Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Matteo Camilli, Elisabetta Di Nitto, and Matteo Rossi

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering 2 – Written Exam 1 (WE1)

July 14, 2023

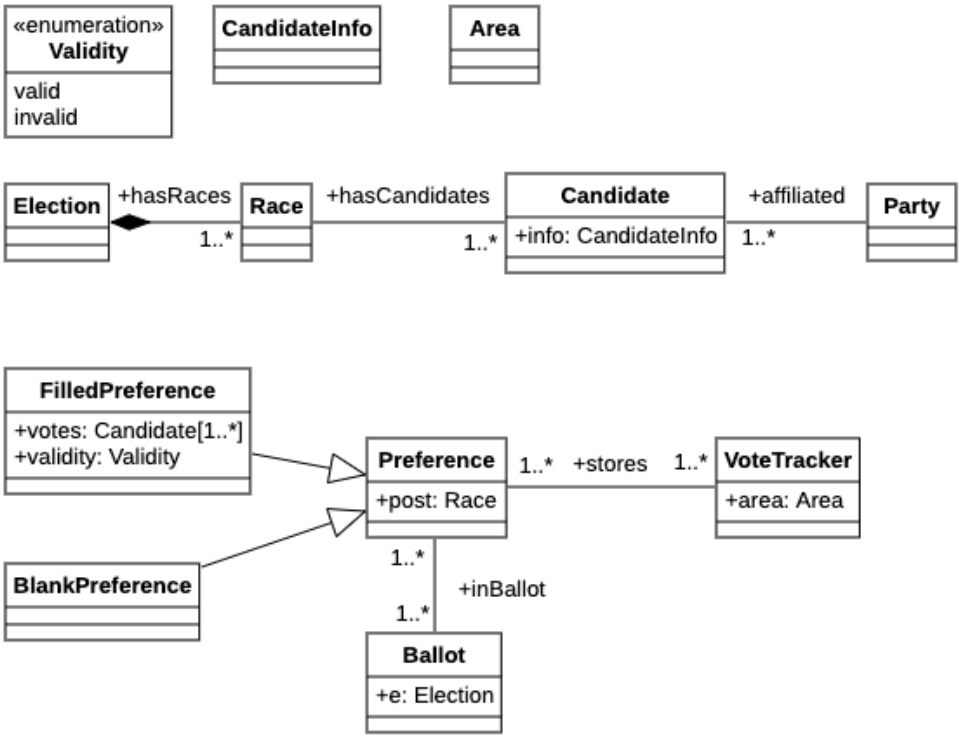
Last name, first name and Id number (Matricola)
Number of paper sheets you are submitting as part of the exam

Notes

A. Remember to write your name and Id number (matricola) on each piece of paper that you hand in.
B. You may use a pencil.
C. Unreadable handwriting is equivalent to not providing an answer.
D. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden, except for an ebook reader or a plain calculator.
E. The exam is composed of three exercises. Read carefully all points in the text.
F. Total available time for WE1: 1h and 30 mins

Question 1 – Alloy (8 points)
Consider a system for the counting of votes in elections. Each election is made of different races, one for each post (mayor, senator, etc.). Each race has a set of candidates who run for that race, where each candidate has personal information (e.g., first name, last name, etc.) which uniquely identifies the candidate. Also, each candidate is affiliated with a party. Voters fill in ballots to express their preferences for each race. Each preference can be blank (i.e., the voter did not vote for anyone), or it could list one candidate. Even if the voter voted for one candidate, the preference might be invalid (for example if the ballot reader was not able to identify the candidate with sufficient confidence, but also for other reasons). The system has a component (a vote tracker) that keeps track of the votes in a precinct (i.e., in a certain geographical area). Each vote tracker is uniquely identified by the area it covers.

The following UML Class Diagram captures the domain of the system described above:



Alloy_1 (6 points) Define suitable signatures (and any necessary associated constraint) that capture the UML Class Diagram depicted above.

Alloy_2 (2 points) Define a constraint that states that a ballot contains exactly one preference for each race of the associated election.

Solution

Alloy_1

```
sig CandidateInfo{}
sig Area{}
sig Party{}

abstract sig Validity {}
one sig Valid extends Validity {}
one sig Invalid extends Validity {}

sig Candidate {
  info : CandidateInfo,
  affiliated: Party
}


sig Election {
  hasRaces : some Race,
}

sig Race {
```

```
  hasCandidates : some Candidate,
}

abstract sig Preference{
  post : Race
  inBallot : some Ballot
}

sig BlankPreference extends Preference{}
sig FilledPreference extends Preference{
  votes : some Candidate
  validity : Validity
}

sig Ballot{
  e : Election
}

sig VoteTracker{
  area : Area
  stores : some Preference
}

fact uniquenessAndExistenceConstraints {
  all disj c1, c2: Candidate | c1.info != c2.info
  all p : Party | some c: Candidate | c.affiliated = p
  all r : Race | one e: Election | r in e.hasRaces
  all c : Candidate | one r: Race | c in r.hasCandidates
  all b : Ballot | some p: Preference | b in p.inBallot
  all p : Preference | some vt: VoteTracker | p in vt.stores
  all disj vt1, vt2: VoteTracker | vt1.area != vt2.area
}
```

Alloy_2

```
fact noBallotsForWrongRaces {
  all b: Ballot |
      // the ballot contains only preferences related to races of
      // the associated election
      b.~inBallot.post in b.e.hasRaces
      and
      // each race has exactly one preference
      all r : b.e.hasRaces | one p : b.~inBallot | p.post = r
}
```

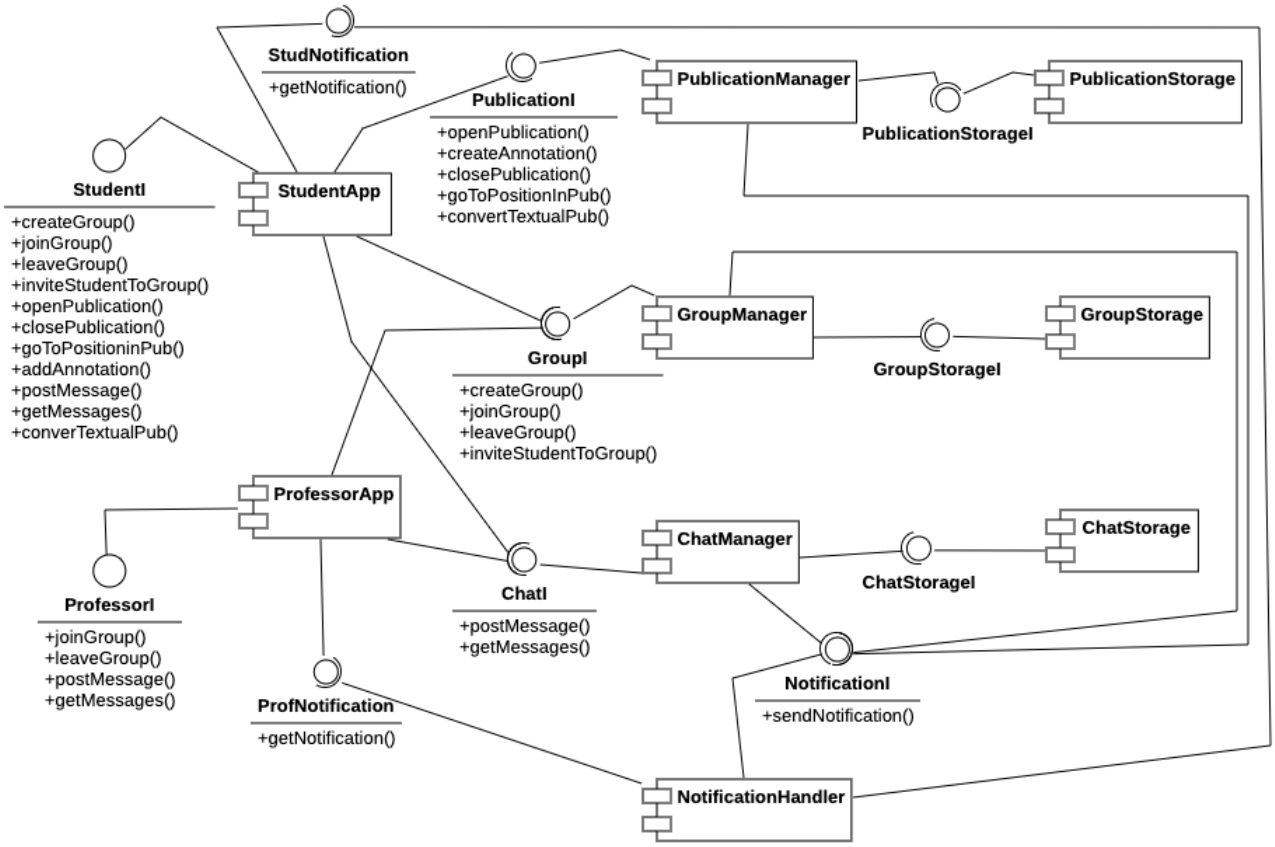
Question 2 – Integration testing (5 points)

Consider the following system description.

PubCoReader is a software system conceived to be used in courses with multiple instructors. It allows groups of students to read (or watch – for simplicity, in the following the two terms are used as synonyms) and annotate publications (books, papers, reports, videos…) in a collaborative way. **PubCoReader** should support the creation of multiple groups of readers that read the same or different publications independently one from the other. Group members collaborate online.

For instance, at a certain point in time, we can observe the following scenario, where three groups of students are using the system.
Group 1 is studying the book titled "Software Architectures" and is currently annotating Chapter 2 by complementing the book content with a picture from the lecture slides and some textual notes. Each group member sees the same book page and the identity of the group mate that has added an annotation to the book, as well as the exact position where the annotation has been added.
Group 2 is watching the video of the latest lecture in the course. The group members are discussing through the chat implemented as part of **PubCoReader** about a point that is not clear to any of them and add the following annotation to the specific point under discussion "@professor: please help us understand this point".
Group 3 has run the "translate to audio" feature on Chapter 3 of the "Software Architectures" book and is currently listening to the generated audio. Each time a group member stops the audio and inserts an annotation, this is associated with the corresponding part of text in the book.
Two professors are monitoring the students' activities. One of them notices the @professor message and temporarily joins the corresponding group to discuss the misunderstanding with them.

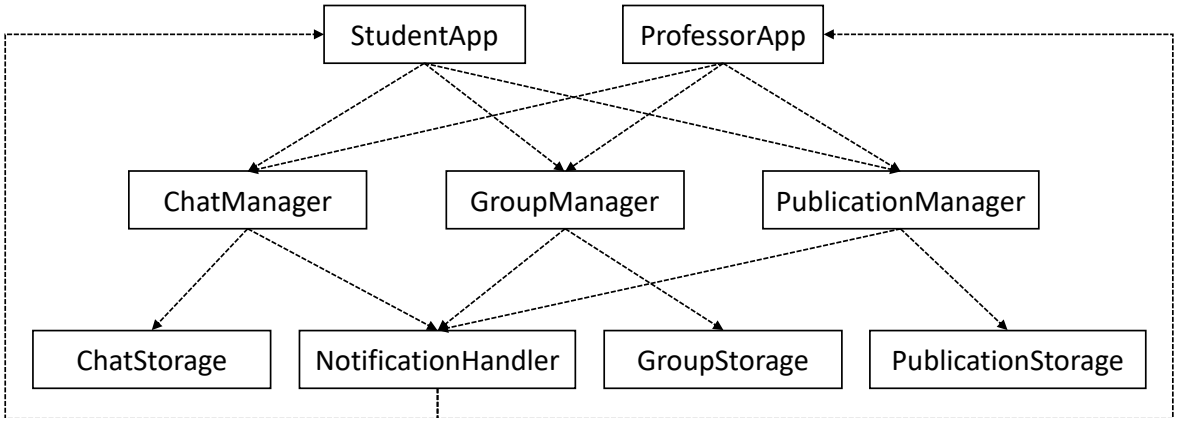
The development team has defined the following high-level architecture for PubCoReader.



The interfaces offered by *StudentApp* and *ProfessorApp* capture the functions the two applications make available to the two types of users of PubCoReader, students and professors.
Given this architecture and the operations defined by the components' interfaces, define an integration testing plan, and explain why you think it is appropriate for the case at hand.

Solution
The integration testing plan we adopt depends on the integration sequence we plan to execute. The plan can be based on one or a combination of the following strategies: bottom-up, top-down, thread-based, and critical modules.

Applying a bottom-up strategy could be the simplest approach. However, we need to consider that, in this case, the usage relationships lead to a cycle due to the fact that *NotificationManager* is using *ProfessorApp* and *StudentApp* that, for all the other features, are actually the top of the usage relationship (see the usage relationships representation in the figure below).



So, we could proceed by excluding from the initial integration iteration the implementation of the *NotificationHandler*, which could be replaced by a stub until the last step. More specifically, we could start from the integration and testing of the following elements:

- *PublicationManager* and *PublicationStorage*, using a stub for *NotificationHandler* and a driver for *PublicationManager*
- *GroupManager* and *GroupStorage*, using a stub for *NotificationHandler* and a driver for *GroupManager*
- *ChatManager* and *ChatStorage*, using a stub for *NotificationHandler* and a driver for ChatManager.

These three integration steps could be performed even in parallel if we can organize the development team in three disjoint subteams.
Then, we could proceed as follows:

- Integration of *StudentApp* with *PublicationManager*, using a driver to replace the *NotificationHandler*
- Integration of *StudentApp* with *GroupManager*, using a driver to replace the *NotificationHandler*
- Integration of *StudentApp* with *ChatManager*, using a driver to replace the *NotificationHandler*.

At this point we could focus on the integration between the *NotificationHandler* and the *StudentApp* and, finally, we could proceed with the integration and testing of the *ProfessorApp* and *PublicationManager*, *GroupManager*, and *ChatManager*, respectively, followed by the integration between the *NotificationHandler* and the *ProfessorApp*.

Question 3 – Project management (3 points)
Suppose you are the PM in charge of leading the software development in an Electric Vehicles (EV) charging station company. The company is developing an application, called *StationFinder,* to find charging stations in specific geographic locations. The company is currently in the process of making some decisions to limit the overall complexity of the project.

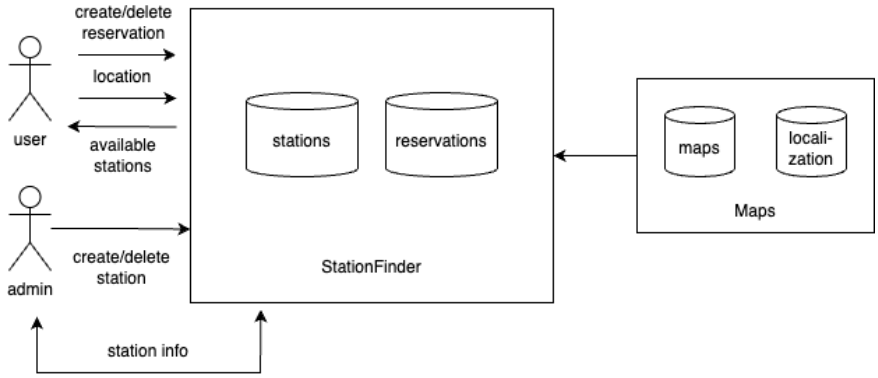
The first release of the app shall include the following features (paired with their estimated level of complexity):

- F1 (simple): user management, including login/logout operations as well as storage and inquiry of profile information;
- F2 (complex): search for locations, visualization of maps, and geographical localization;
- F3 (simple): management of data about charging stations; more specifically, administrators create/delete/view this data while users can search for available stations;

- F4 (medium): management of reservations in selected charging stations; more specifically, users create/delete reservations of available charging slots.

The development team makes the following decisions:
- F2 is implemented by an external system *Maps*;
- F3, F4 are internal features of the system *StationFinder*.

The following figure represents *StationFinder* in terms of function points elements referring to F2 (implemented by *Maps*), F3, and F4.

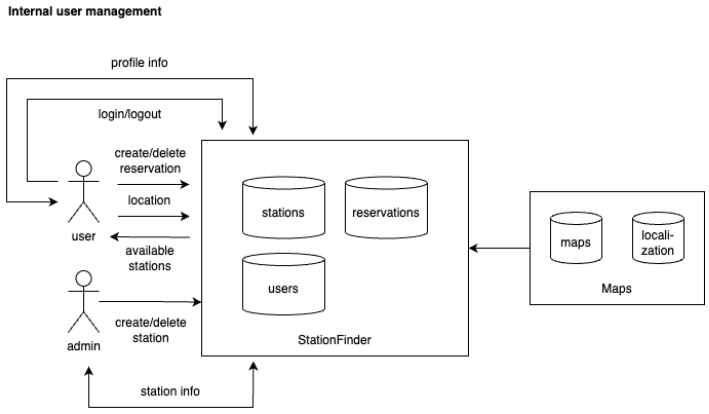


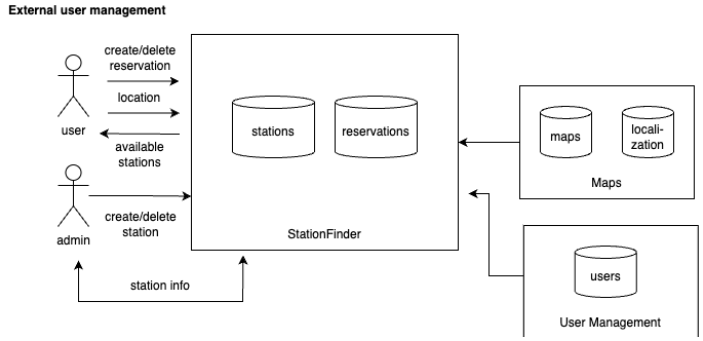
Consider the following two alternatives:
- Develop F1 as part of *StationFinder*;
- Develop F1 as an external system.

For each alternative, complete the figure above.
Then, for each alternative, estimate the total amount of function points considering the weights listed in the table below. Which alternative has the lower overall estimated complexity?

function type	simple	medium	complex
Input	3	4	6
Output	4	5	7
Inquiry	3	4	6
ILF	7	10	15
EIF	5	7	10

Solution





Internal user management:

- Input: login/logout (simple), location (medium), create/delete station (simple), create/delete reservation (medium)
- Output: available stations (simple)
- Inquiry: profile info (simple), station info (simple)
- ILF: stations (simple), users (simple), reservations (medium)
- EIF: maps (complex), localization (complex)

External user management:

- Input: location (medium), create/delete station (simple), create/delete reservation (medium)
- Output: available stations (simple)
- Inquiry: station info (simple)
- ILF: stations (simple), reservations (medium)
- EIF: maps (complex), localization (complex), users (simple)

FP estimates

Internal user management:

- Input: 4*3 + 3*4
- Output: 4
- Inquiry: 2*3
- ILF: 2*7 + 10
- EIF: 2*10
- Estimated FP: 4*3 + 3*4 + 4 + 2*3 + 2*7 + 10 + 2*10 = 78

External user management:

- Input: 2*3 + 3*4
- Output: 4
- Inquiry: 3
- ILF: 7 + 10
- EIF: 2*10 + 5

- Estimated FP: 2*3 + 3*4 + 4 + 3 + 7 + 10 + 2*10 + 5 = 67

Best choice**:** external user management