Daniel **Fitz**
43961229

University of Queensland
CSSE3002 – The Software Process

Lecture Notes

Table of Contents

Software Engineering

Application of a systematic, disciplined, quantifiable approach to the development, operation, and maintence of software.

- This is, the application of engineering to software
- IEEE Standard 610.12-1990 Concerned with theories, methods and tools that enable professional software development. “The topic that we call software engineering is both exciting and frustrating. Exciting because it draws on many technical disciplines and provides a harness that binds each discipline to the next. Frustrating, because it demands knowledge in a multitude of topic areas and seems to be infinitely expandable.” (Roger Pressman, 1992) Discipline behind the process (you actually know what you’re doing and why) Being able to
- choose appropriate tools and techniques
- work effectively in a team
- manage your own work and the process

Software Engineering Process

A structured set of activities followed to develop software system.

- Tools
- Methods
- Practices

A lot of companies have fucked up

Well Engineered Software

- Usable
- Dependable
- Maintable
- Efficient
- How do costs come into this?
  - Trade-offs may be involved
    - appropriate
    - cost-effective

Process Models

- Abstract representation of a process
- Plan Driven
  - Structured / Traditional
- Incremental
- Agile
- Lean
- Formal

Plan Driven Processes

- Waterfall
- V Model
- Spiral

Waterfall

- Introduced iteration between phases

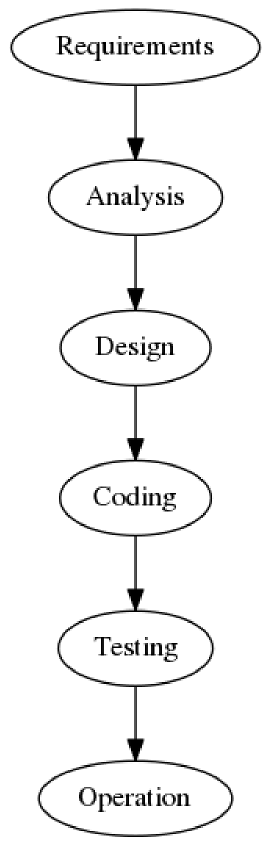

Figure 1: Diagram explaining Waterfall

- Prototyping
  - Requirements
  - Design
- See: http://www.allaboutagile.com/dr-royce-and-waterfall/

V Model

Spiral

- Focus on process control
- See http://csse.usc.edu/TECHRPTS/1988/usccse88-500/uscsse88-500.pdf

Incremental Processes

- Unified Process
- OPEN
  - Object-oriented Process Environment and Notation

Unified Process

Unified Process is allied closely with UML

- Four distinct phases
  - Inception, Elaboration, Construction and Transition
- Considers activity balance across workflows and phases

OPEN

- Process framework
  - process is instantiated from the framework
  - metamodel documents the framework
- Contracts between components
  - process Construction
  - scheduling

Agile Processes

- Embrace change
  - Requirements are never fixed
  - Stop pretending, and get used to it
- Deliver early and deliver often
  - A working system delivers value
    - telephone book scale documentation does not
  - A deployed system generates revenue
  - 80:20 rule

Lean Development

- More a philosophy than a process
  - Think big
  - Act small
  - Fail fast
- Eliminate Waste
- Amplify Learning
- Decide as Late as Possible
- Deliver as Fast as Possible
- Empower the Team
- Build Integrity In
- See the Whole

Formal Processes

- Application of mathematical formality to software development
  - formal specification

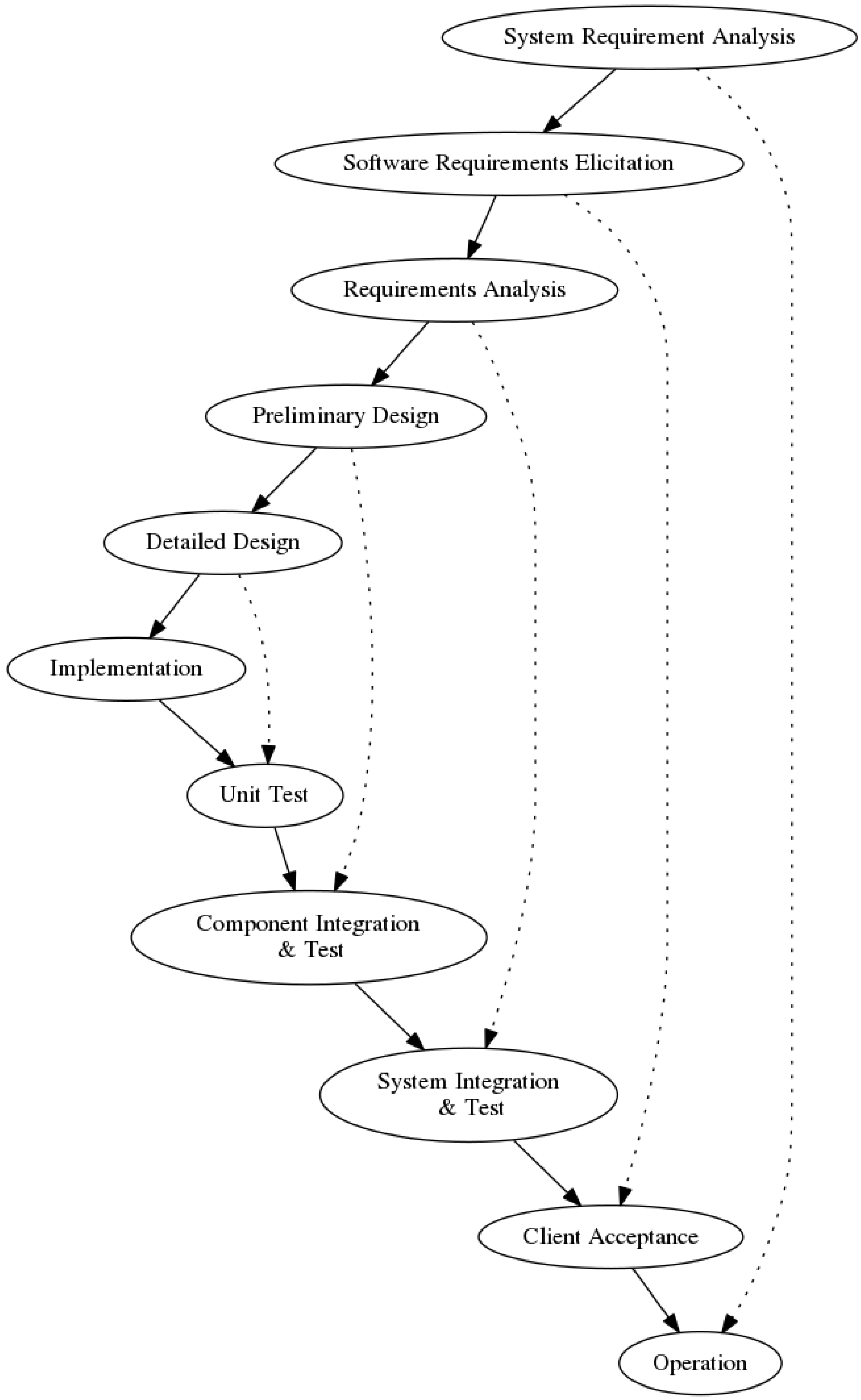

Figure 2: Diagram explaining V Model

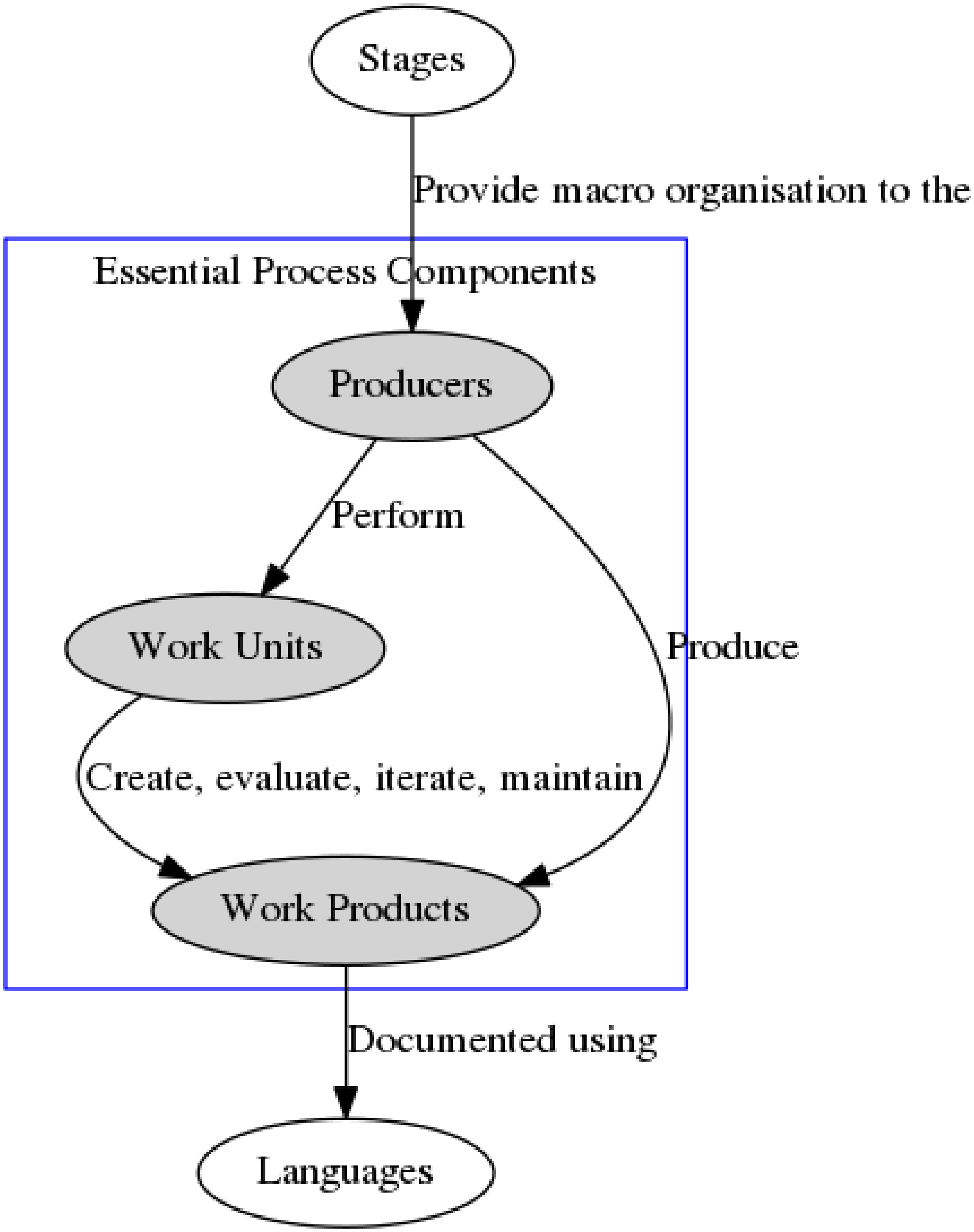

Figure 3: Diagram of OPEN Process

    - transformation of specification to code

Process

- All Software Engineering Processes involve phases
    - Requirements
    - Design
    - Development (implementation, coding)
    - Testing (Verification)
    - Delivery and Maintence
- These are never disjoint, never just sequential
- We iterate between them, and we blur the distinctions because we want to get it right
- Software Engineering cannot work without a defined development process
    - anything else is randomised hacking
- Processes cannot work if they are not usable
    - people don't read telephone books cover to cover
- Good processes should engage the team
    - support technical excellence and innovation
    - embed a culture of trust and reponsibility
        * no place to hide

Standards

- Rules, guidelines and heuristics
    - assist in achieving "good" practice
        * not best practice
- De facto - implicit agreement
    - easily changed
- De jure - formal agreement
    - usually debated & documented

Standard Adoption

- Voluntary
    - achieving good practice
    - safety net
- Required
    - demands of clients
    - certification requirements
    - follow on from other standards
    - process improvement activity

SE Standards

- Normative and informative
- Document centred
- Adaptable

Main SE Standards

- ISO/IEC 122207:2008
    - *Systems and software engineering – Software life cycle processes*
    - Framework for lifecycle modelling

- Focus on bespoke software
  - including product and services
- Includes process for defining, controlling and improving software processes

(Last reviewed in 2013)

- ISO/IEC/IEEE 15288:2015
  - *Systems and software engineering – Software life cycle processes*
  - Framework for process descriptions
  - Focus on system engineering
    - software as a component of system
  - Focus on bespoke system development
  - Includes process for defining, controlling and improving processes

(Ratified in 2015)

- ISO/IEC/IEEE 15289:2015
  - *Systems and software engineering – Content of life-cycle information items (documentations)*
  - Standard project documentation
  - Focus on purpose and content
    - not necessarily a formal document
      - e.g. central data repository

(Ratified in 2015)

Compare 12207 and 15288

- 15288 focusses on systems
  - hardware, software, people, facilities, material, …
- 12207 focusses on software
  - intended to be used for software component of 15288

IEEE Standards

- Terminology
- QA Plans
- Configuration Management
- Requirements Specification
- Unit Testing
- V&V
- Reviews & Audits
- Productivity Metrics
- Quality Metrics
- Project Management Plans
- User Documentation
- Maintence

Ethics

Code of Ethics

- Agreed standard of behaviour
- Mark of professionalism
  - most professional bodies have one
- Enforceable?

ACS

- Primary of Public Interest
  - place interests of public above personal, business or sectional interests
- Enhancement of Quality of Life
  - strive to enhance quality of life of those affected

- Honesty
  - honest representation of skills, knowledge, services and products
- Competence
  - work competently and dilidently for stakeholders
- Professional Development
  - enhance your own development and your colleagues and staff
- Professionalism
  - enhance integrity of the ACS and respect of members for each other

Requirements Project

- Software Requirements Specification
  - template will be provided
  - contents
    * functional & non-functional requirements
    * requirements & analysis models
    * risk management plan
    * size, time & cost estimates
    * UAT
- Stakeholder's Project Outline

eVisa Issues

- Security
- Privacy
- Translation
  - Auto and Manual
- Interfaces to other departments & authorities
- Data Mining
  - Applications
  - Applicants

eVehicle

- Battery swap for electric vehicles
- Batteries owned separately to vehicles
- Swap done at a service station
- Charged for energy used
- Battery can be charged in vehicle
  - Vehicle owner pays maintence fee to battery owner

eVehicle Issues

- Tracking batteries
- Billing system
  - Vehicle owners
  - Battery owners
  - Service owners
  - Energy suppliers
- Security
- Privacy

Project Charter

- Goal & Objectives

- reason for doing work
- Summarises project strategy
    - from a business perspective
- Set direction for project
- Scope
- Generate "buy-in"
- Usually produced by sponsor

Vision Statement

- Long term purpose of system (a bit idealistic)
- For – target audience
- Who – statement of need
- The – product
- Is – category
- That – reason to use
- Unlike – alternative
- Our product – advantage

Goals

- High-level
- What project will accomplish
- One sentence (sometimes more)

Objectives

- Specific
- Supports a goal (think "how" it does this)
- Describe with an action verb
    - measurable
    - address project end result

SMART

Specific

- what is to be accomplished
- only essential aspects

Measurable

- need success/completion criteria

Agreed-upon

- common understanding amongst stakeholders

Realistic

- achievable with available resources

Time-based

- realistic deadline

Business Benefits

- High-level but concrete
    - Increased revenue
    - Reduced costs
    - Improved efficiency
    - Improved customer satisfaction
    - ...

Scope

Focuses on what needs and doesn't need to be done. What hasn't been figured out yet. Project wide

- What is to be delivered
    - by end of project
    - releases determined later

- What is explicitly out of scope
  - does not relate to business benefit

Stakeholders

Who is interested in this project - Sponsor (The main person in the company who has the vision for where they want this project to go) - Influencers (Have good insights into the way this project should head) - Users - key (People who use this the most or get the biggest benefit) - restricted (Will use the system but their use of system will be restricted) - super (Administrator or power user, get special permissions) - Anti-Users (Who should not use the system, prevent them from accessing) - Other

Assumptions

- Expected to occur
  - Air Traffic Controller's workstations will be updated before system is rolled out

Constraints

- Restrictions
  - project
  - development team
- *Must run on existing web server infrastructure*

What is Requirements Engineering?

- Requirements engineering is a term often used for a systematic approach to acquire, analyse, validate, document and manage requirements
- Typically implemented as a cyclic or iterative process
- Requirements validation may include prototype Construction and evaluation
- Applied at both system and software levels, often with interleaved system architecture design

What is a Requirement?

1) A condition or capability needed by a user to solve a problem or achieve an objective
2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document
3) A documented representation of a condition or capability as in 1 or 2

- From IEEE Standard Glossary of Software Engineering Terminology

Requirements Engineering Products

- Primary outcome is a requirements specification
  - Essentially a contract between user and developer
  - Basis for all subsequent development and verification processes
- Secondary outcome is usually system and software acceptance test criteria

Why is RE important?

- Most faults observed in a software project are from incorrect, incomplete, or misinterpreted functional specifications or requirements.
  - Glass' Law *It is expensive to fix an error the later it occurs in a project*
- Helps earlier detection of mistakes, which are much more costly to correct if discovered later
- Forces clients to articulate and review requirements
- Enhances communications between participants
- Helps record and refine requirements
- All this is about producing good requirements

What happens if the Requirements are Wrong?

- System may be delivered late and cost more than originally expected
- Customer and end-users are not satisfied with the system
  - May not use its facilities or may even decide to scrap it altogether
- System may be unreliable in use with regular system errors and crashes disrupting normal operation

- If the system continues in use, the costs of maintaining and evolving the system are very high

Benefits of Good Requirements

- Agreement among developers, customers, and users on the job to be done and the acceptance criteria for the delivered system
- A sound basis for resource estimation
  - cost, personnel quantity and skills, equipment, and time
- Improved system usability, maintainability, and other quality attributes
- The achievement of goals with minimum resources
  - less rework, fewer omissions and misunderstandings

Product vs User Centred

Product-Centred

- Focus on features to be delivered
  - expect users will use features to complete tasks

User-Centred

- Focus on anticipated usage
  - what do users need to accomplish
- Reveal necessary functionality
- Assists with prioritisation

Advice / Perspective

- … a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system [RUP]
- In practice it is impossible to produce a complete and consistent requirements document [Somerville]
- Getting the requirements right is critical for success
- Requirements are rarely right at the start of a large project
  - Expect change
  - Manage it
  - Agile mantra "Embrace Change"

Functional Requirements

- Requirements (or capabilities) for functions (specific behaviour) that must be performed by the system
  - e.g. read a bar code, change a username, maintain a temperature
- Primary focus of most requirements activities

Non-Functional Requirements

- Constraints on performance or quality
- **Product Properties**
  - Requirements on the behaviour of the product
    - System shall process a minimum of 8 transactions per second
    - User credit card details shall be secured
- **Process Properties**
  - Requirements on the practices used to develop / produce the system
    - Control software shall be verified in accordance with IEEE STD 1012-1998

Classification of Non-Functional Requirements

According to the ISO standard - Safety requirements - Security requirements - Interface requirements - Human engineering requirements - Qualification requirements - Operational requirements - Maintence requirements - Design constraints

Non-Functional Requirements Examples

- Safety Requirements
  - The system shall not permit operation unless the operator guard is in place
- Security Requirements
  - Only the system administrator can change system data
  - All system data must be backed up every 24 hours
- Interface Requirements
  - The system's interaction with other existing or proposed systems
    - e.g. specific databases, API's for other systems
- Human Engineering Requirements (usability)
  - Adequacy requirements – system does what is required
  - Learning requirements – time needed to learn the facilities of the system
  - (Error) handling requirements – error rate of the end-users
  - Recovering requirements – time to restart after system failure
- Qualification Requirements (set V&V targets)
  - The Queensland government requires certification to the quality standard ISO 9001 (Quality management systems) for its major software suppliers
- Operational Requirements
  - (Components of) an industrial or military control system may have to withstand extreme heat or cold, to survive vibration, movement, sudden impact, etc
  - System efficiency or performance
    - limits on memory and processor speed are often consequences of the operational environment
- Maintence Requirements
  - Software readability, flexibility/portability and testability
- Design Constraints
  - Use database X because of user familiarity
  - Use algorithm Y for function Z because of user preference

Requirements

- There is a relationship between the quality / cost / timeliness / etc. of the product and the quality of the process
- Both functional and non-functional requirements are essential for successful software
  - Both must be verified
    - Consequently they must be testable
      - Non-functional Requirements should be measurable

The Requirements Engineering Process

See Figure 4

Who does it?

User organization (Domain competent)

Universities prepare requirements specifications for student enrolment systems (Si-net)

Developer organisation (IT competent)

IT companies (e.g. Technology 1) prepare requirements specifications for their clients

Third party organisation (Domain and IT competent)

Consulting companies (e.g. Accenture) prepare requirements specifications for their clients

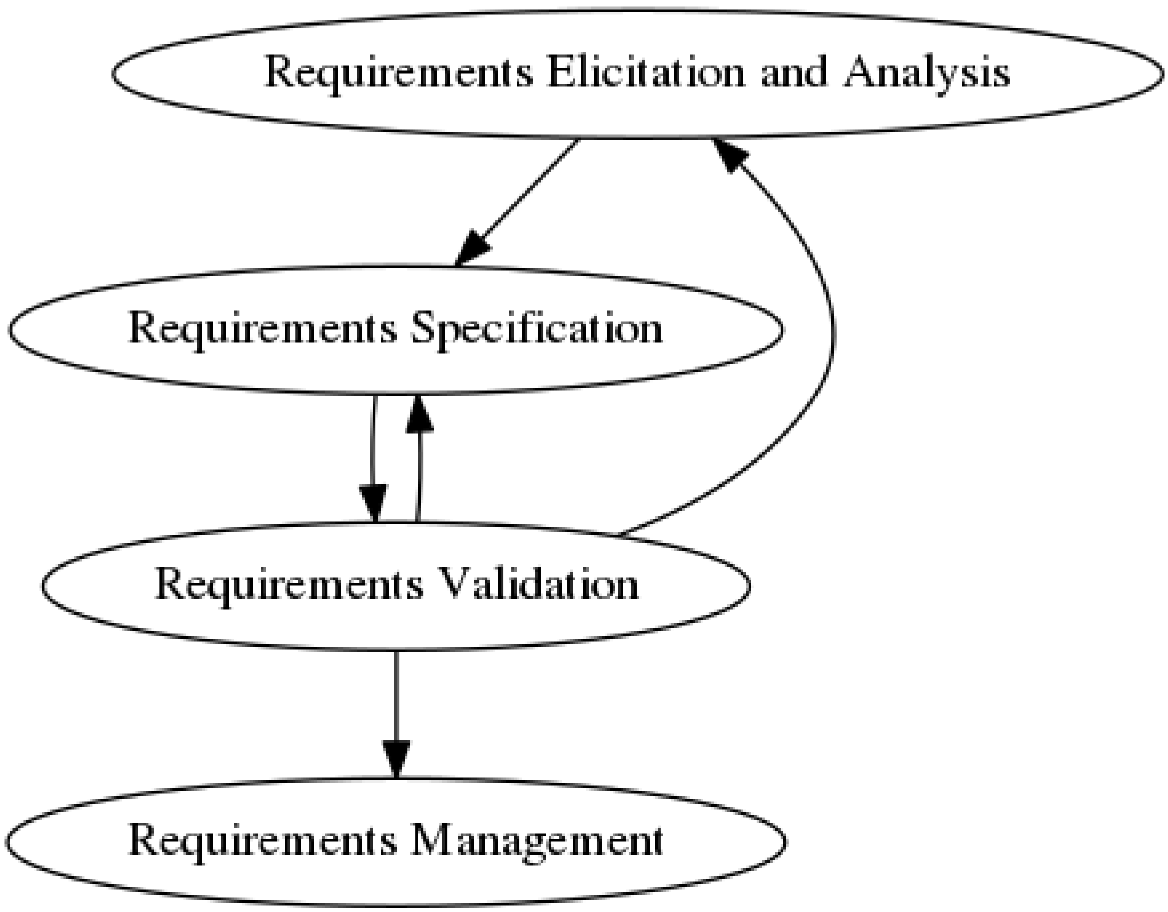

Figure 4: The Requirements Engineering Process

Sources of Requirements

Users

e.g. customers or end-users – user requirements

Non-Human Sources

e.g. other devices or systems in the environment

Other Stakeholders

e.g. marketing experts, regulators, managers, business owners, developers

Business Value Not System Requirements

Understanding the Business

- Developers and Stakeholders need a shared understanding of the project's purpose
  - easier when they collaborate continuously
- Focus on value to be delivered
  - not just the requirements
- Enables better decisions, designs and suggestions
  - developers are part of the value chain
    - not just serving it

Business Model

A business model describes the rationale of how an organisation creates, delivers, and captures value

Canvas

The canvas can be broken down into these key areas: - Key Partners - Key Activities - Key Resources - Value Provided - Customer Relationships - Channels - Customers - Revenue - Costs

Customers

- Personas
- Who's Impacted
- Stakeholders

Value Proposition

- Use Cases
- Specification by Example
- Customer Savings / Revenue
- Improvements
- Experience Improvements

Channels

- Systems
- Methods
- Related Features

Relationship

- Direct / Indirect
- Human / Automated
- Assisted / Self Service
- Individually / Collaboratively

Revenue Streams

- Opportunity
- Savings

- Profit
- Improvements

Key Resources

- Systems
  - Primary
  - Secondary / Connected
- Team
  - Development
  - Business

Key Activites

- Use Cases
- Who's Activities
- Connected Activities

Key Partnerships

- Development Team
- Business
- Secondary / Connected Teams
- Impacted Teams
- Related Teams

Cost Structure

- Opportunity
- Development Estimates
- Quantity of Customers

Requirements Elicitation

Elicitation Process

Preparation – Sources of Requirements

- Stakeholders
- Users
  - Identify classes of users
  - How will they use the system?
- Environment
  - Application domain
  - Organisation
  - Operations

Know Your Users – User Role Modelling

- What types of people will use the system?
  - each will have different goals
- Don't think of an anomymous user
  - over simplification
- Identify different user roles
  - brainstorm initial set
  - group related roles
  - consolidate roles
  - refine roles
- Don't get stuck on organisational roles

Personas

- Fictitious character representing a user role

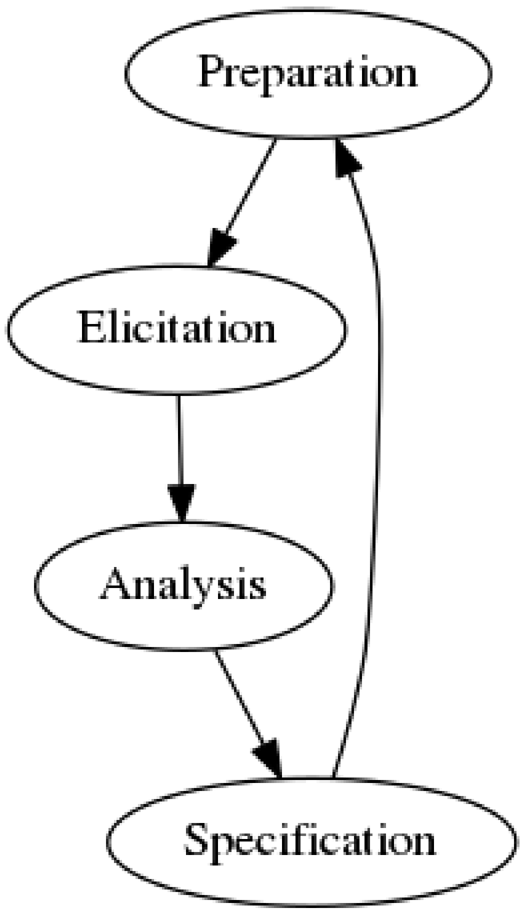

Figure 5: Elicitation Process

- Makes important roles more realistic
  - mock person, including photo and profile

Application Domain

- Knowledge of area in which system is used
- Sources
  - manual
  - books
  - journals
  - users

Organisation

- Structure
  - most IT systems reflect organisation structure
- How fixed is the structure
  - is the system meant to change it?
- Policies and practices

Operations

- Other system dependencies
  - interface requirements
  - timing constraints
- Execution environment
  - platform
  - reliability and performance
- Criticality
  - mission
  - safety

Elicitation Challenges

- Stakeholders and users may not be able to describe their tasks well
  - make assumptions and leave things unstated
- No-one knows everything
- Requirements conflict
- Implicit requirements

Elicitation Techniques

- Interviews
- Workshops
- Focus Groups
- Observations
- Questionnaires

Interviews

- Effective for understanding problem and eliciting *general* requirements
- Prepare questions in advance
  - discussion needs a starting point
  - primarily open-ended questions
  - strawman model if you have some data
- Suggest ideas and alternatives
  - users may not realise what is possible
- Active listening
  - paraphrase what you understand
- Clarify what's unclear
  - draw me a diagram
  - card sorting

- Maintain focus

Workshop

- Structured meeting
  - formal roles
  - clear goals
- Multiple stakeholders
  - resolve conflicting requirements
  - quickly gather broad system usage

Focus Groups

- Less structure
  - still need clear goals
- Exploratory discussion
  - needs
  - preferences
  - expectations
- Broad stakeholder representation
- Gather broad-based ideas

Observations

- Observe how users perform their tasks
  - learn workflow
- Users often cannot describe everything they do
  - too many fine details
  - habitual tasks
- Time consuming
  - silent observation
  - interactive

Questionnaires

- Inexpensive and easily administered to remote sites
- Collect data from many users
- May feed into interviews or workshops
- Good questionnaires difficult to write

Good Questionaires

- Answer options for all possibilities
- Answer choices mutually exclusive
- Avoid phrasing that implies a correct answer
- Closed questions for statistical analysis
- Open questions to gather ideas
- Keep short

Independent Elicitation Techniques

- Discover information on your own
- System interface analysis
- User interface analysis
- Document analysis

System Interface Analysis

- Look at other system's functionality
  - what does your system need to do?
  - what can you use?
- Data exchange
  - including formats and validation

- Services

User Interface Analysis

- Study existing systems
  - what do they do?
  - how are they used?
- What should be replicated and avoided?
- Good way to learn existing system and processes

Document Analysis

- Business process descriptions
- Existing system documentation
  - user manuals
  - specifications
  - what must be kept
  - what can be improved
- Industry standards or legislation
- Gain understanding of domain or system

Soft Skills

- Active Listening
- Interviewing and Questioning
- Facilitation
- Negotiation
- Observation
- Writing
- Organisation
- Interpersonal
- Creativity

Use Case Modelling

- Models and documents the functional requirements of a problem domain
- Results in the producation of the
  - functional requirements
  - which are the detailed, role based, functional account of the requirements

Background

- Use cases were first described by Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case-Driven Approach. Addison Wesley, 1992
- Subsequently incorporated into numerous other methods
- Basic technique for capturing user requirements

Why Use Cases?

- Formalises users' expectations of what the system is to do and how the system is to be used
- Easy technique to understand
  - documents actual paths through the system
- User-driven process
  - encourages user involvement
- Basis for scoping and prioritisng development work
- Basis for acceptance testing
- Well aligned with Business Process Modelling

Actors, Use Cases and Association

Actors: things outside of the system that interact with the system (*stick figures*) Use Cases: features of the system that an actor uses (*circle things*) Associations: indicates a relationship between an actor and a use case (*a line*)

Actors

- Everything that interacts with the system
  - people as well as other systems
  - important in defining the boundary of the system
- Not described in detail (they're outside the system)
  - though their interface is usually defined
- Normally act in several use cases
- Represents a role that a user can play
  - many users are represented by a single actor
  - one user can be different actors at different times
    - performs different use cases depending on role

Actors and Users

- A user plays the role of an actor
  - an example of an actor is a ”clerk“
  - an example of a user is ”John Smith“ – an actual clerk
- A particular user may play the roles of many different actors
- **Actors are External to System**
  - Make sure that actors are people or other systems that would actually use the system (*Not things that would be modelled within the system)
  - The system does not use itself
  - The use cases describe what the system does
- **Actors are Not Technical Choices**
  - A DBMS is a particular storage stategy
    - it is a design decision
    - it should not appear in the requirements
- **Actors Interact with the System**
  - Actors interact with the system
    - use it to perform a task
    - provide a service used by the system
- **Primary and Secondary Actors**
  - Primary actors are those actors that the system is designed to serve
    - main users of the system
    - represent the users for whom the system is designed
    - focus for modelling the system
  - Secondary actors are support roles
    - secondary actors only exist so that primary actors can use the system
      - administration
      - services

What is a Use Case?

- A way to use the system
  - *”A sequence of transactions offered by the system that yield a useful result for an actor“ [Jacobson et al, 1992]
- Externally required functionality
- What a system does, not how it does it
- Specify the behaviour of a use case as a flow of events
  - textually
  - graphically

Task 1: Identify Actors

- Identifying actors is the starting point
  - ”Who is this system supposed to help¿‘
- Finding human actors is relatively straight forward

- Other systems are sometimes less obvious
- Don't assume that current users will be actors
  - think about potiential users
  - i.e. how can the current system be improved
- May wish to record current system actors to help think about potential actors

Actor Description Template

Actor Name	Actor Name
Description	A description of the role this actor plays
Aliases	Other names that may be used in the problem domain to describe this actor
Inherits	The name of any other actor from which behaviour is inherited
Protocol	Only included if this actor represents an external device or system. Receives: Messages that this actor receives from the system. Sends: Messages that this actor sends to the system.

External Systems as Actors

- If an entity interacts with a domain, then it is considered to be an actor
- An actor that is another system or device is considered to be an External System Actor

Task 2: Identify Initial Use Cases

- For each actor note tha activities/tasks they do as part of their role
  - each activity/task should be listed as a verb phrase
- Examine the tasks, or potential tasks, of each actor in turn
  - an initial use case can be identified from each activity/task
- To identify use cases read the specifications from each actor's perspective
- Interview business users as to what they wish the system to do and how they wish to interact with the new system
- Where an activity is common to a group of actors, it's possible to identify an abstract actor to deal with the interaction.

Sources of Use Cases

- Possible sources for use cases
  - functional requirements specification
  - domain material
  - interviews with users
  - personal experience
  - workshops with users
- Identify the logical functioning of the business process and do not include technical details
  - State "what" is required and "why"
  - Not "how"

Tips for Identifying Use Cases

- The following questions are useful starting points:
  - What are the main tasks of each actor?
  - Will the actor have to read/write/change any of the system information?
  - Will the actor have to inform the system about outside changes?
  - Does the actor wish to be informed about unexpected changes?

Task 3: Draw Use Case Diagram

- Once the actors and their use cases have been identified a use case diagram can be drawn
- A use case diagram represents the actors, the boundary of the system and the use cases of the system
- Most useful for presentation and communication purposes
- A Use Case Diagram provides a high-level overview of the system requirements

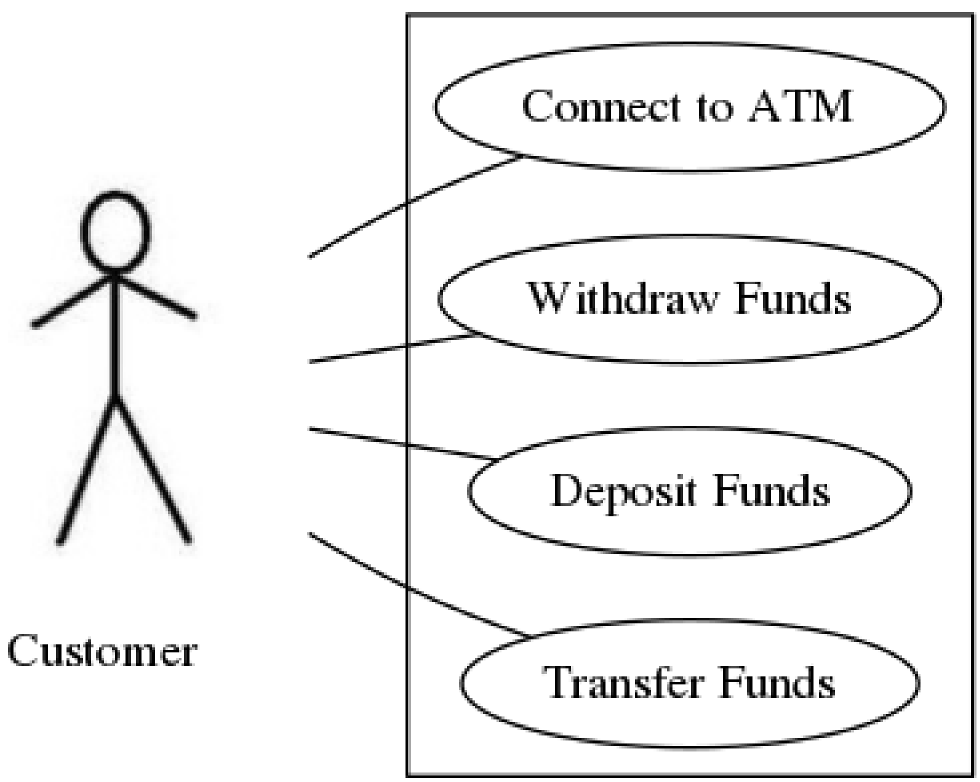

Figure 6: Example of a User Case

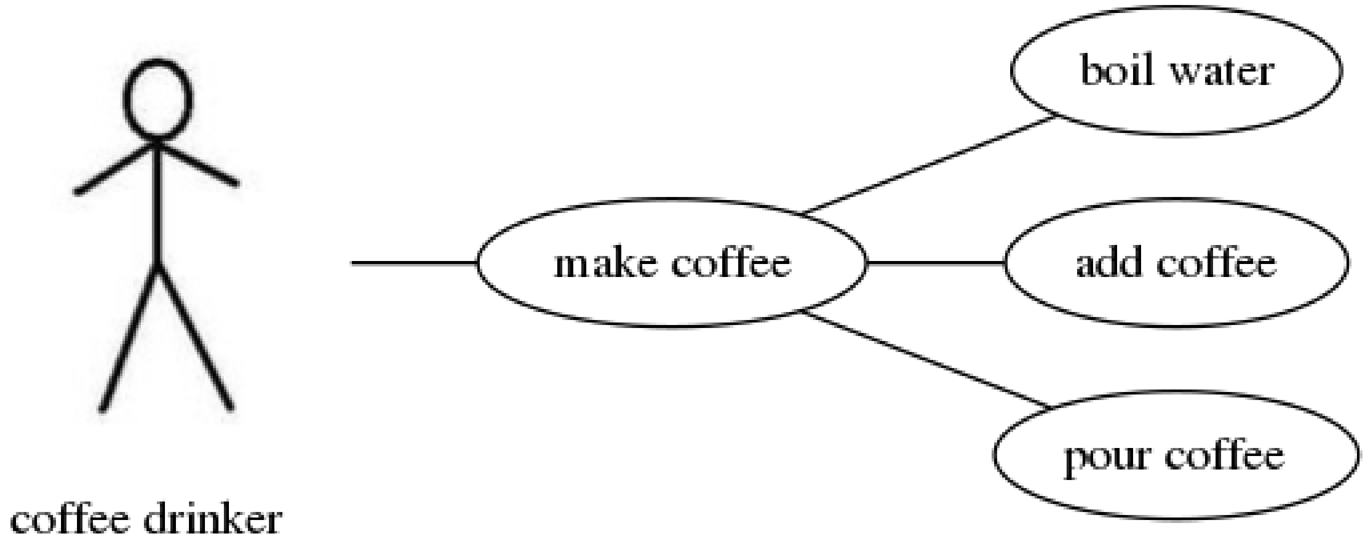

Figure 7: Not a Valid Use Case

Mis-Use Cases

- Associations in use case diagrams connect actors to use cases (they never relate actors to actors or use cases to other use cases)
- Most importantly, they're not designed to show a functional decomposition of the system
  - Figure 7 is an example of not a valid use case

Use Cases are Not a Flowchart

Task 4: Identify Packages

It is useful to group use cases into packages that represent subject areas - A logical aspect of the problem space where a group of use cases work towards a common goal.

Packages – UML Notation

- See Figure 8

Task 5: Develop Initial Use Cases

- Develop in workshop/JAD sessions in conjuncation with client representatives
  - Requirements Modeller acts as a facilitator
  - Prototype Developer may be observer
- Initially keep use cases simple and at a logical level only, describe what needs to be done in the use case
  - Use Cases may vary considerably at this stage as ideas are brainstormed – detailed use cases will be hard to change at this stage

Priorities the Use Cases

- The complete listing of use cases should be prioritised based on organisational needs
- This allows important functions to be delivered first and the system roll-out to be business requirements driven

Identify Typical and Alternative Scenarios

- A use case may be composed of typical and alternative scenarios
  - Typical scenarios – normal sequence of events
    - represent the course taken in 80% of cases
  - Alternative scenarios – abnormal courses of operation

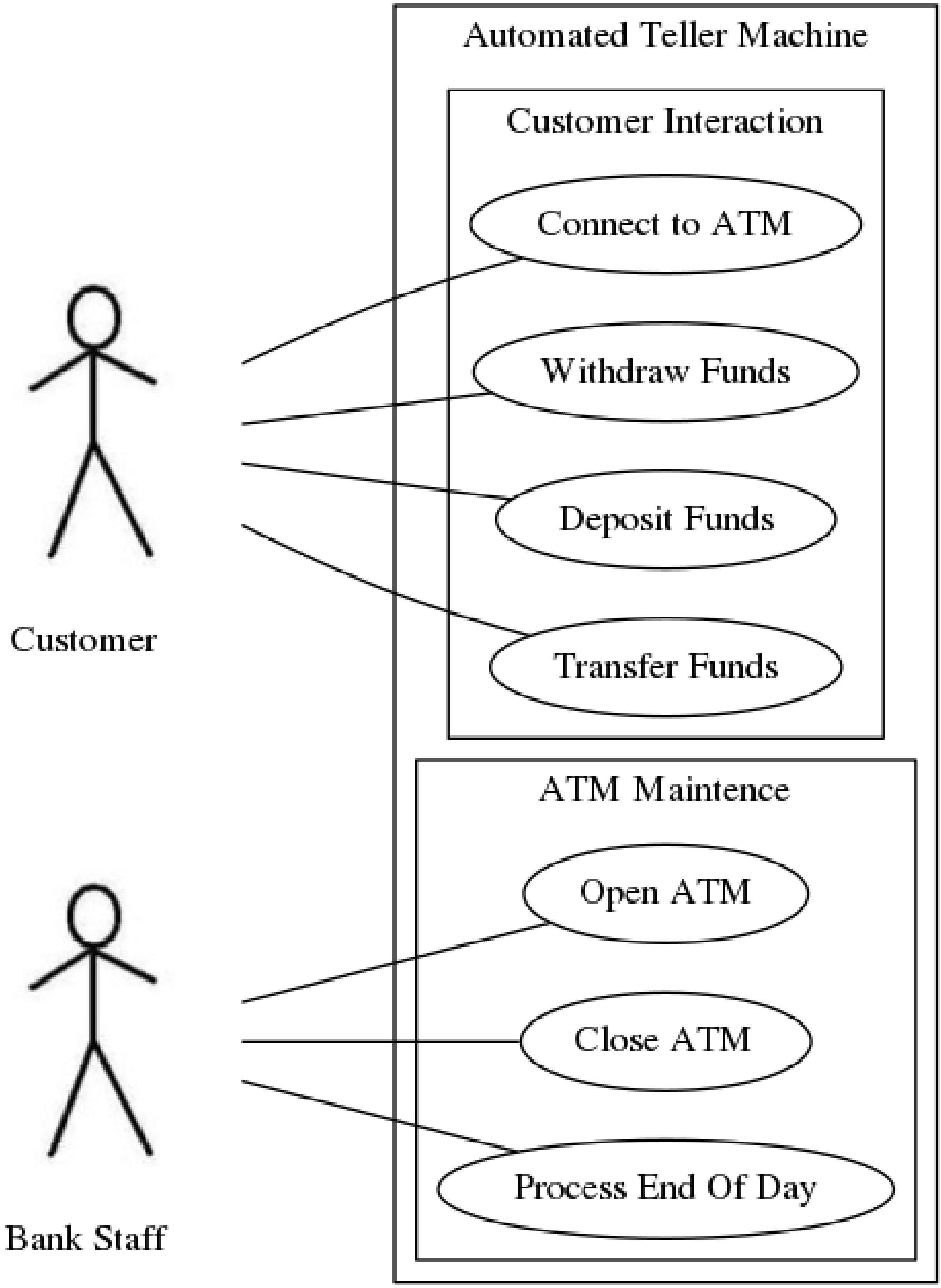

Figure 8: Packages – UML Notation

    * represent errors, or conditions not often encountered
- Try to understand the typical scenario first
  - just list the alteratives

Documenting Use Cases Guidelines

- Initially just specify the events of the use case as a series of steps
- During refinement ensure that a statement of "why" a step is being executed is included
  - It is easy for a domain expert to forget to document why, because this seems obvious
  - Try to remember that the model will be read by people unfamiliar with the domain
- Number the steps
- If steps are completed within the context of another step, indent the text and the numbers
  1) The First Step
     2) This step must be completed to accomplish step 1
     3) This is the next step to accomplish step 1
  2) The Second Step
- When the typical scenario is being documented, identify any alternative scenarios
  - they do not have to be fully documented until the final release point
- Document the steps of a use case sequentially
  - a step the actor performs
  - followed by a step the system does in response
  - next step the actor performs

Description Detail

- Provide enough detail so the Business Owner can verify it performs the actions they require
- Functionality of the system needs to be described in enough detail to allow object decomposition

Use Cases and the User Interface

- Keep use cases general
  - use cases should describe what the actor logically wants to achieve (not the means by which they achieve those goals)
- UI/UX design is a seperate activity
  - after use cases have been validated

Task 6: Refine the Typical and Alternative Scenarios

- Once the typical and alternative scenarios have been identified and agreed they need to be refined
- Prototyping may be used during the refinement process to model the interaction with the system
- Refining use cases involves adding detail
  - May be done through workshops or prototyping (depending upon the complexity of the use case)
- Requires attention to detail
  - good understanding of problem domain and system goal
- Use cases can be developed in parallel
  - different primary actors by different teams
- Use cases can be added to each other to create more complex use cases

Task 7: Restructure Use Cases

- Restructure the use cases to maximise reuse and minimise redundancy
- Use cases can be restructured in three ways
  - <<include>> relationship
  - <<extend>> relationship
  - generalisation
- Functional decomposition
  - identify repeatedly used or specialised portions and factoring them out

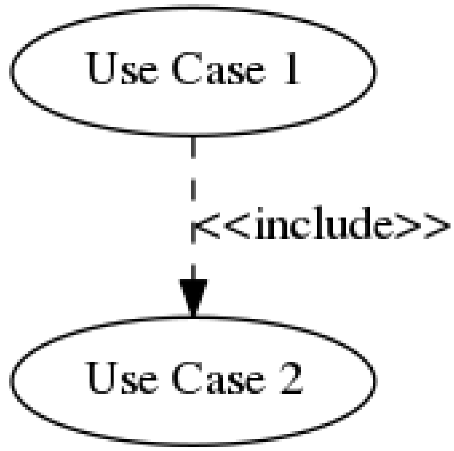

Figure 9: Include UML Notation

<<include>> Relationship

- Factor out common behaviour in use cases
  - sequence of steps appearing in multiple use cases
- Scenario always uses the included steps
  - included steps do not have to be complete use case (i.e. initiated by an actor)

Illustrating the <<include>> Relationship

- In the original use case the phrase <<include>> *<use case name>* is inserted where the included use case begins (steps are removed from the original use case)
- <<include>> relationship can be shown in the use case diagram
  - Use case 1 <<include>> Use case 2
  - See Figure 9

<<include>> Relationship in the Scenario Text

Use Case Title:	Reply to a message
Ref.Number:	ES002
Summary:	Allows the user to reply to a message
Primary Actor:	Mail User
Inherits:	None
Includes:	Compose a Message
Extension Points:	None
Preconditions:	A message is selected
Typical Sequence of Events:	1. Read Message 2. Create a reply – «include» Compose a Message 3. Send the reply

<<extend>> Relationship

- Factors out optional behaviour in use cases
  - used when a scenario produces a different result in certain situations
    - when there are optional or uncommon steps that may occur
- The extended scenario may be completed without including the steps from the extending use case

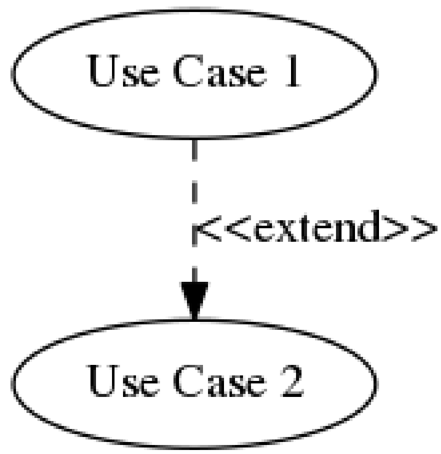

Figure 10: Extend UML Notation 1

  - removes the need to have many primary scenarios to capture the optional paths of a use case
- Delivery of extending use cases can occur in later phases of development

Extension Point
- Included in the use case description to indicate the point at which the extending use case begins
  - condition that causes the extension is highlighted
- Scenario for the base use case runs as normal until the extension point
- Under certain conditions the extending use case then begins and runs to completion
- Base use case then resumes

Illustrating the <<extend>> Relationship
- Shown in two different ways
  - See Figure 10, 11

Use Case Generalisation
- General behaviour can be factored out using generalisation
  - like inheritance in class design
- Use when the behaviour of the child use case is more specific than that described by the parent use case
  - general – specialized relationship

Illustrating Use Case Generalisation
- In the original use case the phrase inherits: <parent use case> is included in the text
- Scenario description of the child indicates the behaviour of the parent that is replaced or refined
  - refine <step number in parent use case>
  - replace <step number in parent use case>
- Generalisation relationship can be shown in the use case diagram
  - See Figure 12 (Use Case 2 inherits Use Case 1)

Naming Guidelines

Actors
- Use a noun for an actor name
- Begin each word in the name with a capital letter

Use Cases
- Use a verb phrase. Not a noun phrase. Use cases describe an activity
- Name from the primary actor's perspective

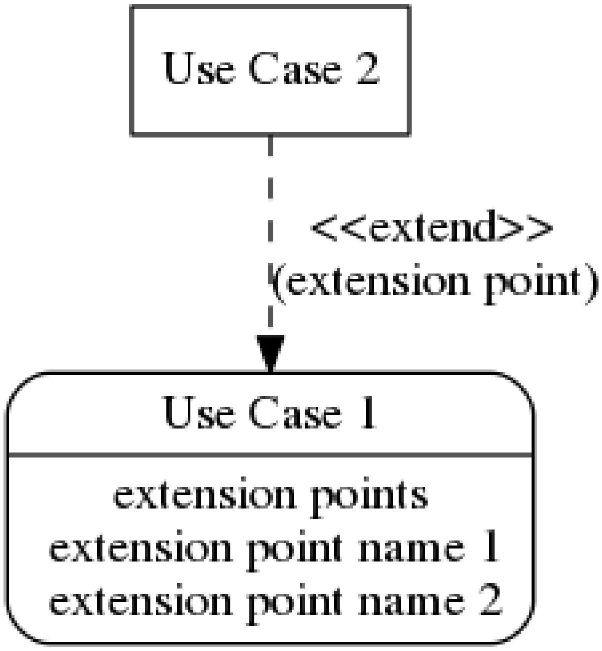

Figure 11: Extend UML Notation 2

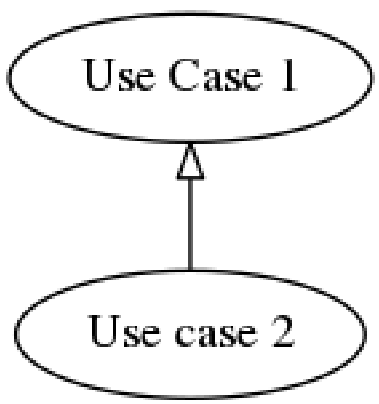

Figure 12: Generalisation UML Notation

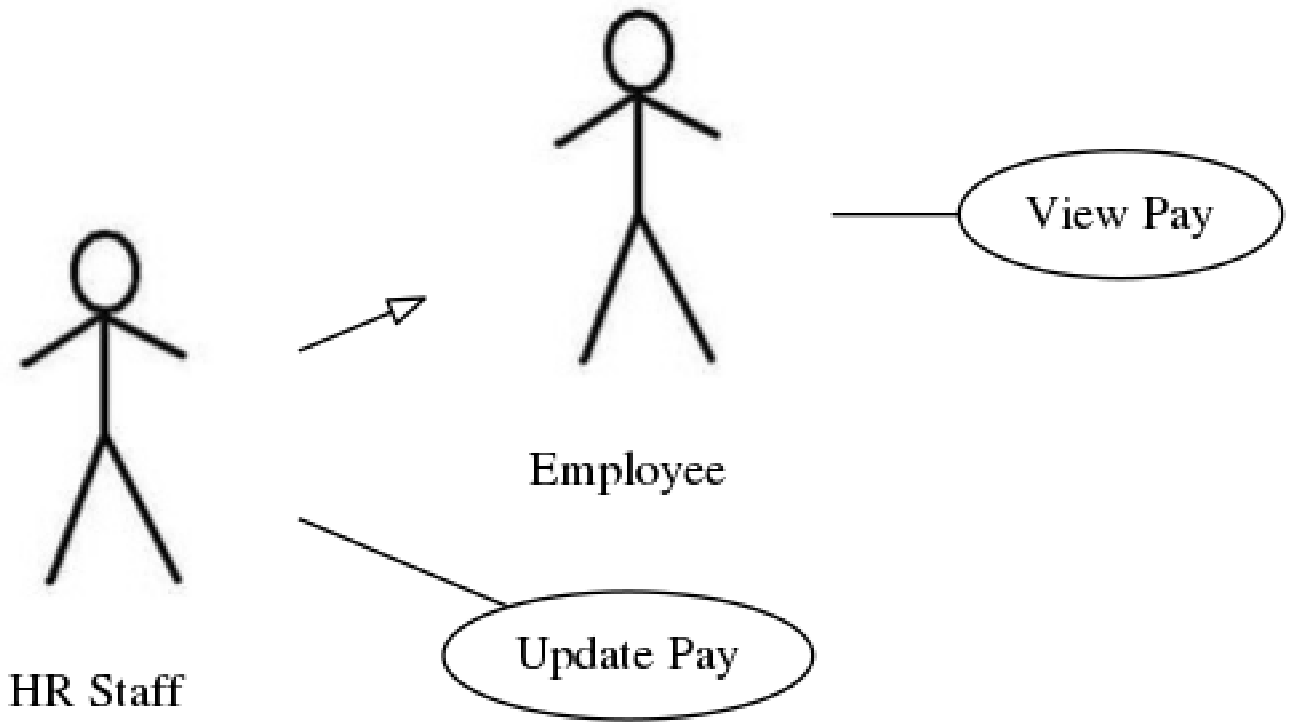

Figure 13: Abstract Actors Example

General (Abstract) Actors

- Actors that share use cases could be abstracted such that a new general actor is created (don't go overboard)
- Ensure the general and specialised actors both represent useful concepts or users *See Figure 13*

Use Cases Limitations

- Based on usage scenario
- Not suitable for
  - batch processing
  - computationally intensive (e.g. business analytics)
  - embedded systems

Use Cases vs User Stories

- Both are user-centred
  - what is to be accomplished
- User Stories
  - statement of user needs
    - details to be discovered during development (fleshed out by acceptance tests)
- Use Cases
  - description of interaction with system
    - allows analysis
    - verification

Activity Diagram

- Shows the steps involved in performing a task
  - can be considered an algorithm

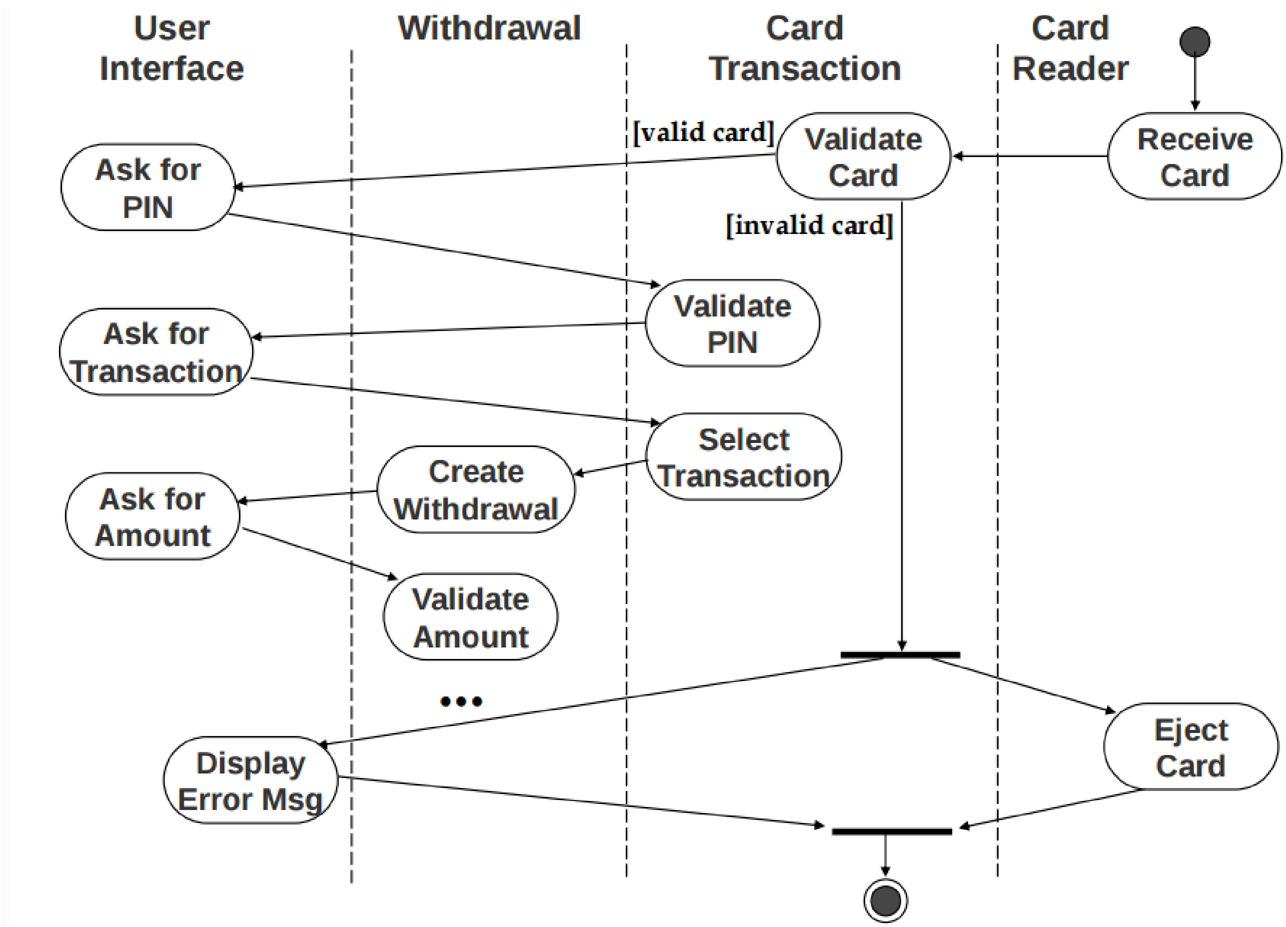

Figure 14: Activity Diagram

- Special form of state diagram
  - all states are action states
  - all transitions are triggered by completion of actions
- Describes a complex task for objects of a class, operations, or use cases.
  - focuses on internal processing
  - use where all the events represent the completion of internally generated actions (procedural flow of control)
  - use state diagrams where asynchronous events occur

See Figure 14

Requirements Modelling

- Product-Centered
  - Focus on features to be delivered
    * expect users will use features to complete tasks
- User-Centred
  - Focus on anticipated usage
    * what do users need to accomplish
  - Reveal necessary functionality
  - Asists with prioritisation

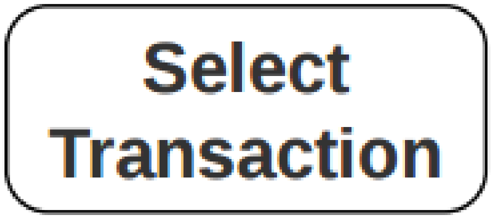

Figure 15: Simple Action

Figure 16: Invoke Activity

Activity Diagram

- Models dynamic behaviour
  - process workflows
  - use case scenarios
  - operations
- Steps involved in performing a task
  - like an algorithm

Activity Diagram and Use Cases

- Model flow of control in scenarios
  - all scenarios can be shown on one diagram, if required
- Flow-chart like representation of a use case
  - can be useful for workshop reviews of scenarios

Activity Diagram and BPM

- Model activities carried out by actors in business domain
- Can establish the business context
  - before use cases are extracted from the processes
- Illustrates interactions between actors in a process
  - using partitions (swimlanes)

Activities

Simple Action

Lowest level of detail shown in diagram. See Figure 15

Invoke Activity

Includes a number of steps shown in another diagram. See Figure **??**

Time Event

Triggered by some time related condition (e.g. date, time, period). See Figure 17

Nodes

Initial Node

One per diagram. See Figure 18

Figure 17: Time Event

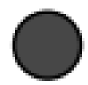

Figure 18: Initial Node

Final Node

Optional and more than one allowed (all flows stop when reached). See Figure 19

Flow Final Node

Flow stops, others continue. See Figure 20

Decision Node

One in, multiple out. See Figure 21

Merge Node

Multiple in, one out. See Figure 22

Parallel Flows

Can have concurrent process flows

Fork Node

Start concurrent flows. See Figure 23

Join Node

Joins concurrent flows. See Figure 24

Example

See Figure 25, 26

Activity Diagrams in Use Case Modelling

- Determine triggering event that starts use case flow
- Identify actions and determine control flow
- Add guard conditions and decision points
- Add forking and joining to show parallel activity
- Create invoke activities if complexity requires it
- Group activities into partitions if needed

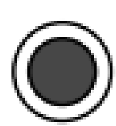

Figure 19: Final Node

Figure 20: Flow Final Node

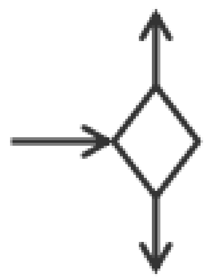

Figure 21: Decision Node

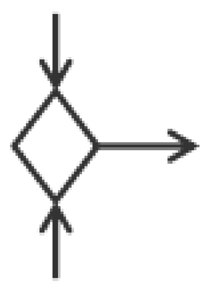

Figure 22: Merge Node

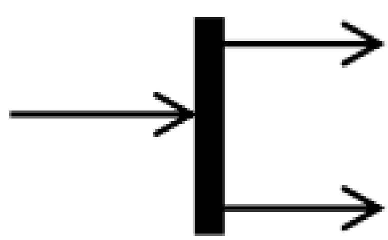

Figure 23: Fork Node

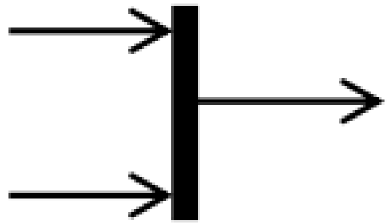

Figure 24: Join Node

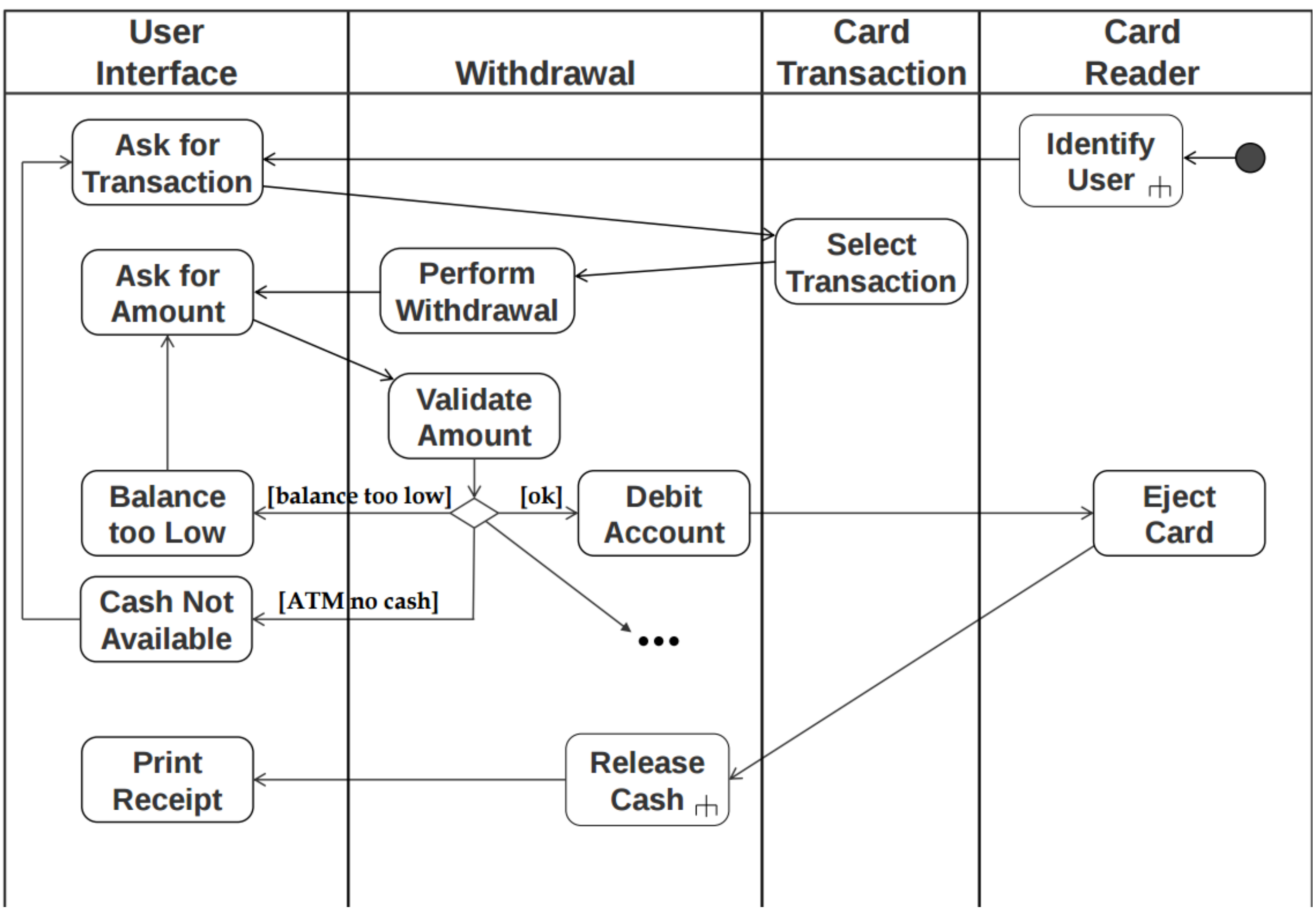

Figure 25: Example Activity Diagram 1

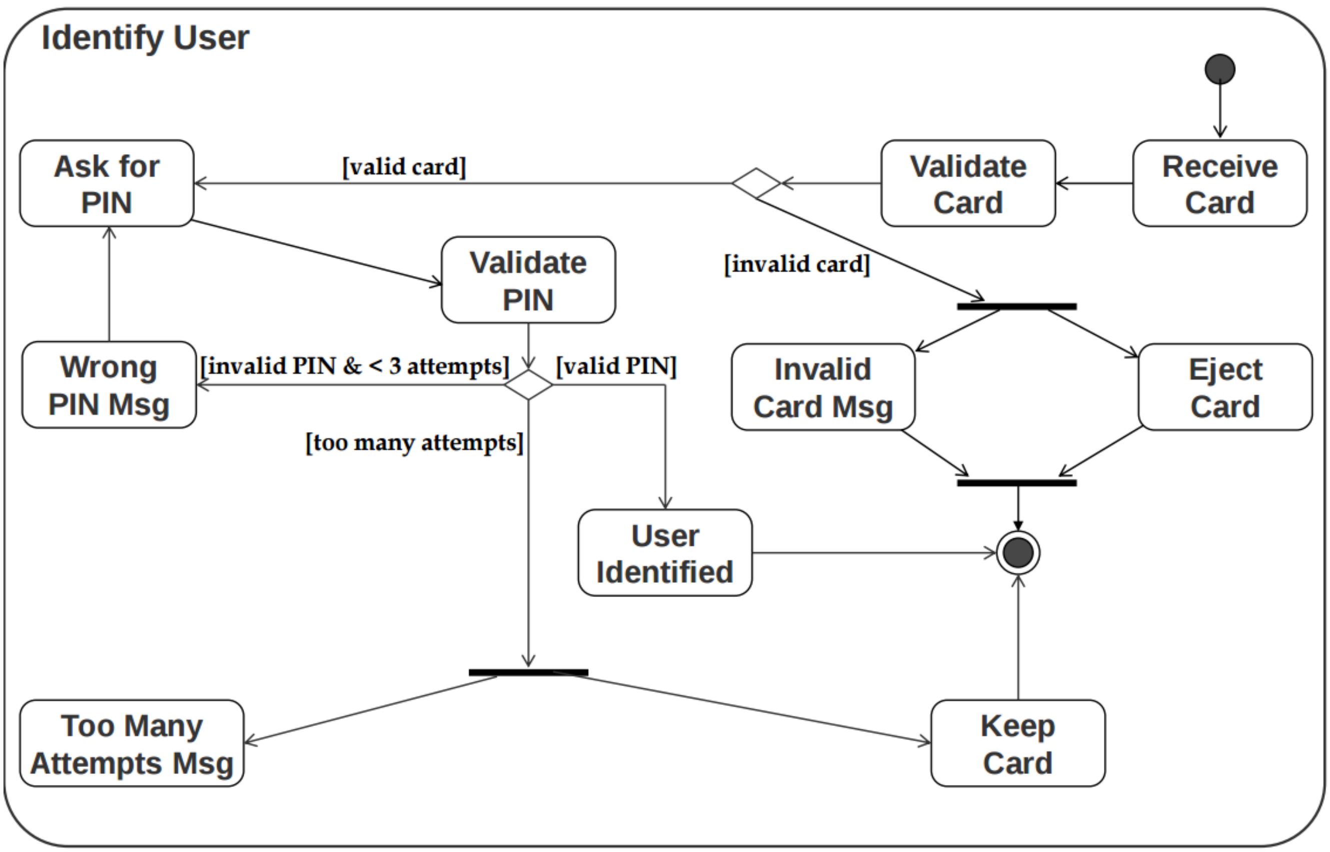

Figure 26: Example Activity Diagram 2

- Add flows corresponding to alternative scenarios
- Each path should correspond to an individual scenario