Object-Oriented Analysis and Design

Session 1: Introduction

Course Goal

The course focus on the **design** of software systems

– Design means how the system will be built and operate
– Modeling as a means to express design decisions
– Design Patterns serve as best practice guidelines for coding
– Refactoring provides methods for improving the code

Outline

- Software System Characteristics
- Development Processes
- Modeling Languages
- The Unified Modeling Language

Software System Characteristics

What is a software system?

- Computer programs and associated documentation:
  - Requirements;
  - Analysis and Design models;
  - User manuals;
- Software products may be
  - **Generic** - developed to be sold to a range of different customers:
    - PC software such as *Excel* or *Word*.
  - **Bespoke** (custom) - developed for a single customer according to their specification.

A Case Study – The Library System

- Computerize the university library.
- Build it as an interactive online system.
  - Books and journals
    - Several copies of a given book.
    - Short-term loans.
    - Long-term week loans.
    - Only staff members can borrow journals.
    - An individual can borrow up to 6 books at a time.
    - Staff can borrow up to 12 books at a time.
  - Searching
  - Borrowing
    - Track when books are borrowed and returned.
    - Issue reminders.
    - Possible future requirement for loan-extension functionality.

Characterization of a Good System (1)

- Useful and usable
  - The library may be operated by less people.
  - The students will be able to order books.
- Reliable
  - The system should mark the right ordered book for a specific student.
- Flexible (modifiable) and easy maintenance
  - It would be easy to add new requirements such as dealing with CDs.
- Affordable
  - The university has allocated the right budget to develop the system.
- Available
  - The system should work 7 days a week.

Characterization of a Good System (2)

Difficulties:

- Our understanding of systems is usually partial:
  - It is not trivial to understand the tracking mechanism.
- We tend to misinterpret casual properties as characteristic ones:
  - A borrower account number is not essential to the system.
- We tend to solve problems locally:
  - Developing an additional search mechanism within the order processing.
- We tend to solve whole problems. Small changes in specifications lead to reconstruction of the whole system:
  - The system is planned to support only books and journals. Addition of CDs implies changes to the design (and code).

A Key for Building a Good System: Modularity

A **modular system** consists of :

encapsulated modules

with

dependencies among them.

Encapsulation and Abstraction

Encapsulation:

When a client **is not able** to know more than is in the Interface.

Abstraction:

When a client **does not need** to know more than is in the interface – declared services

For example:

Core Module – List

and its Client – Queue

Abstraction/encapsulation are essential for modularity

A measure for abstraction: Cohesion

Cohesion is the amount of coherency of a module – a module needs a main theme:

Theme 1 - Borrowing

Theme 2 - Searching

High cohesion is a major **pattern** (advice) in assigning **responsibilities** to modules

Good abstraction implies **reuse → Pluggable components**.

What is Dependency?

Module *A* **depends** on module *B* if changes to module *B* can imply changes to module *A*:

A is a **client** of *B* – *A* use services of *B*;

B acts as a **server** to *A*

Ordering depends on Searching

- Important information for a module:
  - Which modules are its clients?
  - What assumptions clients make about it?
  - Which modules are its servers?

A measure for dependency: Coupling

Coupling is the amount of dependency among modules.

A system with many dependencies has
high coupling.
Good systems have
low coupling.

Low coupling is a major **pattern** (advice) in assigning **responsibilities** to modules.

Achieving Modularity

- Interface is the means for achieving modularity:
  - Interfaces → encapsulated modules
- Interface of a module**:**
  - ***Encapsulates*** knowledge about a module.
  - Defines ***features*** on which clients can rely.
- Contract of a module
  - Declares the responsibilities of a module (interface, services, constraints).
  - States the context dependencies of a module -- the services it requires in order to work.

Layering

- A common technique for breaking apart complex software systems.
  - Machine architectures:

    Programming languages + Operating systems.

    Device drivers + CPU instructions.

    Logic gates inside chips.
  - Networking: FTP → TCP → IP → Ethernet.

- Principles:
  - Higher layers use services of lower layers.
  - Lower layers are unaware of their higher layers.
  - Each layer hides its lower layers from its super layers.

Pros. and Cons. of Layering

- Pros.
  - Understand one layer independently of other layers:
    - Abstraction
  - Replace layers with alternative implementations.
  - Minimize dependencies between layers.
  - A layer can be used for multiple purposes in higher layers
    - Information systems
- Cons.
  - Reduce efficiency: Translation takes time.
  - Changes in a higher layer propagate downwards.
    - Addition of a field in a user interface propagates down to the database layer – instead of a direct database schema revision.

The Three Principal Layers

Layers	**Responsibilities**
Presentation	Display information –windows, HTML. Handle user/communication requests – mouse, keyboard, command line.
Domain Logic (business logic)	Application core
Data source	Persistent data storage (Data base, files)

Splitting Information among Layers (1)

Example: *"Find the award winning staff member – serves maximal number of book loans".*

Bad splitting: *1. The UI layer access the database for all staff members and the number of book loans they each handled.*

(no splitting)

2. It then finds the maximal number of book loans.

3. Finally, it finds the corresponding staff member and singles it with a ***red*** color.

Drawbacks:

- *Change in the awarding policy implies changes in the UI layer.*
- *If the awarding policy is duplicated (e.g., for communication with the Human Resource department) than policy change implies multiple revisions (consistency problems).*
- *Change in the singling-out technique implies system changes.*

Splitting Information among Layers (2)

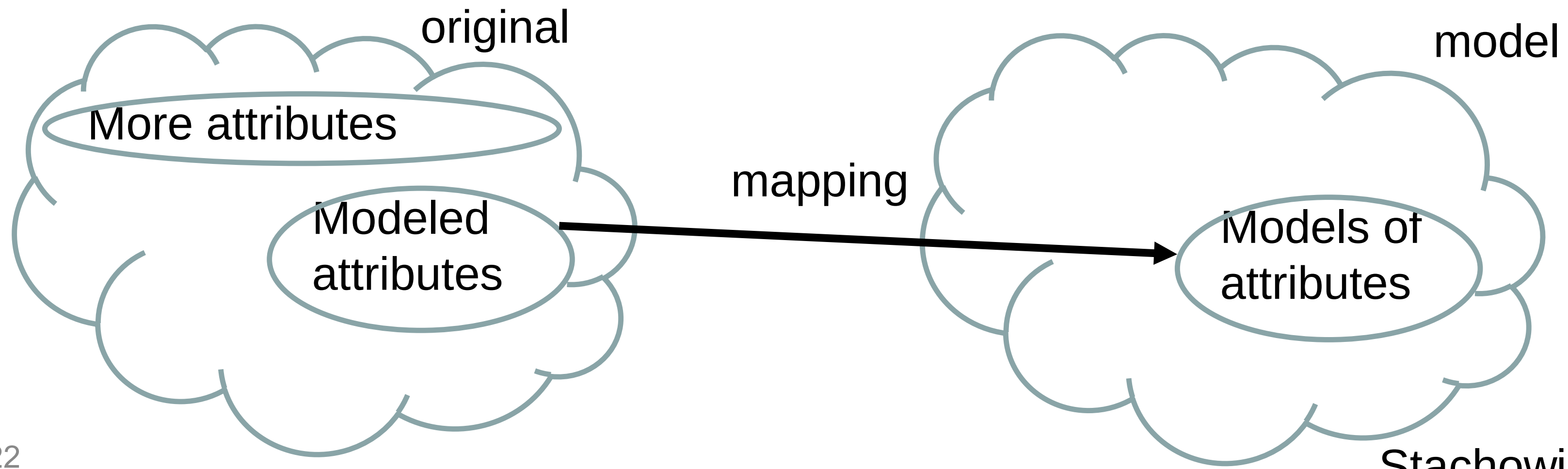
- Good splitting:
  - UI Layer asks Domain Layer for the winning staff member.
  - Domain Layer asks Data Layer for all staff members and the number of book loans they handled.
  - Domain Layer determines the award winning staff member according to the awarding policy.
  - Domain Layer asks the UI Layer to single out the award winning staff member.
  - UI Layer singles out the award winning staff member with a **red** color.
- Advantages**:**
  - Changing the awarding policy affects only the domain layer in a single point (3).
  - Change in the singling-out technique (5) does not affect the rest of the system.

Development Processes

Modeling Languages (also called “Models”)

What is a model?

- **Mapping criterion:** there is an original phenomenon that is mapped to the model.
- **Reduction criterion:** the model reflects only a (relevant) selection of the original's properties.
- **Pragmatic criterion:** the model is usable in place of the original, for some purpose.



Stachowiak [Sta73]

What is a software model?

A simplification of the real problem:

- An abstraction of software features
- A structure that singles out selected features
- Independent from other software elements
- Independent from marginal software details
- May be partial
- Usually visual (diagrammatic)

Why modeling?

- Business level and partiality enable
  - Early structuring **before software construction**:
    - validation of intentions
    - verification
    - Early detection of mistakes
- Requirement engineering
  - Elaboration of intended goals
- Specification (Analysis/Design) of
  - Restrictions
  - Priorities
  - Essential services
  - Structure
- Platform independence enables
  - Reuse
- Implementation
- Testing

The Role of Models

- Communication
  - Abstraction
- Specification
- Implementation
- Documentation

Model-Based software engineering

- Focus on ***models*** – rather than on ***programs***
- Model automation:
  - Analysis
  - Verification
  - Validation: testing
- Gains:
  - Confidence: early detection of errors
  - Improved quality:
    - Decomposed architecture
    - Reuse
  - Product automation

Model characteristics

- Usually visual structure (diagrammatic)
  - But OCL is symbolic
- ***Partial***
- Abstract
- Business level
- Desirable feature: Testable
- Platform (technology) independent

- Difference from Programs:
  - Level of abstraction
  - Possibly no operational semantics

Model-Driven-Engineering (MDE)

- MDE =
  - Software development by repeated transformations of models
- Motivation
  - Growing complexity requires multiple levels of abstraction not supported by programming languages
- Technologies
  - Domain specific modeling languages (DSMLs)
  - Transformation engines
  - Model level development platforms
    - EMF, MetaEdit, Epsilon

Unified Modeling Language (UML)

- Widely accepted as modeling standard
  - OMG standard: http://uml.org/
- Collection of modeling diagrams
  - Each describes a view of OO software
- A UML model = A collection of diagrams
- Object Constraint Language (OCL) – textual
  - Invariants
  - Queries
  - Pre/post conditions
- UML standard specification is informal
  - Much research on its formalization

Are models used intensively?

- Not yet: not used throughout software evolution life cycle
  - Neglected in later stages of software development
- The missing link:
  - Automatic code generation
  - Reverse transformations
  - Model-level advanced IDEs

→ Users neglect essential modeling features
  - Why bother if features are not translated into the code?

Model-level integrated development environment

- Supports
  - Reasoning
  - Verification
  - Testing
  - Warnings
  - Refactoring
  - Patterns
  - Integration with code

Specification of Software Modeling Languages

- Modeling languages:
  - Syntax:
    - Abstract syntax
    - Concrete visual/symbolic syntax
  - Semantics:
    - Direct denotation: Class diagrams
    - Indirect (translation):
      - Sequence diagrams, LSC, statecharts: through automata
      - Class diagrams: though Logic, typed-graphs

Historical View on Modeling Languages

- 1960's – Output-oriented methods
- 1970's – Process-oriented methods (Structured System Analysis and Design – SSAD):
  - Use DFDs – Data Flow Diagrams.
- 1980's – Data-oriented methods:
  - Use ERDs – Entity-Relationship Diagrams.
- 1990's – Object-oriented methods (OMT, OOD, OOSE, UP):
  - Standard: UML – Unified Modeling Language.

The Unified Modeling Language (UML)

started at 1995

What Is the UML?

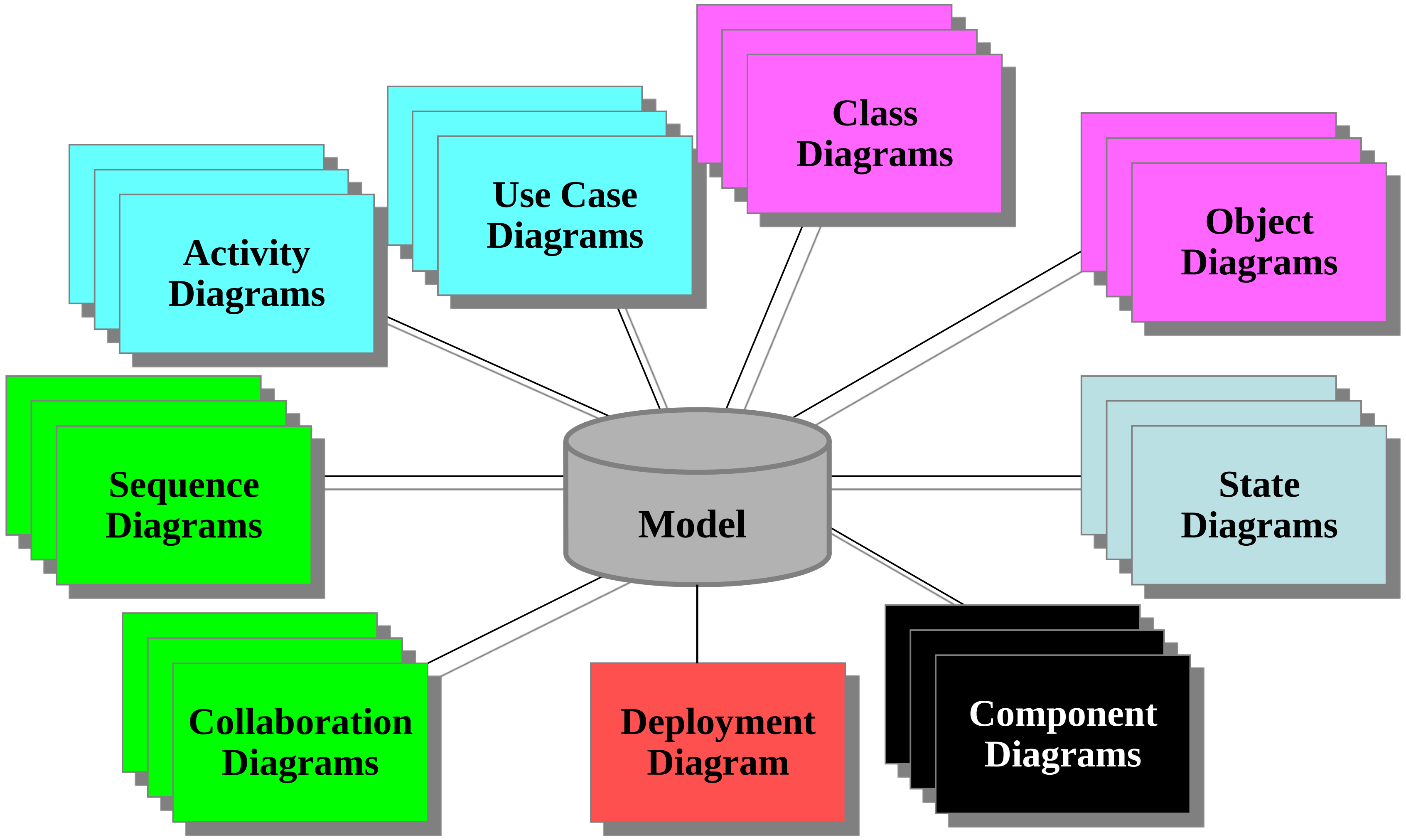
- The Unified Modeling Language (UML) is a family of independent languages for
    - Specifying
    - Visualizing
    - Constructing
    - Documenting

  different aspects of a software-intensive system.

UML Goals

- Define an easy-to-learn but semantically rich visual modeling language.
- Unify existing Object-Oriented modeling languages:
  - Booch, OMT, and OOSE modeling languages.
- Incorporate industry best practices.

The UML Visual Languages



The UML Modeling Languages

- Requirements engineering languages:
  - Use-cases and use-case diagrams.
  - Activity diagrams.
- Structure (static) modeling languages:
  - Class diagrams.
  - Object diagrams.
- Behavioral (dynamic) modeling languages:
  - Sequence diagrams; collaboration diagrams.
  - Statecharts.
- Implementation level languages:
  - Deployment diagrams.
  - Component diagrams.
- Object Constraints Language (OCL).

The UML Modeling Languages

- In this course we teach software modeling in various stages of software development.

- We use the following UML languages:
  - Use-cases and use-case diagrams.
  - Activity diagrams.
  - Class diagrams and object diagrams: In much detail.
  - Sequence diagrams, collaboration diagrams; Statecharts.
  - Object Constraints Language (mainly the static aspects).

Object-Oriented Analysis and Design

Session 2: Requirements

Back to Square 1 - Requirements

User Stories

With contributions from Sjaak Brinkkemper, Garm Lucassen, Fabiano Dalpiaz, Jan Martijn van der Werf Marcel Robeer, Ivor van der Schalk

What is a user story?

- “As a Visitor, I want to purchase an event ticket”
- “As a visitor, I want to search for new events by favorited organizers so that I am the first to know of new events”
- “As a Visitor, I want to be notified when an event is close to becoming sold out, so that I do not miss the event”

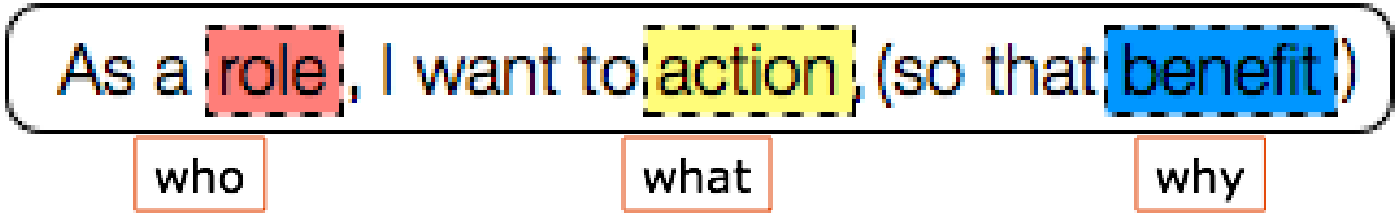
What is a user story?

- User stories represent customer requirements in a card, leading to conversation and confirmation

- User stories only capture the *essential* elements of a requirement:
  - *who* it is for
  - *what* it expects from the system
  - *why* it is important (optional?)

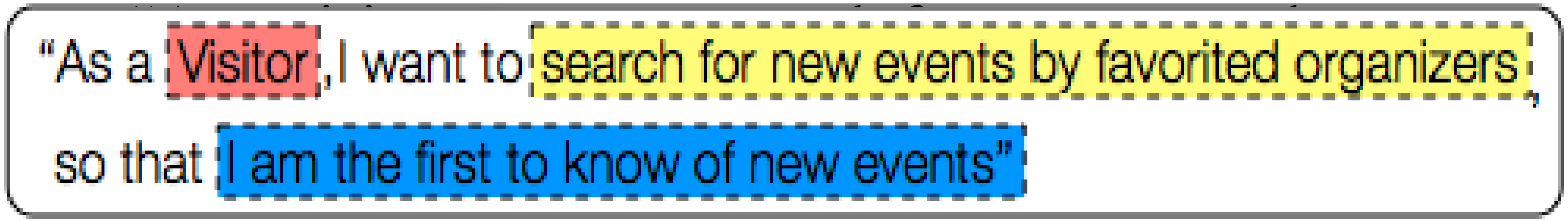
- Simple format used by 70% of practitioners

(Lucassen et al., 2016)



What is a user story?

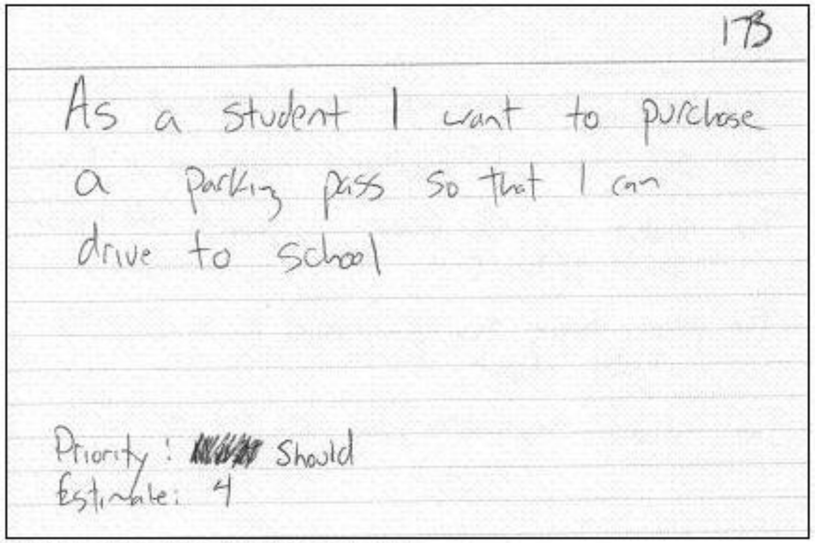
- “As a Visitor, I want to purchase an event ticket”



“As a Visitor, I want to search for new events by favorited organizers, so that I am the first to know of new events”

- “As a Visitor, I want to be notified when an event is close to becoming sold out, so that I do not miss the event”

Front of Card

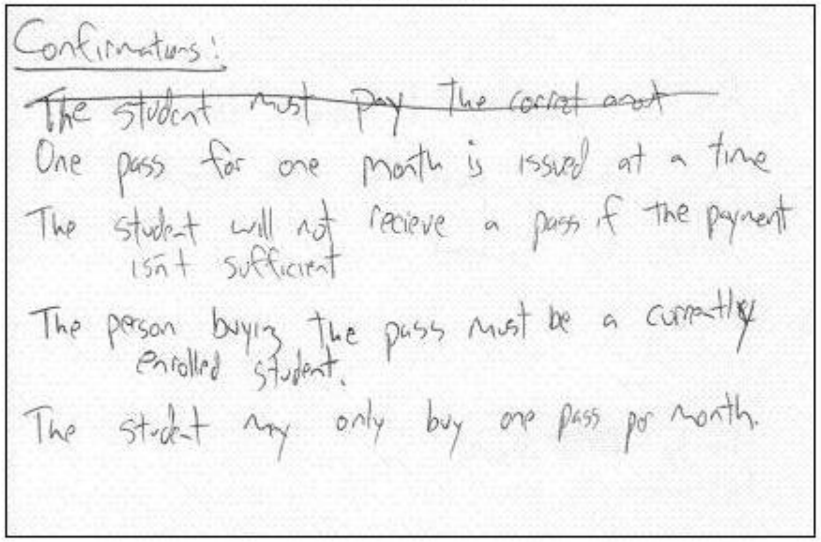


173

As a student I want to purchase a parking pass so that I can drive to school

Priority: ~~[illegible]~~ Should
Estimate: 4

Back of Card



Confirmations:

~~The student must pay The correct amt~~

One pass for one month is issued at a time

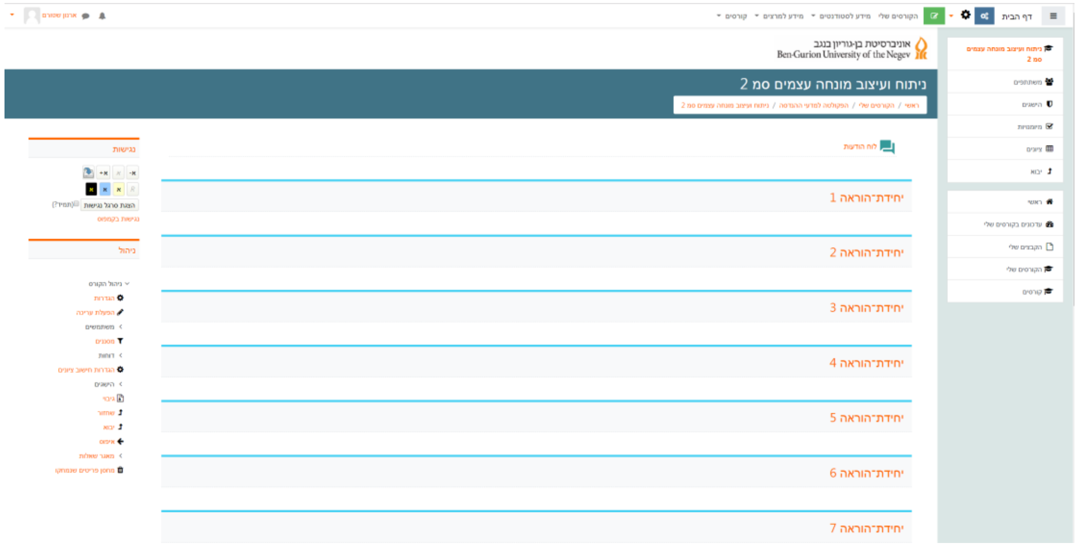
The student will not recieve a pass if the payment isn't sufficient

The person buying the pass must be a currently enrolled student.

The student may only buy one pass per month.

Exercise #1

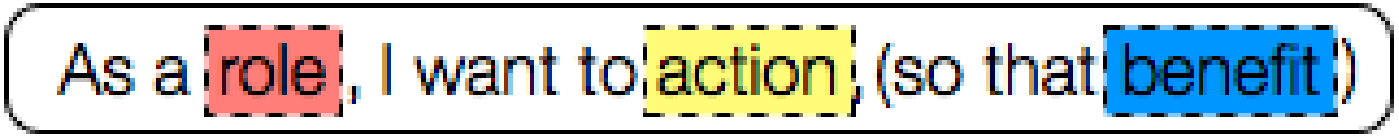
- Form groups of two students
- Use a sheet of paper or a text editor
- Write at least 15 user stories for a new course management system



Exercise evaluation

- Let us discuss a few user stories!
- What are the key roles?
- Discussion triggers
- Is the role the actual role?
- Did you specify the why part?
- Have you forced the format?
- Did you use the domain jargon?
- Are there technical details?

User story template



As a role, I want to action, (so that benefit)

Quality problems in practice

- Model captures correct stories
- Stories from practitioners:
  - Too long
  - Unnecessary information
  - Too little information
  - Inconsistent
  - Irrelevant
  - Ambiguous

INVEST?

  - Most popular and simple reminder for characteristics of a good user story:

- **I**ndependent: dependencies between user stories shall be avoided to the extent this is possible
- **N**egotiable: details of the story can be discussed during the iteration planning meetings
    - **V**aluable to the customer
    - **E**stimable there is enough details to estimate the effort required
    - **S**imple in effort, i.e., no big requirements
    - **T**estable with certain acceptance criteria

  - **Drawbacks**: non-specific, generic, qualitative

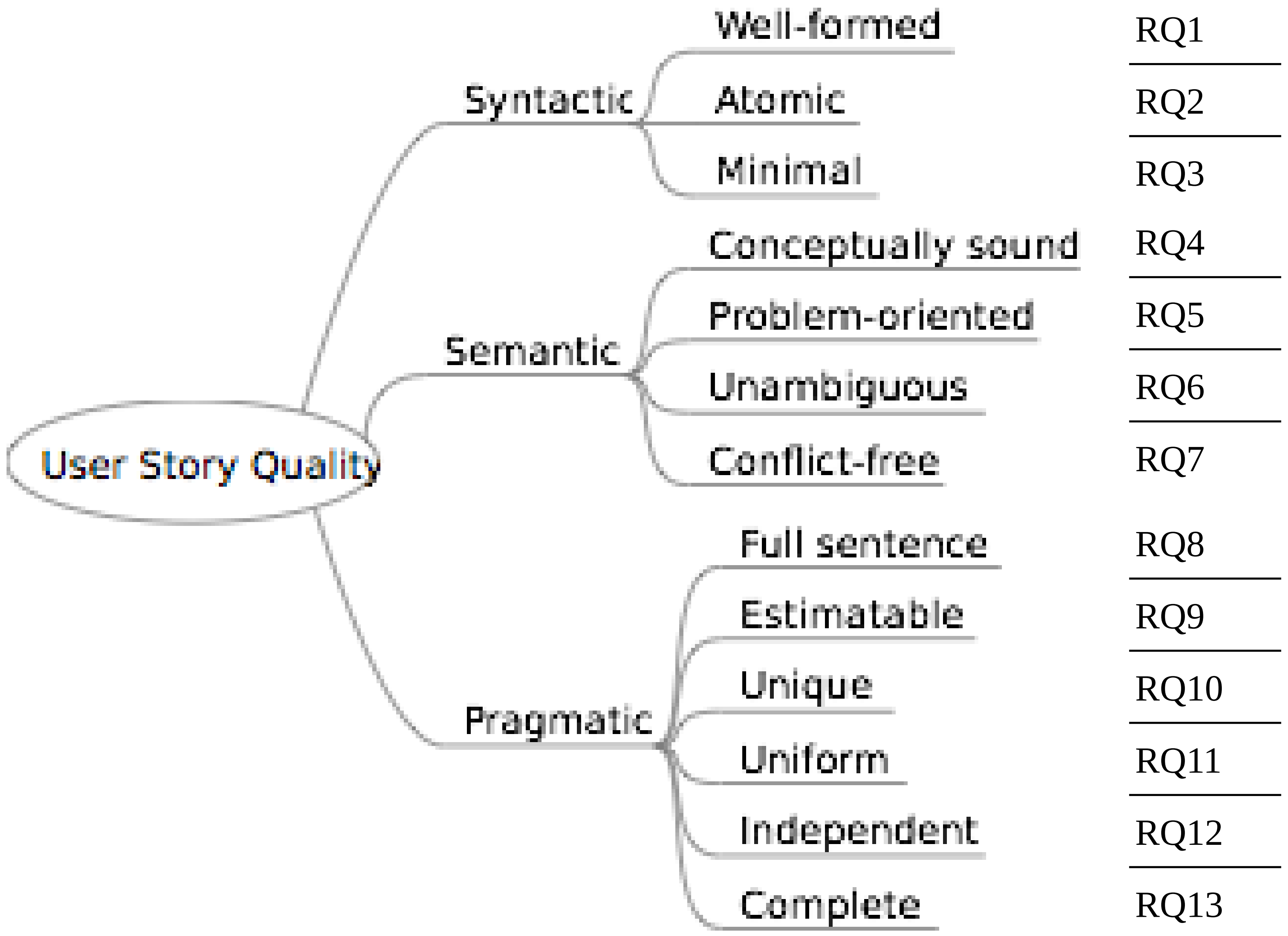
Exercise #2

- Take 5 user stories you have written
- Analyze them according to the INVEST framework
- Independent from each other
- Negotiable, i.e., no unnecessary details
- Valuable to the customer
- Estimable, enough details
- Small in effort
- For 2+ stories, write acceptance tests (Testable criterion)

Exercise Evaluation

- Let us discuss some of your stories
- How simple was it to use the template?
- Any doubtful case?
- How about writing acceptance criteria?

Quality User Story Framework


User Story Quality
Syntactic
Well-formed
RQ1
Atomic
RQ2
Minimal
RQ3
Semantic
Conceptually sound
RQ4
Problem-oriented
RQ5
Unambiguous
RQ6
Conflict-free
RQ7
Pragmatic
Full sentence
RQ8
Estimatable
RQ9
Unique
RQ10
Uniform
RQ11
Independent
RQ12
Complete
RQ13

Quality **U**ser **S**tory Framework

	Individual	
RQ1	Well-formed	A user story includes at least a role and an action
RQ2	Atomic	A user story expresses a requirement for exactly one feature
RQ3	Minimal	A user story contains nothing more than role, action and benefit
RQ4	Conceptually sound	The action expresses a feature and the benefit expresses a rationale
RQ5	Problem-oriented	A user story only specifies the problem, not the solution to it
RQ6	Unambiguous	A user story avoids terms or abstractions that lead to multiple interpretations
RQ8	Full sentence	A user story is a well-formed full sentence
RQ9	Estimatable	A story does not denote an unrefined requirement that is hard to plan and prioritize

	Set	
RQ7	Conflict-free	A user story should not be inconsistent with any other user story
RQ10	Unique	Every user story is unique, duplicates are avoided
RQ11	Uniform	All user stories in a specification employ the same template
RQ12	Independent	The user story is self-contained and has no inherent dependencies on other stories
RQ13	Complete	Implementing a set of user stories creates a feature-complete application, no steps are missing

Creating user stories

- Not all quality criteria immediately critical
- Focus on:

  RQ1. Well-formed

  RQ2. Atomic

  RQ3. Minimal

  RQ4. Conceptually sound

  RQ5. Problem oriented

  RQ8. Full sentence

  RQ11. Uniform
- Avoid premature optimization!

RQdemrof-llew -1

A user story includes at least a role and an action

I want to revoke access for problematic event organizers

RQ1 - well-formed

A user story includes at least a role and an action
I want to revoke access for problematic event organizers
⬇ add role
As a TicketExpert Employee, I want to revoke access for problematic event organizers

RQcimota -2

A user story expresses a requirement for exactly one feature/problem

As a Visitor, I want to register for an <u>event and create</u> a personal account, so that I can quickly register for more events in the future

RQ2 - atomic

A user story expresses a requirement for exactly one feature/problem

As a Visitor, I want to register for an event and create a personal account, so that I can quickly register for more events in the future

⬇ split

1. As a Visitor, I want to register for an event, so that I am admitted to the event
2. As a Visitor, I want to create a personal account during event registration, so that I can quickly register for more events in the future

RQ3 - minimal

A user story contains nothing more than role, action and benefit
As an Event Organizer, I want to see the personal information of attendees <u>(split into price levels). See: Mockup by Alice NOTE: - First create the overview screen</u>

RQ3 - minimal

A user story contains nothing more than role, action and benefit
As an Event Organizer, I want to see the personal information of attendees <u>(split into price levels). See: Mockup by Alice NOTE: - First create the overview screen</u>
⬇ (re)move unnecessary information
As an Event Organizer, I want to see the personal information of attendees

RQ4 - conceptually sound

The action expresses a feature and the benefit expresses a rationale
As an Event Organizer, I want to open the event page, so that I can see the personal information of attendees
⬇

RQdnuos yllautpecnoc -4

The action expresses a feature and the benefit expresses a rationale

As an Event Organizer, I want to open the event page,
so that <u>I can see the personal information of attendees</u>

⬇ ends becomes separate means

1. As an Event Organizer, I want to open the event page, so that I can review event related information
2. As an Event Organizer, I want to <u>see personal information of attendees</u>, so that I know the demographical distribution of the event

RQ5 - problem oriented

A user story only specifies the problem, not the solution to it

As a Visitor, I want to download an event ticket. <u>- Add download button on top right (never grayed out)</u>

⬇

RQ5 - problem oriented

A user story only specifies the problem, not the solution to it
As a Visitor, I want to download an event ticket. <u>- Add download button on top right (never grayed out)</u>
⬇ remove solution
As a Visitor, I want to download an event ticket

RQ8 - full sentence

A user story is a well-formed full sentence

update profile

RQecnetnes lluf -8

A user story is a well-formed full sentence

update profile

⬇ add ‘want to’

As a Visitor, I want to update my profile

RQ 11 - uniform

All user stories follow roughly the same template

1. As a Visitor, I want to create an account
2. As a Visitor, I want to reset my password
3. As a TicketExpert Manager, <u>I receive</u> an email notification when a new user is registered

RQ mrofinu -11

All user stories follow roughly the same template

1. As a Visitor, I want to create an account
2. As a Visitor, I want to reset my password
3. As a TicketExpert Manager, <u>I receive</u> an email notification when a new user is registered

⬇ add ‘want to’

As an TicketExpert Manager, <u>I want to</u> receive an email notification when a new user is registered

Estimating and developing

- Remaining criteria gradually become relevant
- Focus on:

  RQ6. Unambiguous
  RQ7. Conflict-free
  RQ9. Estimatable
  RQ12. Independent
  RQ10. Unique
  RQ13. Complete
- Keep iterating!

RQ6 - unambiguous

A user story avoids terms or abstractions that lead to multiple interpretations

As an Event Organizer, I want to edit <u>the content</u> that I added to an event's page

⬇

RQ6 - unambiguous

A user story avoids terms or abstractions that lead to multiple interpretations
As an Event Organizer, I want to edit <u>the content</u> that I added to an event's page
⬇ clarify the content
As an Event Organizer, I want to edit <u>video and text content</u> that I added to an event's page

RQ7 - conflict-free

A user story should not be inconsistent with any other user story

1. As an Event Organizer, I’m able to edit any event 2. As an Event Organizer, I’m able to delete only the events that I added

⬇

RQ7 - conflict-free

A user story should not be inconsistent with any other user story

1. As an Event Organizer, I'm able to edit any event 2. As an Event Organizer, I'm able to delete only the events that I added

⬇ change 1

1. As an Event Organizer, I'm able to edit <u>events that I added</u>

RQ9 - estimatable

A story does not denote a coarse-grained requirement that is difficult to plan and prioritize
As an Event Organizer, I want to see my task list during the event, so that I can prepare myself (for example I can see at what time I should start traveling)

RQ9 - estimatable

A story does not denote a coarse-grained requirement that is difficult to plan and prioritize

As an Event Organizer, I want to see my task list during the event, so that I can prepare myself (for example I can see at what time I should start traveling)

⬇ split

1. As an Event Employee, I want to see my task list during the event, so that I can prepare myself
2. As an Event Organizer, I want to <u>upload a task list for event employees</u>

RQ10 - unique

Every user story is unique, duplicates are avoided

1. As a Visitor, I’m able to see a list of new events, so that I stay up to date
2. As a Visitor, I’m able to see a list of new events, so that I stay up to date

RQ10 - unique

Every user story is unique, duplicates are avoided

1. As a Visitor, I’m able to see a list of new events, so that I stay up to date
2. As a Visitor, I’m able to see a list of new events, so that I stay up to date

⬇ remove one

1. As a Visitor, I’m able to see a list of news items, so that I stay up to date

RQ12 - independent

The user story is self-contained and has no inherent dependencies on other user stories

1. As an Event Organizer, I am able to add a new event
2. As a Visitor, I am able to view an event page???

⬇

RQ12 - independent

The user story is self-contained and has no inherent dependencies on other user stories

1. As an Event Organizer, I am able to add a new event
2. As a Visitor, I am able to view an event page

⬇???

- Try to avoid dependencies as much as possible
- Impossible to be fully independent
- But… always remain flexible!

RQ13 - complete

Implementing a set of user stories creates a feature-complete application, no steps are missing
1. As an Event Organizer, I want to update an event 2. As an Event Organizer, I want to delete an event

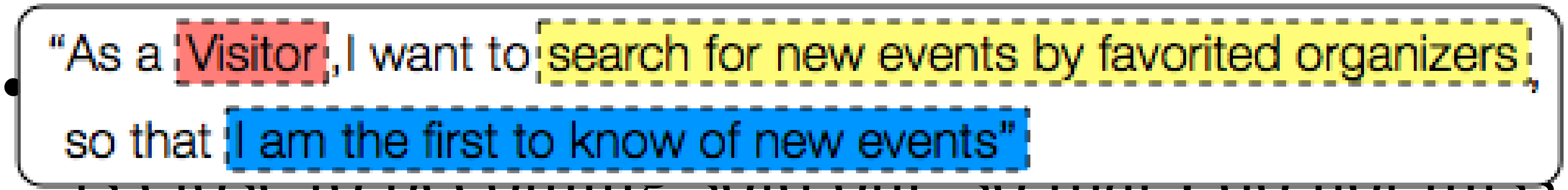
RQ13 - complete

Implementing a set of user stories creates a feature-complete application, no steps are missing
1. As an Event Organizer, I want to update an event 2. As an Event Organizer, I want to delete an event
⬇add story
As an Event Organizer, I want to create an event

Going from User stories to models

- “As a Visitor, I want to buy an event ticket”

-


“As a Visitor, I want to search for new events by favorited organizers, so that I am the first to know of new events”

is close to becoming sold out, so that I do not miss the event”

Going from User stories to models

Noun Subject	→	Concept
Verb (including preposition)	→	Relationship
Verb ‘to be’	→	Hierarchical relationship

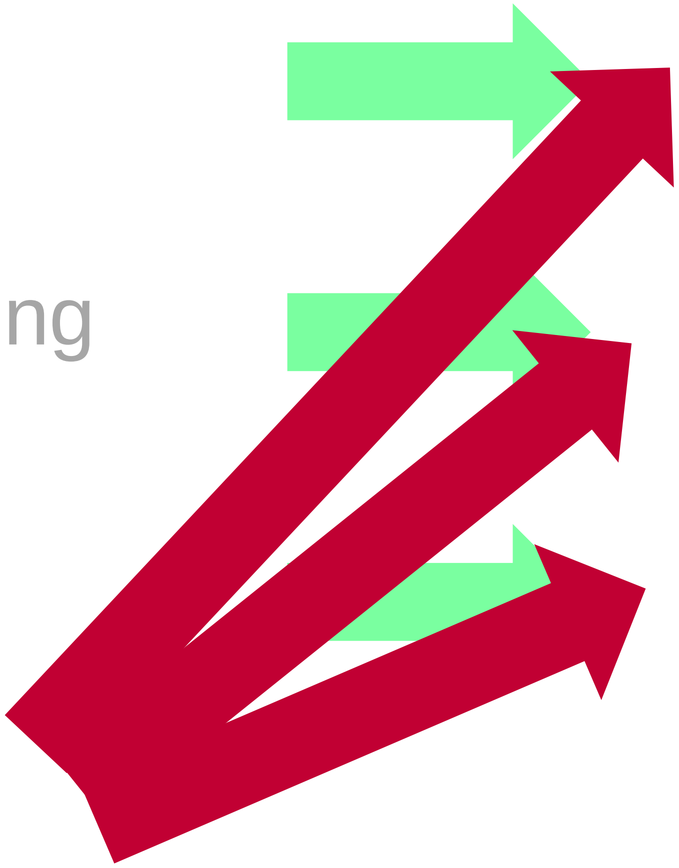
Going from User stories to models

Noun
Subject

Verb (including preposition)

Verb ‘to be’

Noun-noun compound



Concept

Relationship

Hierarchical relationship

Functional role

Role	As a visitor
Means	I want to choose an event
End	so that I can book a ticket for that event

Simplify the means

Role	As a visitor
Means	I ~~want to~~ choose an event
End	so that I can book a ticket for that event

Simplify the means

Role	As a visitor
Means	I ~~want to~~ choose an event
End	so that I can book a ticket for that event

Main verb & main object

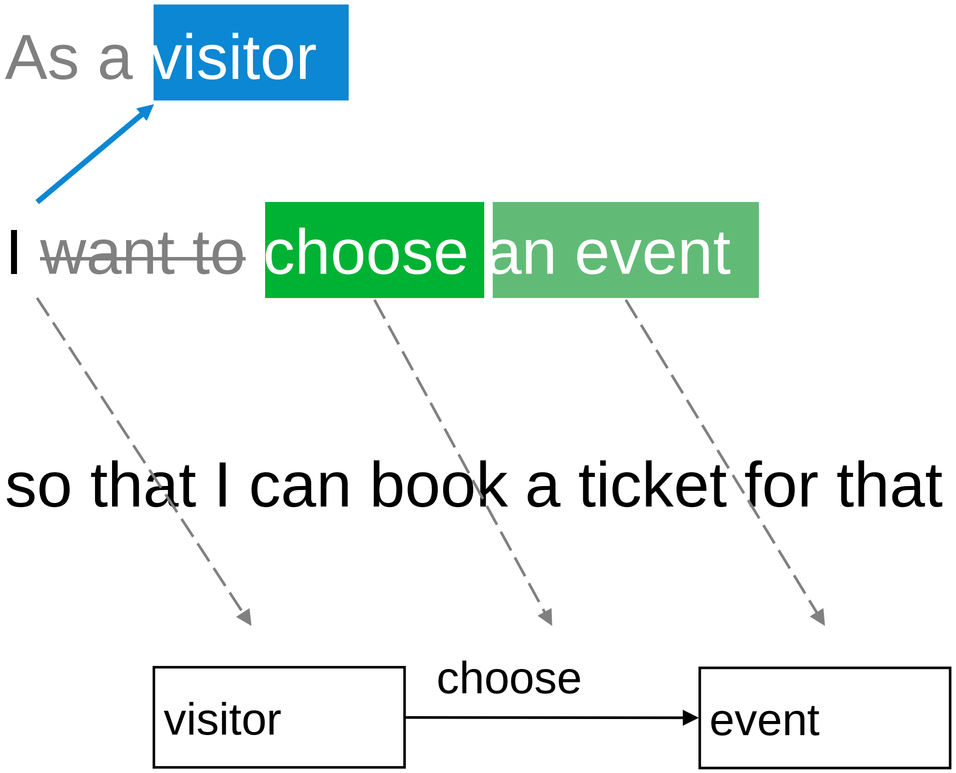
Role	As a visitor
Means	I ~~want to~~ choose an event
End	so that I can book a ticket for that event

Main relationship

Role — As a visitor

Means — I ~~want to~~ choose an event

End — so that I can book a ticket for that event

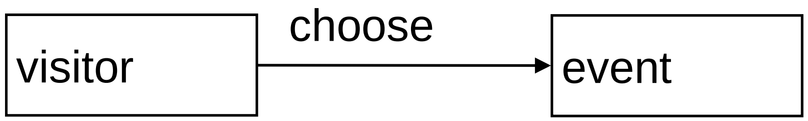


Remaining information

Role — As a visitor

Means — I ~~want to~~ choose an event

End — so that I can book a ticket for that event

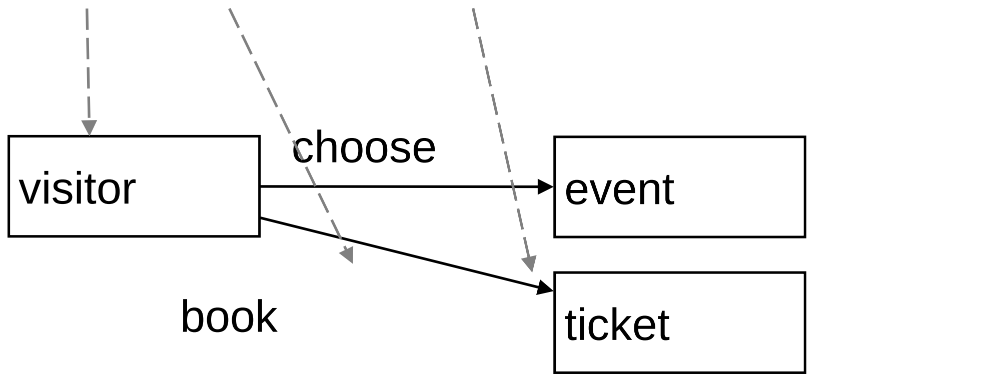


Remaining information

Role As a visitor

Means I ~~want to~~ choose an event

End so that I can book a ticket for that event

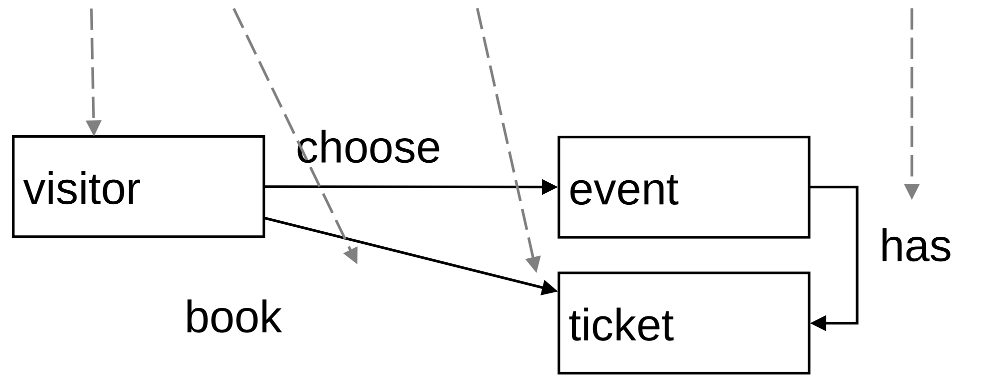


Remaining information

Role — As a visitor

Means — I ~~want to~~ choose an event

End — so that I can book a ticket for that event



visitor
choose
event
has
book
ticket