designing the user experience

upa

usability professionals' association

www.upassoc.org

SE463
Software Requirements Specification & Analysis

Functional Requirements

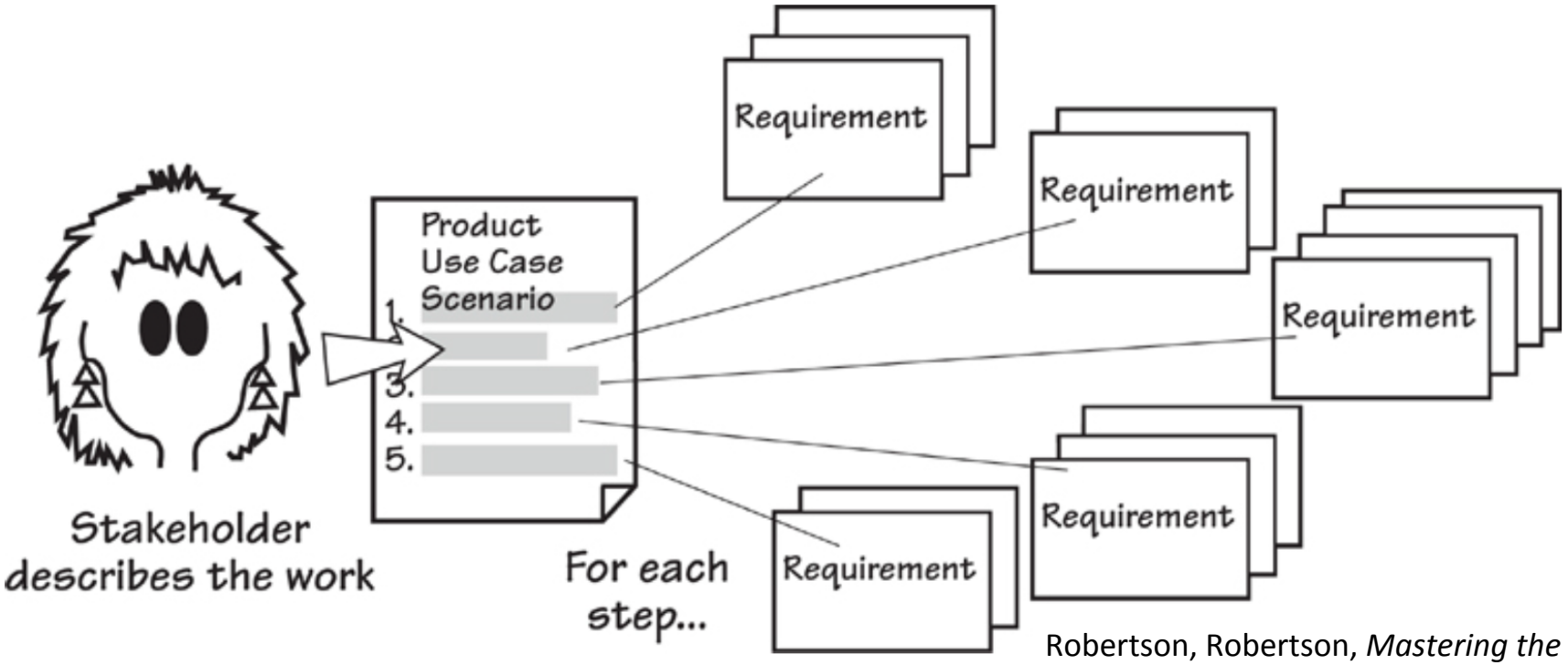
Functional Requirements

Functional requirements describe what the product has to do to support and enable the Work.

They should be, as far as possible, independent of the technology used by the eventual product.

- functional requirements
- EARS (Easy Approach to Requirements Syntax)
- user stories
- graphical function models

Atomic Requirements

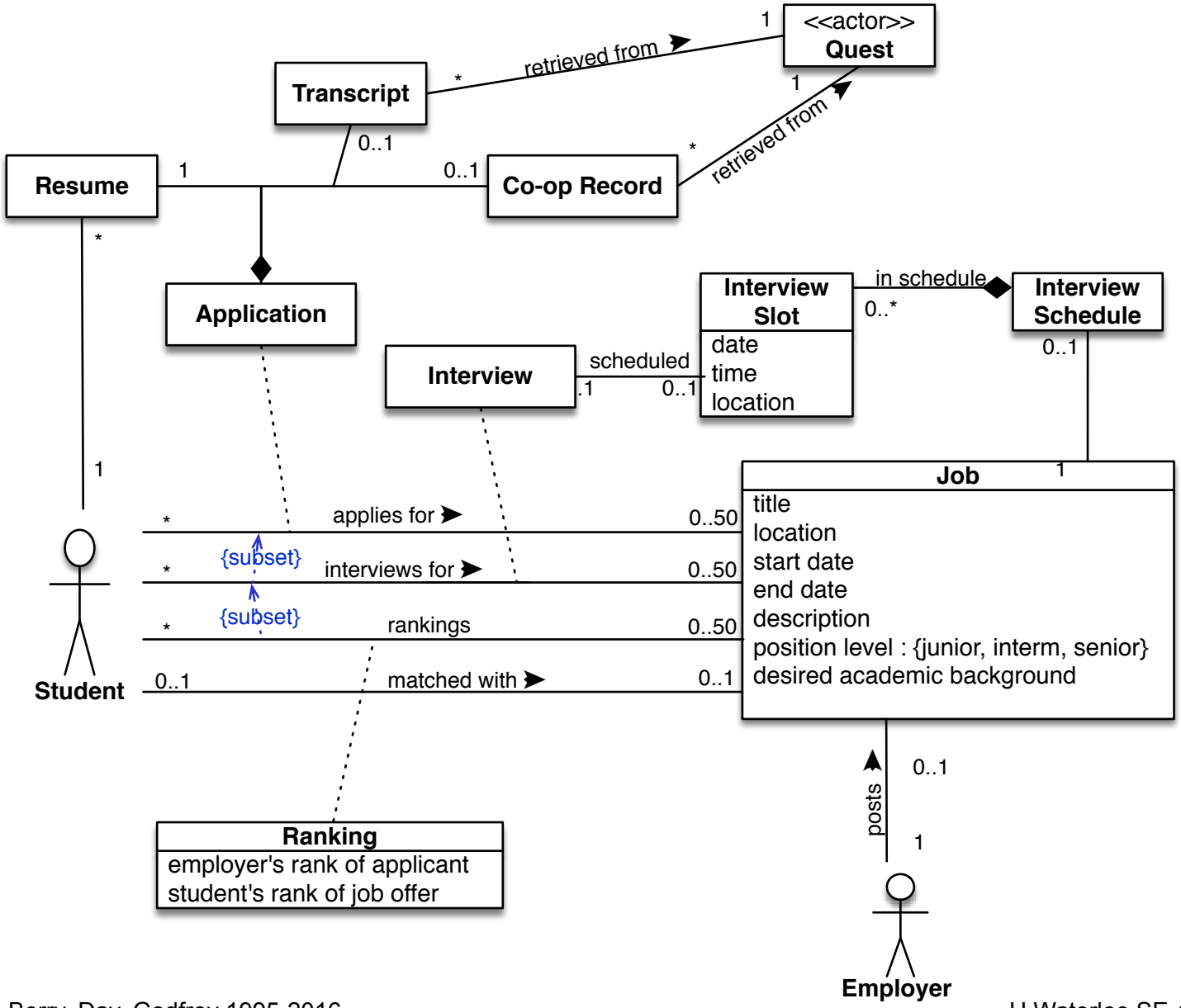


Robertson, Robertson, *Mastering the Requirements Process*, 2012, Figure 10.2

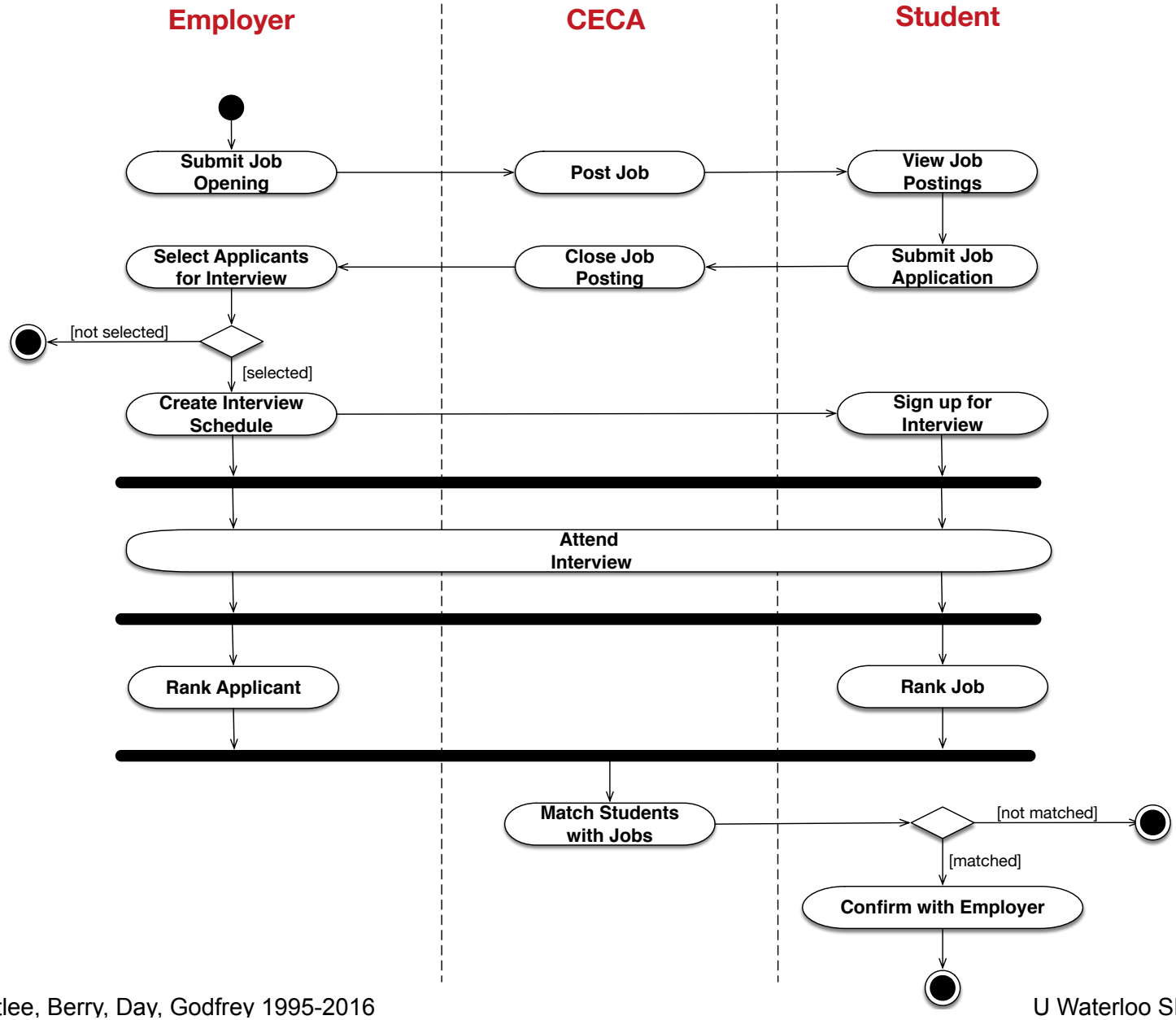
Atomic (functional) requirements are derived from scenarios or activity diagrams

- What the product has to do to achieve each step/activity
- Expressed using vocabulary from the domain model

JobMine Domain Model



JobMine Activity Diagram



Level of Detail or Granularity

Atomic requirements are written as a single sentence with a single verb.

> *Product will require a ranking for each job where the student user has been ranked by the employer*

Do *not* use a mixture of modal verbs (shall, must, might, could) to indicate the priority of a requirement.

- Use a single modal verb consistently, and indicate priority as an attribute of the requirement.

Alternatives and Exceptions

Include atomic (functional) requirements that are derived from scenario alternatives and exceptions.

These requirements are conditional on the event / condition that leads to the alternative or exception.

The product will display the set of available interview slots and allows the student user to select one of them.

- *If the student user neglects to select an interview time, then two days before the interview date the product will automatically schedule the student into one of the remaining interview slots.*

The product will confirm receipt of a student user's submitted job application.

- *If submission of a job application fails, the product will warn the user that the submission failed.*

Rationale

Each atomic requirement should be accompanied by a rationale that explains why the requirement exists.

If there are multiple matches whose sums of employer and student rankings are all the lowest sum, the product will select among these matches at random

Rationale: To ensure that job matches are not biased toward employer or student rankings.

The product will record the start and end times of a student user's session when viewing job applications.

Rationale: CECA wants to know which users use the product the most.

Rationale: Viewing job applications should take 90% of students less than 6 hours a week.

Requirements Template

Requirement #: 75 Requirement Type: 9 Event/BUC/PUC #: 7, 9

Description: The product shall record all the roads that have been treated

Rationale: To be able to schedule untreated roads and highlight potential

Originator: Arnold Snow - Chief Engineer

Fit Criterion: The recorded treated roads shall agree with the drivers' road treatment logs and shall be up to date within 30 minutes of the completion of the road's treatment

Customer Satisfaction: 3 Customer Dissatisfaction: 5

Dependencies: All requirements using road and scheduling data Conflicts: 105

Supporting Materials: Work context diagram, terms definitions in section 5

History: Created February 29,2010

Volere

Robertson, Robertson, *Mastering the Requirements Process*, 2012, Appendix A, Figure A.10

EARS Template

Easy Approach to Requirements Syntax (EARS)

A template for structuring the syntax of the one-sentence description of an atomic requirement

- acknowledges that the vast majority of requirements will be expressed in natural language
- promotes a gently constrained use of natural language

EARS Generic Syntax

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

<optional precondition> <optional trigger> the <system name> shall <system response>

The generic requirement syntax is specialized into five types of requirement

EARS Ubiquitous Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

A ubiquitous requirement is an invariant behaviour of the system (that should always hold).

The <system name> shall <system response>

Example:

"The control system shall prevent engine overspeed."

EARS Event-Driven Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An event-driven requirement applies on when a triggering event is detected at the system boundary.

WHEN <trigger> the <system name> shall <system response>

Example:

"When commanded by the aircraft, the control system shall switch on continuous ignition"

EARS State-Driven Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An state-driven requirement applies as long as the system is in a particular state, or mode of operation.

WHILE <in a specific state> the <system name> shall <system response>

Example:

"While the aircraft is in-flight, the control system shall maintain engine fuel flow above XXlbs/sec"

EARS Option Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

An option requirement when system behaviour is dependent on the presence of a particular feature.

WHERE <feature is included> the <system name> shall <system response>

Example:

"Where the control system includes an overspeed protection function, the control system shall test the availability of the overspeed protection function prior to aircraft dispatch."

EARS Unwanted Events

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

Special syntax distinguishes required system responses to unwanted events (e.g., failures, disturbances, deviations from desired user behaviour or any unexpected behaviour of interacting systems).

IF <trigger>, THEN the <system name> shall <system response>

Example:

"If the computed airspeed fault flag is set, then the control system shall use modelled airspeed."

EARS Combination Requirements

Mavin, Wilkinson, Warwood, Novak EARS (Easy Approach to Requirements Syntax), RE'09

Examples:

"While the aircraft is on-ground, when reverse thrust is commanded, the control system shall enable deployment of the thrust reverser".

"When selecting idle setting, if aircraft data is unavailable, then the control system shall select Approach Idle"

Real Examples

Rolls Royce Control Systems	Re-expression in EARS
It must be demonstrated that, when a Fault or Failure results in a change from one Control Mode to another, or from one channel to another, or from the Primary System to the Back-Up System, the change occurs so that the Engine does not exceed any of its operating limitations.	When the Engine Control System changes operational mode, the Engine Control System shall maintain the engine within the approved operation limits.
Single Failures leading to loss, interruption or corruption of Aircraft-Supplied Data must not result in a Hazardous Engine Effect for any Engine.	If a single Failure leads to deficient Aircraft-Supplied Data, then the Engine Control System shall not cause a Hazardous Engine Effect.

User Stories

User stories provide a light-weight approach to managing requirements:

- Short statement of new functionality or feature
- Written from the point of view of the user
- Its details are fleshed out just in time before the story is placed in the sprint.

Three C's of User Stories

- **Card** - stories are traditionally written on note cards, using a structured syntax:

  As a <role>, I want <something>, so that <benefit is achieved>

> *As a vacation traveller, I want to see photos of hotels, so that I can get a sense of the quality of the hotel*

> *As a user, I want to cancel a reservation, so that my credit card is not charged*

- **Conversation** – discussions with the product owner reveal details of the requirements

- **Confirmations** – acceptance criteria for objectively determining whether an implementation meets the requirements.

Details as conditions of satisfaction

As a user, I can cancel a reservation.

- The product owner's conditions of satisfaction can be added to a story
- These are essentially tests

- ☐ Verify that a premium member can cancel the same day without a fee.
- ☐ Verify that a non-premium member is charged 10% for a same-day cancellation.
- ☐ Verify that an email confirmation is sent.
- ☐ Verify that the hotel is notified of any cancellation.

Details added in smaller sub-stories

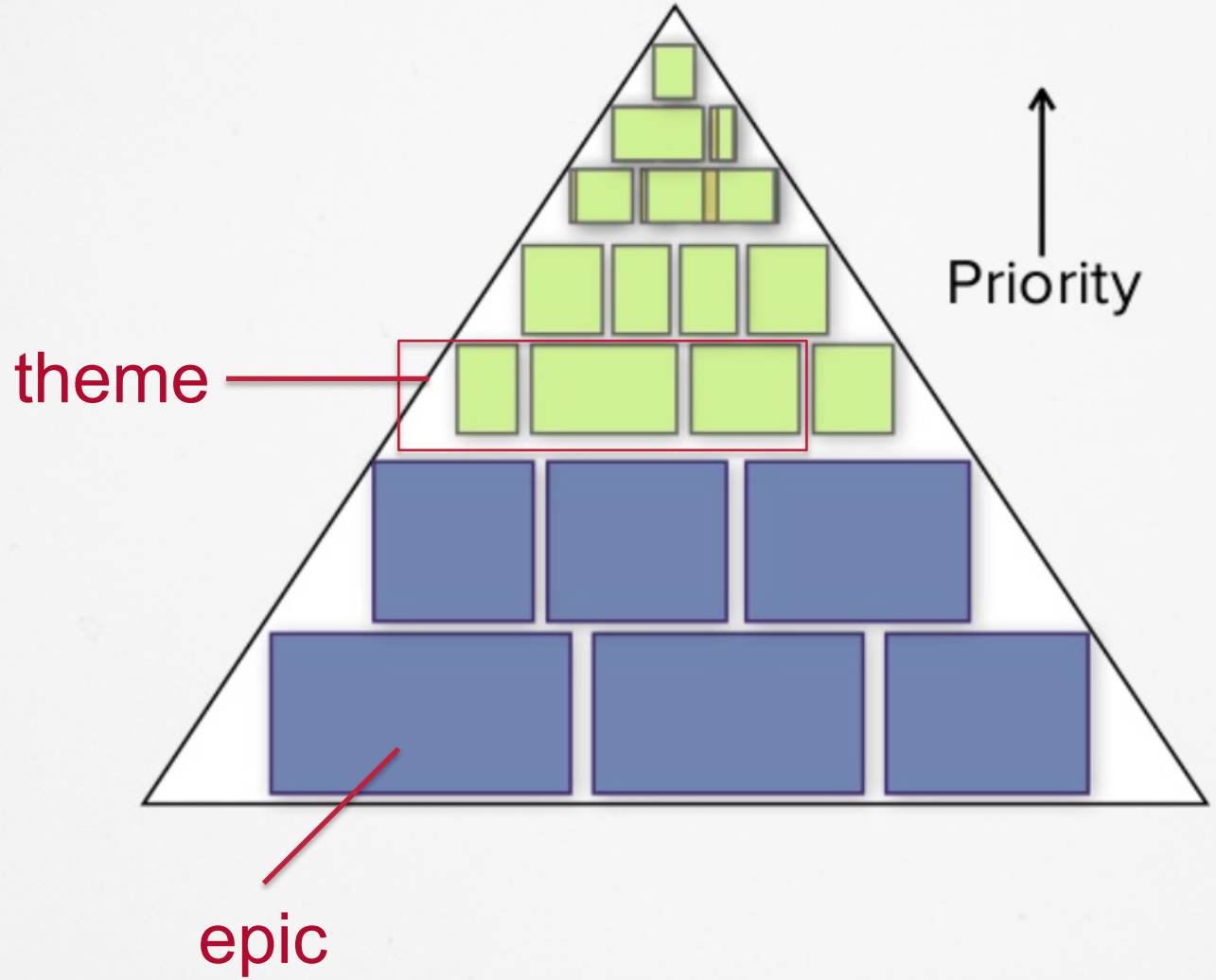
As a user, I can cancel a reservation.

As a premium site member, I can cancel a reservation up to the last minute.

As a non-premium member, I can cancel up to 24 hours in advance.

As a site visitor, I am emailed a confirmation of any cancelled reservation.

The product backlog iceberg



Why User Stories?

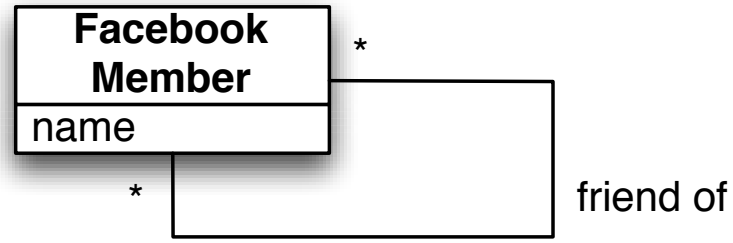
- Easy for stakeholders to understand, and to remember.
- Shift the focus from written documentation to discussion.
- Encourage iterative development, with stories being appropriate sized increments for planning
- Delay the elicitation of requirements details until just before development.
- Support participatory elicitation

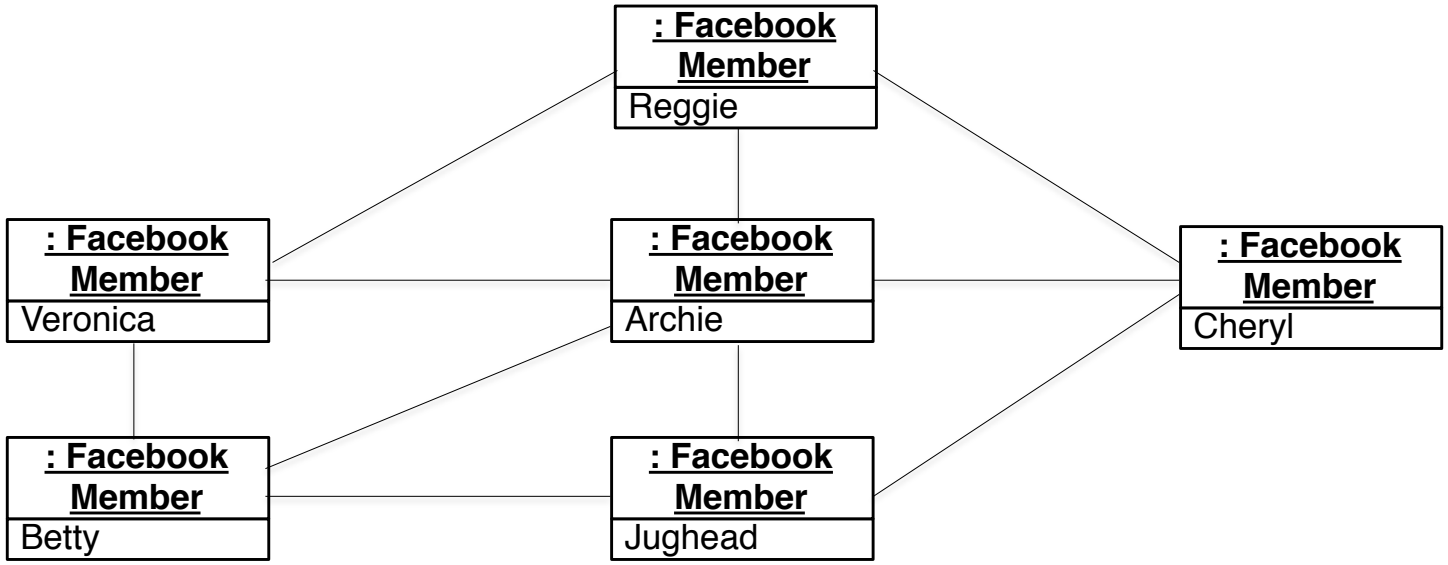
Deliverable #5

- EARS requirements or User Stories
  - For all atomic requirements needed to implement the steps of all of your scenarios in Deliverable #4
  - Including the atomic requirements needed to implement the steps in your scenarios' alternative and exception paths
  - Cite the source of each atomic requirement and the elicitation technique that was used
  - Your atomic requirements should
    - reflect the needs of at least three (3) stakeholders
    - elicited from at least two (2) concrete stakeholders
    - using at least four (4) elicitation techniques

Function as a Change to the World State

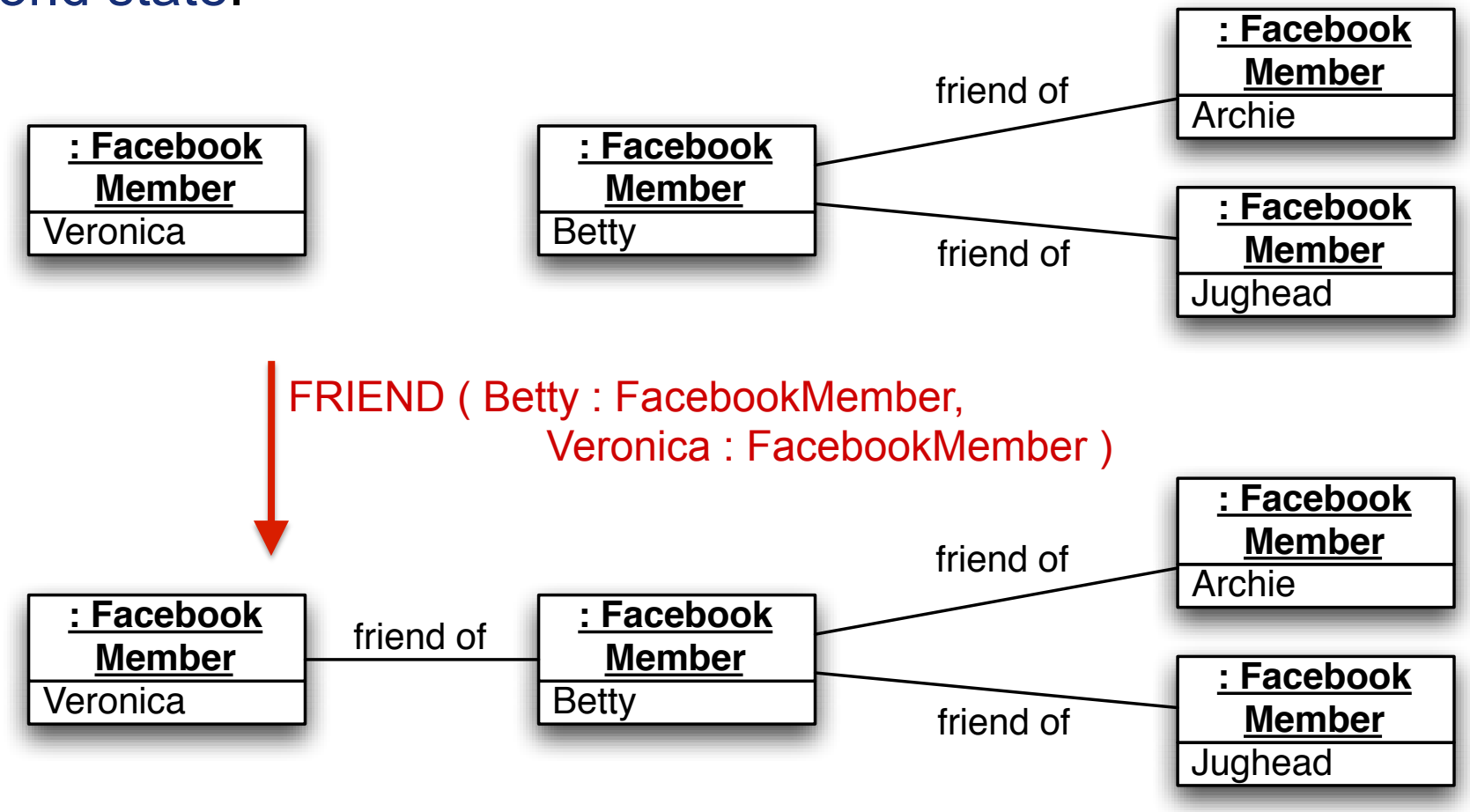
The domain model represents the set of possible states of the world (called world states)



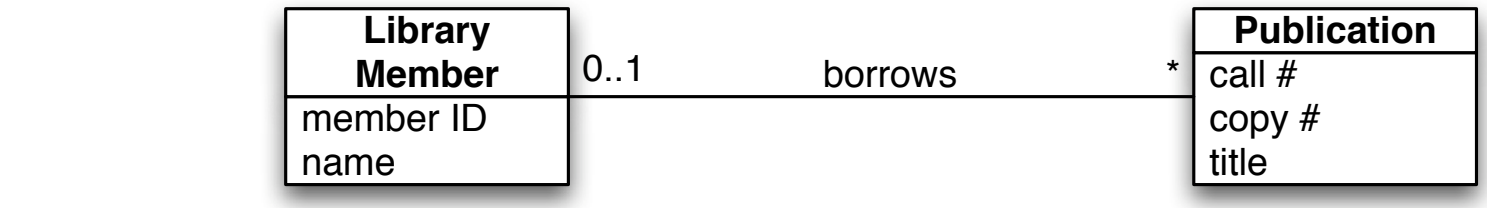


Graphical Function Model

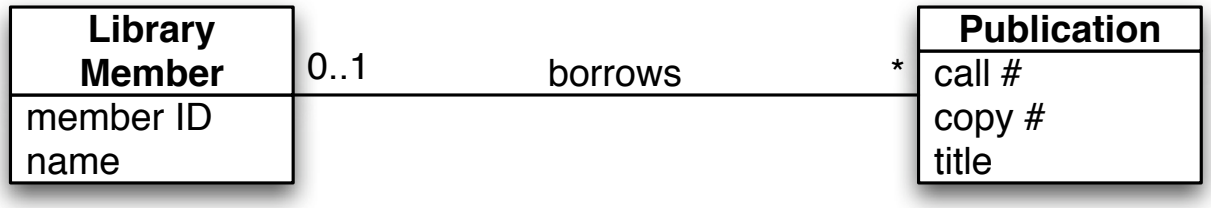
Graphical Function Model – expresses a system-level function as a rule that matches and replaces patterns in the world state.



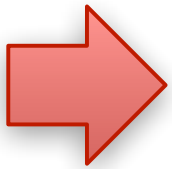
Graphical Functional Model



Another Example



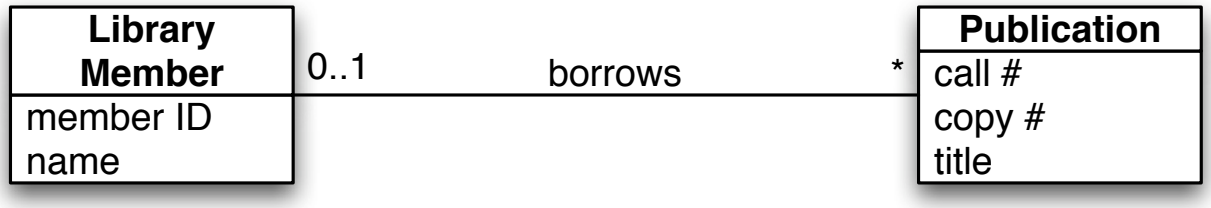
AddMember (name)



: Library Member
memberID = unique ID
name

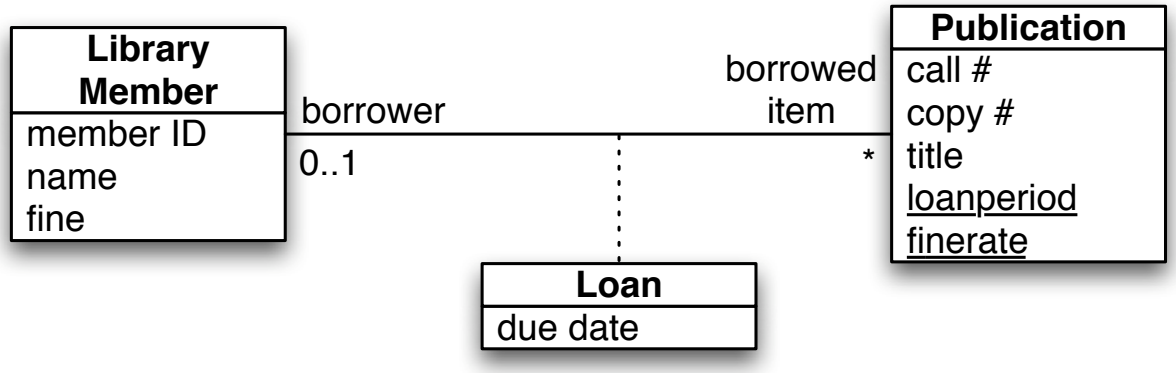
AddMember (name)

Another Example



DeleteMember(memberID)

Another Example



BorrowItem (m:LibraryMember; p:Publication; today)

Summary

Express atomic requirements as system-level functions.

- functional requirements
- EARS (Easy Approach to Requirements Syntax)
- user stories
- graphical function models