SEIS 610

Chapter 13

1

in: Z User Group, Organizations, Z notation

Z User Group

NEW Last ABZ conference:

ABZ 2018: Southampton↑, United Kingdom↑, 5-8 June 2018

Previous ABZ conferences:

ABZ 2016: Linz↑, Austria↑, 23-27 May 2016

ABZ 2014: Toulouse↑, France↑, 2-6 June 2014

ABZ 2012: Pisa↑, Italy↑, 18-22 June 2012

ABZ 2010: Orford, Quebec↑, Canada↑, 23–25 February 2010

First ABZ conference:

2

Agenda

- Review Object Oriented-Analysis (Chapter 13)
  - Analysis Workflow (13.1)
  - Extracting Entity Classes (13.2)
    - Noun Extraction (13.5.1)
  - CRC Cards (13.5.2)
    - (Only because they remind me of the play’s we are writing)
  - Test Workflow (13.7)
  - Extracting Boundary and Control Classes (13.8)
  - Specification Document (13.18)
  - Actors and Use Cases (13.19)
  - Metrics (13.21)
  - Challenges to Object-Oriented Analysis (13.22)

3

Object Oriented Analysis

- Entity Class – information that is long lived
- Boundary Class – models' interaction between product and its actors
- Control Class – Models complex computations and algorithms

4

Object Oriented Analysis

- Entity Class – information that is long lived
- Boundary Class – models interaction between product and its actors
- Control Class – Models complex computations and algorithms
- MVC
- MVC is an architectural model for the UI based world!
- Entity, Boundary, Control is for business/and other modeling

5

MVC

- Model –Represents the problem domain, maintain state, and provide methods for accessing and mutating the state of the application.
- The Controller's job is to translate incoming requests into outgoing responses.
- View : The View's job is to translate data into a visual rendering for response to the Client

- http://www.bennadel.com/blog/2379-a-better-understanding-of-mvc-model-view-controller-thanks-to-steven-neiland.htm

6

Quick Pause

- Design Patterns
- "In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**. A **design pattern** isn't a finished **design** that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations."

- https://sourcemaking.com/design_patterns

7

Object Oriented Analysis

- Entity Class – information that is long lived
- Boundary Class – models interaction between product and its actors
- Control Class – "Controls" complex computations and algorithms

- https://www.youtube.com/watch?v=JWcoiXNoKxk&feature=youtu.be&t=15m14s

(start 15 minutes in!)

8

Extracting Entity Classes (13.2)

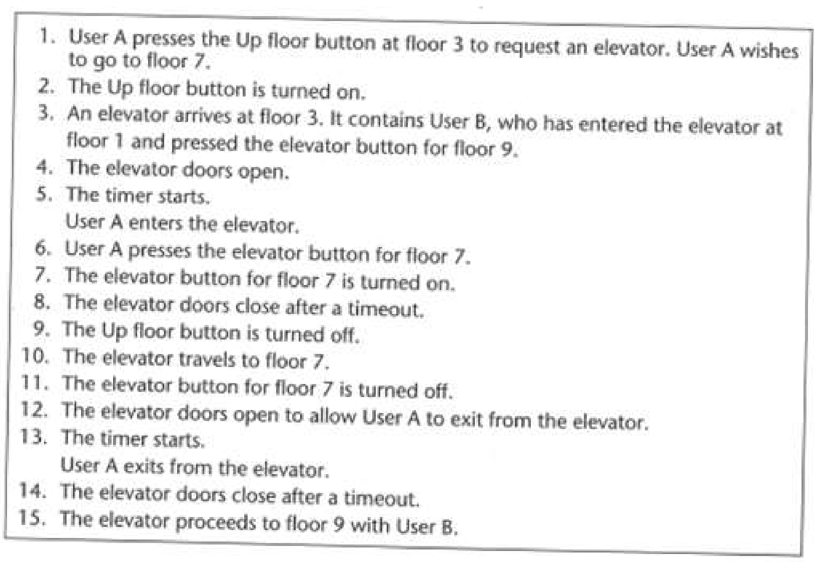
- Functional Modeling
- Entity Class Modeling
- Dynamic Modeling

9

Functional Modeling

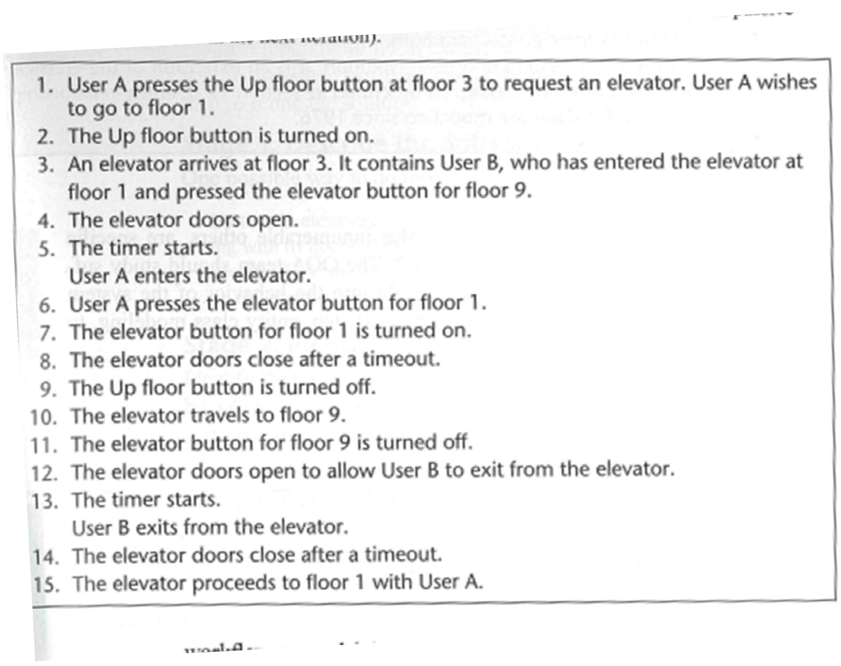
- A functional modeling perspective concentrates on describing the dynamic process.
- The main concept in this modeling perspective is the process, this could be a function, transformation, activity, action, task etc.
- A well-known example of a modeling language employing this perspective is data flow diagrams.
- It is illustrative to think like this?
  - Think of data and processes identified in the DFD?
  - You can turn that data into classes
- https://en.wikipedia.org/wiki/Function_model
- Another way of looking at this is with use cases

10



1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
   User A enters the elevator.
6. User A presses the elevator button for floor 7.
7. The elevator button for floor 7 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.
    User A exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 9 with User B.

11



1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
   User A enters the elevator.
6. User A presses the elevator button for floor 1.
7. The elevator button for floor 1 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
    User B exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 1 with User A.

12

Use Case

- **Use case:** User/System actions that are required to reach or abandon a goal. A set of success and failure scenarios that describe an actor reaching or abandoning their goal.
- **Use case scenario:** A single path through the use case.
- **Use case instance**: A sequence of actions a system performs that yields an observable result of value to a particular actor.
- **Use case model:** All the written use cases that describe a system's functional requirements.
- **Actor:** Role external party/parties that interact with the system. Does not have to be human! Anything with a behavior.

13

Use Case Types

- **brief**—Terse one-paragraph summary, usually of the main success scenario.
- **casual**—Informal paragraph format. Multiple paragraphs that cover various scenarios.
- **fully dressed**—All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.
- Two column is my favorite variation

14

13.19 Use Case Vocab

- **Primary Actor:** The primary actor is trying to achieve a goal. Also the actor that initiated use case.
- **Supporting Actor:** actor—provides a service (for example, information) to the SuD.
  - An automated payment authorization service is an example.
  - could be an organization or person.
- **Offstage Actor:** Has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.

15

Finding Use Cases

- Choose a system boundary
  - Software? Hardware? A person?
- Identify the primary actors
  - Those that will have goals fulfilled using the software
- Enumerate their goals
- Define use cases that satisfy their goals.

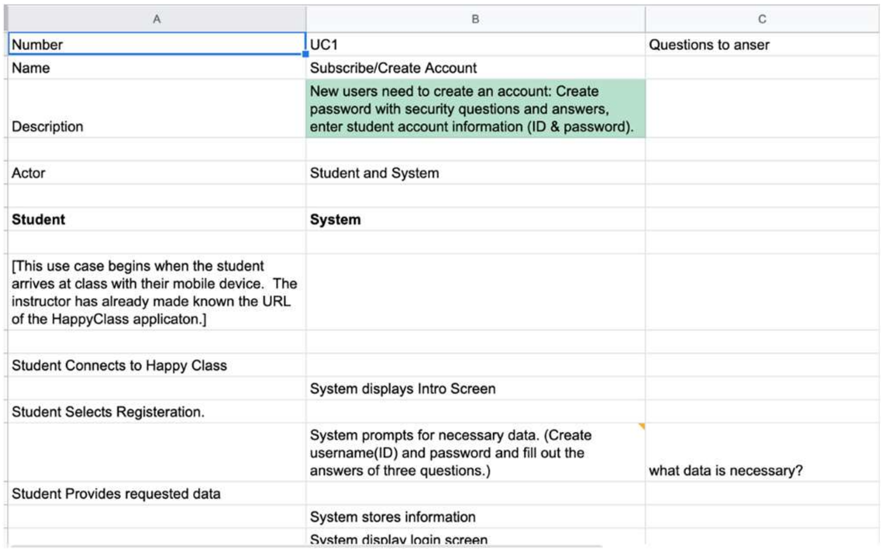
16

About Use Cases

1. The name should start with a strong verb.
2. A use case is a set of scenarios.
    - A scenario is a list of steps.
3. Each step should state what the user does and/or what the system responds.
4. **The steps must not mention how the system does something. Keep the steps essential or logical -- no colors, clicks, typing!**
5. Each step needs to be analyzed in detail before it becomes code.
6. Keep It Simple: use the simplest format you need.
7. Refine interesting use cases first.
8. Use cases are not object oriented
    - That would be too specific
    - They should not identify objects
9. Finally, an activity diagram might do the trick! (No use case needed)

17

Sample 2 Column



A	B	C
Number	UC1	Questions to anser
Name	Subscribe/Create Account	
Description	New users need to create an account: Create password with security questions and answers, enter student account information (ID & password).	
Actor	Student and System	
Student	**System**	
[This use case begins when the student arrives at class with their mobile device. The instructor has already made known the URL of the HappyClass applicaton.]		
Student Connects to Happy Class		
	System displays Intro Screen	
Student Selects Registeration.		
	System prompts for necessary data. (Create username(ID) and password and fill out the answers of three questions.)	what data is necessary?
Student Provides requested data		
	System stores information	
	System display login screen	

18

More about use cases

1. Make sure you store use cases so that they are easily found, edited, and used.
2. Put use cases on a project web site. (confluence or wiki)
3. Keep track of different versions.
4. Writing use cases is a team sport.
5. Focus on a particular user (give them a name) in each use case and each step.
6. **Don't get bogged down in all the special ways it can go wrong until you've finished the main success story.**
7. http://www.csci.csusb.edu/dick/samples/usecases.html
8. Finally Use Case Diagrams are pretty, but not sufficient.

19

Include/Extend

- Extend is used when a use case conditionally adds steps to another first class use case.
- Include is used to extract use case fragments that are duplicated in multiple use cases.
  - These fragments are always run.

20

Entity Class Modeling

- Determine the entity classes and attributes
- Model their relationships
- Create a class diagram

21

13.5.1 Noun Extraction (Entity class modeling)

- Review use case or user stories
- Identify the nouns
- Keep track of the verbs

22

CRC Cards (13.5.2) (Entity Class Modeling)

- Class-Responsibility-Collaboration Cards
- Interaction among team members can highlight missing or incorrect aspects of a class.
- Relationships between classes are highlighted.
- Does not do much for entity classes
  - But you should know the domain by now
  - You should be able to produce the entity classes
- Standard index cards divided into sections

23

Review CRC Cards

- http://agilemodeling.com/artifacts/crcModel.htm

24

CRC Card (re-iterating creation)

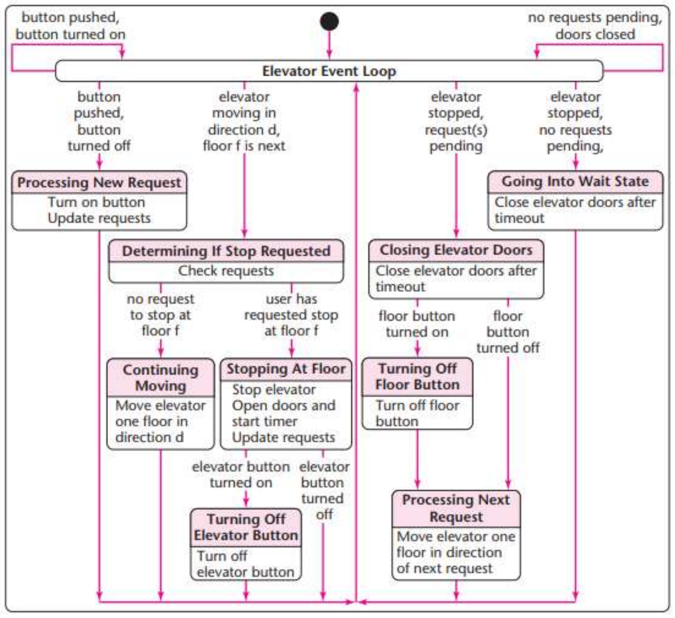
1. Find 3 to 5 classes
   - Noun extraction? Guess?
2. Define Responsibilities
   - What do they do and what do they know?
3. Define Collaborators
   - What do they need done or what information do they need?

25

Dynamic Modeling (13.6, page 414)

- The **dynamic model** represents the time–dependent aspects of a system.
- It is concerned with the temporal changes in the states of the objects in a system
- https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_dynamic_modeling.htm
- Think State diagrams and Petri Nets

26


button pushed, button turned on
no requests pending, doors closed
Elevator Event Loop
button pushed, button turned off
elevator moving in direction d, floor f is next
elevator stopped, request(s) pending
elevator stopped, no requests pending,
Processing New Request
Turn on button
Update requests
Going Into Wait State
Close elevator doors after timeout
Determining If Stop Requested
Check requests
Closing Elevator Doors
Close elevator doors after timeout
no request to stop at floor f
user has requested stop at floor f
floor button turned on
floor button turned off
Continuing Moving
Move elevator one floor in direction d
Stopping At Floor
Stop elevator
Open doors and start timer
Update requests
Turning Off Floor Button
Turn off floor button
elevator button turned on
elevator button turned off
Turning Off Elevator Button
Turn off elevator button
Processing Next Request
Move elevator one floor in direction of next request

Page 415 in your text

27

How to write a play (twist on crc)

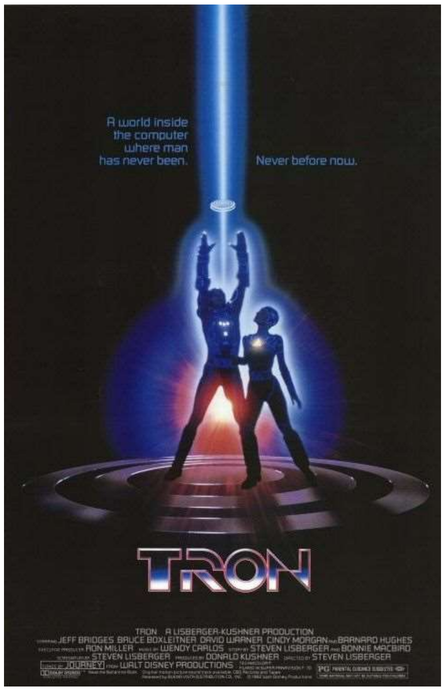
- Come up with a main character.
- Decide on conflict
  - Your play should have a conflict. Give your character a major problem that he or she has to solve immediately.
- Decide on a beginning point
- Show the story in actions and "speech"
  - Ok just actions for us
- Don't over do it!

- http://www.creative-writing-now.com/how-to-write-a-play.html
- http://www.dummies.com/how-to/content/playwriting-for-dummies-cheat-sheet.html

28

Play's!

- Idea for modeling using a 'play' is based on identifying responsibility
- Play's seem more fun than CRC cards.
- CRC cards might be more agile over the long run.



29

Test Workflow (13.7)

- Review Classes
- Review Models
- Start thinking of a test plan

30

Test workflow (13.7)

- Now is the time to start writing the test plan
- We should have a good collection of:
  - entity, boundary and control classes.
- Or if you prefer:
  - Model, View and Controller classes
- We should have a **good dynamic model**
- We should have **a good static model**
- We can write and refine test cases.
  - Why do I say refine??

31

Next slides

- Attribution:
- http://www.guru99.com/test-case.html

32

Test Cases Best Practices

1. Keep simple and transparent
2. Keep user in mind
3. Avid Repetition
4. Make no assumptions
5. Ensure 100% Coverage
6. Test Cases must be identifiable
7. Implement Testing Techniques
   a. Boundary Value Analysis
   b. Equivalence Partition
   c. Stat Transitions
   d. Error Guessing
8. Self Cleaning
9. Repeatable and self-standing
10. Peer review

http://www.guru99.com/test-case.html

33

Test Workflow

- https://www.youtube.com/watch?v=BBmA5Qp6Ghk
  - More detail: http://www.guru99.com/test-case.html

- https://youtu.be/9abf1eQephA

34

Guru99.com Test template

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	1.Go to sitehttp://demo.guru99.com 2.Enter UserId 3.Enter Password 4.Click Submit	Userid = guru99 Password = pass99	User should Login into application	As Expected	Pass

35

Test Case Management Tools

- **For documenting Test Cases**
  - With tools you can expedite Test Case creation with use of templates
- **Execute the Test Case and Record the results:**
  - Test Case can be executed through the tools and results obtained can be easily recorded.
- **Automate the Defect Tracking:**
  - Failed tests are automatically linked to the bug tracker , which in turn can be assigned to the developers and can be tracked by email notifications.
- **Traceability:**
  - Requirements, Test cases, Execution of Test cases are all interlinked through the tools, and each case can be traced to each other to check test coverage.
- **Protecting Test Cases:**
  - Test cases should be reusable and should be protected from being lost or corrupted due to poor version control. Test Case Management Tools offer features like
- **Naming and numbering conventions**
  - **Versioning**
  - Read only storage
  - Controlled access
  - Off-site backup
- Popular Test Management tools are : Quality Center and JIRA

36

Extracting Boundary and Control Classes (13.8)

- Each input screen, output screen, report, etc. Everything generated to show the user is a candidate for a boundary class
- Control classes are identified by algorithms that need implementing.
- They should be visible in a dfd.

37

Extracting Boundary and Control Classes (13.8) (Doing MVC)

- Each input screen, output screen, report, etc. Everything generated to show the user is a candidate for view. (View!)
- “model” classes are identified by algorithms that need implementing.
  - Non trivial operation/calculation, turn into class.
- They are also visible in your dfd.
- Controller
  - It is a class.
  - All the things that need to be bossed around!

38

Specification Document (13.18)

- A **Software Requirements Specification** (SRS) is a technical document that describes in detail the externally visible characteristics of a software product
- Parts of the SRS include: Environmental requirements: OS, platform, interoperability, standards, etc.
  - Non-functional requirements: security, usability, efficiency, etc.
  - Feature specifications: precisely describe each feature
  - Use cases: examples of how a user accomplishes a goal by using one or more features
- Test Plan
- SRS and Test Plan give enough detail
- http://www.jrobbins.org/ics121f03/lesson-spec-design.html

39

13.21 Metrics

- Book suggests number of pages in UML diagram
  - Count how many classes are defined
  - Count how many sequence diagrams are created
- Count how many errors you find during reviews

40

13.21 Metrics for the Object-Oriented Analysis Workflow

- As with the other core workflows
  - It is essential to measure the five fundamental metrics: size, cost, duration, effort, and quality
  - It is essential to keep accurate fault statistics
- A measure of size of the object-oriented analysis
  - Number of pages of UML diagrams

41

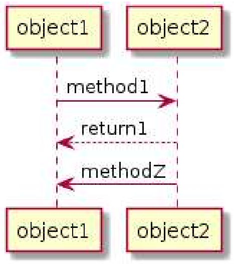
13.22 Challenges of the Object-Oriented Analysis Workflow

- Do not cross the boundary into object-oriented design
- Do not concern yourself with methods to classes *yet*
  - Reallocating methods to classes during stepwise refinement is wasted effort

42

Sequence Diagrams

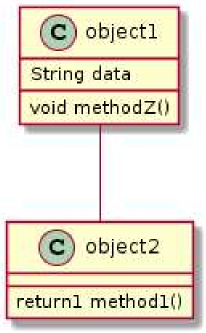
- A **Sequence diagram** is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart.
- Do not spend a lot of time on the books sequence diagrams or use case diagrams.
- Solid line means message to.
- Thank you wiki



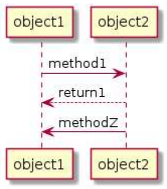
43

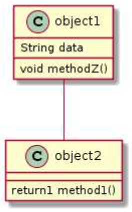
Class Diagrams

- Diagram that describes the structure of a system
  - the system's classes, their attributes
  - operations (or methods)
  - and the relationships among objects
- Class diagrams are static
- These are a key part architecture of a system
- Thank you wiki



44





45

Sample Sequence Diagram

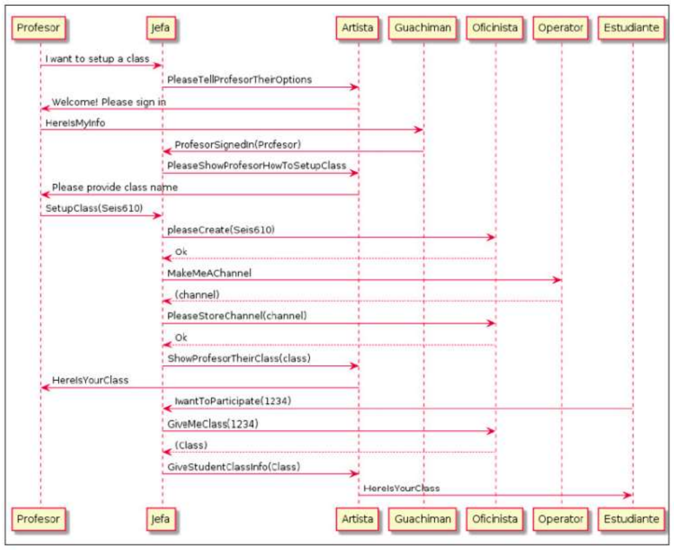
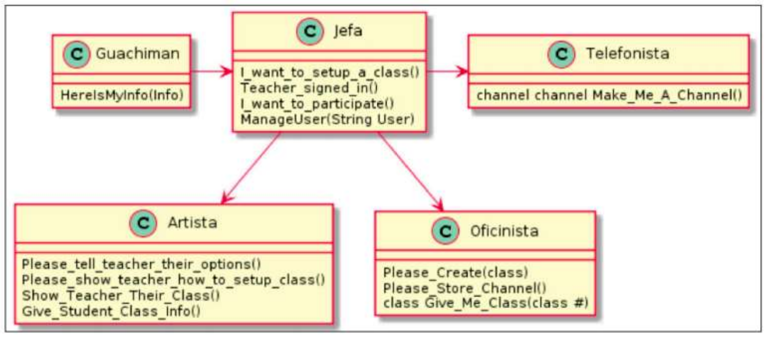


Figure 1. Example sequence diagram

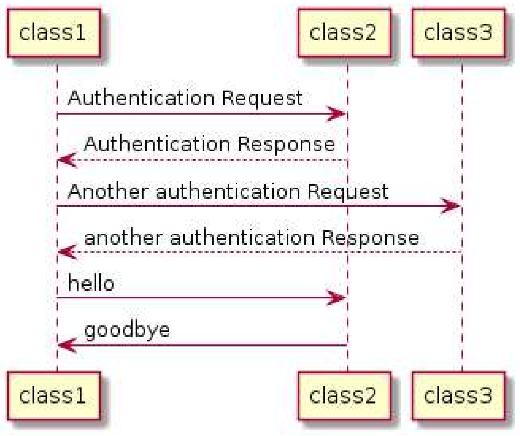
46

Sample Class Diagram


Guachiman
HereIsMyInfo(Info)
Jefa
I_want_to_setup_a_class()
Teacher_signed_in()
I_want_to_participate()
ManageUser(String User)
Telefonista
channel channel Make_Me_A_Channel()
Artista
Please_tell_teacher_their_options()
Please_show_teacher_how_to_setup_class()
Show_Teacher_Their_Class()
Give_Student_Class_Info()
Oficinista
Please_Create(class)
Please_Store_Channel()
class Give_Me_Class(class #)

47

Quiz Part 1: Draw the class diagram for this sequence diagram



48

- The end!

49