **101, 103**
- ❍ **attribute/relationship structure** ➡ **98, 100**
- ❍ **method execution graphs (end-to-end computation)** ➡ **104**
- ❍ **method signature specification** ➡ **107**
- ❍ **method pre- and postconditions** ➡ **107**
- ❍ **class specification sheets**

Discovery: Language Modelling



Attributes and Relationships



Attribute Analysis

- **Rationale**
  - **extract managed data attributes**
  - **enforce encapsulation rules**
  - **maintain data invariants**
- **Technique**
  - **prepare class design spec. sheets**
  - **transfer attributes from responsibility cards**
  - **mark up according to encapsulation rules**
    - **private, or protected? not public!**
  - **transfer invariants from responsibility cards**

Relationship Analysis

- **Rationale**
  - **extract references, embedded objects**
  - **establish creation/deletion order**
  - **enforce constant rules**
- **Technique**
  - **determine client/server collaboration semantics**
    - **input: exclusive, constant, connascent?**
    - **output: embedding, references, constants**
  - **determine client/server lifeline closures**
    - **input: lifelines, collaboration semantics**
    - **output: create calls, create args, GC strategy**

Collaboration Semantics

- **Permanent/Temporary**
  - **already decided by the collaboration graph**
- **Exclusive/Shared**
  - **whether > 1 client may access server**
- **Constant/Mutable**
  - **whether server may be replaced**
- **Independent/Connascent**
  - **whether server is born and dies with client**

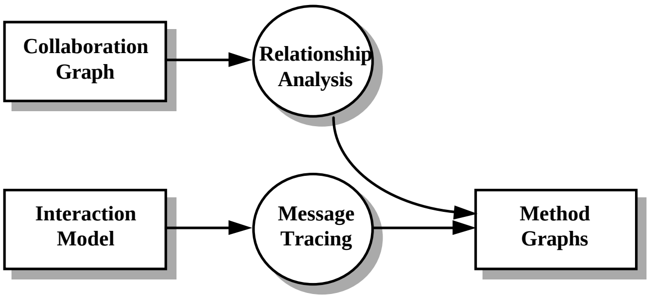
Collaboration Semantics

- **References vs Embedding**
  - **connascent + exclusive => client may embed server, or call create/delete**
  - **otherwise, clients must reference server (or, a distinguished client may embed server)**
  - **add embedded objects/ references to class design specs.**
- **GC Strategy?**
  - **connascent + shared => clients keep refcount on server for GC, delete when zero**
  - **otherwise, clients forget server, external object or system controls server lifetime**

Lifeline Closure

- **Relative lifetimes**
  - **closure visible from interaction diagrams**
  - **cross-check: connascence + mutability**
- **Construction**
  - **server lifeline contained: client embeds, or calls create/delete on server object**
  - **server lifeline not contained: client constructor accepts a server-argument**
  - **add constructors to class design specs.**

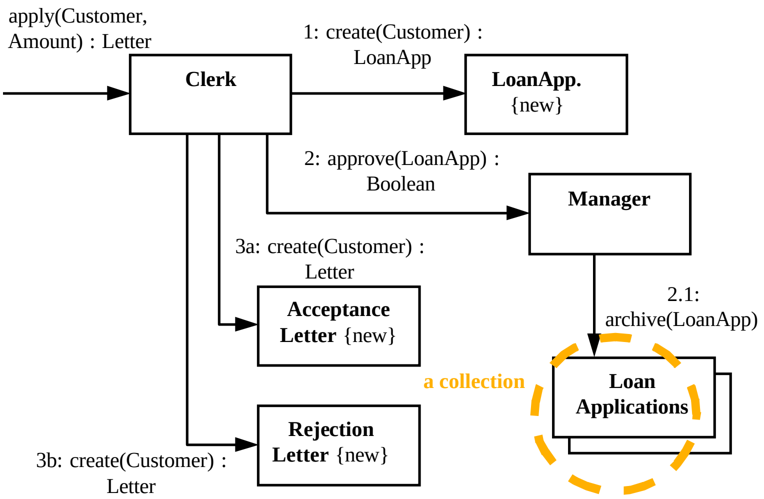
Methods



Message Tracing

- **Rationale**
  - **complete end-to-end processing**
  - **counteract yo-yo effect**
  - **documentation for implementors**
- **Technique**
  - **trace system operations through interactions**
  - **build end-to-end method execution graph**
  - **determine method arguments:**
    - **collaborations = permanent references**
    - **anything else = additional arguments**
  - **mark creation, distribution semantics**

Method Execution Graph



apply(Customer, Amount) : Letter

Clerk

1: create(Customer) : LoanApp

LoanApp. {new}

2: approve(LoanApp) : Boolean

Manager

3a: create(Customer) : Letter

Acceptance Letter {new}

2.1: archive(LoanApp)

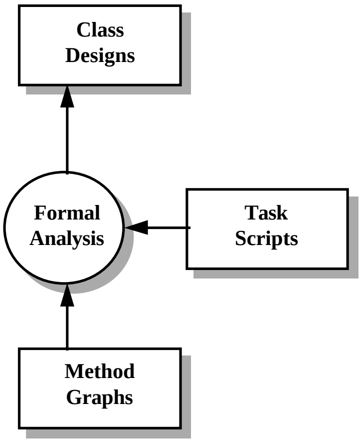
a collection

Loan Applications

Rejection Letter {new}

3b: create(Customer) : Letter

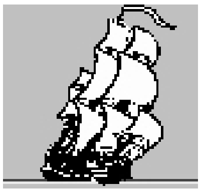
Formal Analysis



Formal Analysis

- **Rationale**
  - **guarantee contract specification**
  - **correct operation of methods**
  - **specification for testers**
- **Technique**
  - **transfer method protocols to class spec. sheets**
  - **check for partial semantics against scripts**
  - **specify method preconditions**
  - **check for success criteria against scripts**
  - **specify method postconditions**

Object Discovery



Thank-you for your participation;

I hope we learned some useful things from each other!