Ingineria programării

Curs 13 – 18 mai

Outline

- Course overview
- How do we use Programming Engineering techniques?
- Exam

Context

- **Large applications** (millions of lines of code) written over several months, years ...

- **Working teams**: Project managers, analysts, architects, programmers, testers, support engineers (the order of 10, 100, 1,000 people ...)

Programming Engineering

- Deals with all the aspects of developing a **large program**, by a team of developers
- **Systematic** and **organized** approach to the software development process
-
- Use of appropriate **techniques** and **tools** taking into consideration:
  - The problem to be solved
  - Restrictions
  - Available resources

Defining the Project

- Build a system that manages documents collaboratively (similar to Google Docs). The system should contain Rich Text documents and Spreadsheets. The system should allow users to:
  - Create documents and spreadsheets
  - Format documents
  - Insert formulas into a spreadsheet
  - Share documents with users
  - Edit concurrently

Development Models

- How to perform those activities specified by the stages of software development
- Examples of development models:
  - Ad hoc: manage as best you can
  - Waterfall (with feedback)
  - Prototyping
  - Spiral
  - RUP (Rational Unified Process)
  - V-Model
  - XP (Extreme Programming)
  - Agile, Lean, Scrum
  - MDD, AMDD
  - TDD

Project Phases

- Requirements engineering
- Architectural design
- Detailed design
- Implementation
- Integration
- Validation
- Verification
- Deployment
- Maintenance

Requirements Engineering

- Process of understanding customer needs and expectations regarding our app
- Which functionalities do we expect one application to have
- How should the system behave and what are its characteristics

Requirements Engineering

- Types of requirements
  - **User requirements**
  - **Functional requirements**
  - **Performance requirements**
  - **Constraints**

User Requirements

- The system is intended for managing reports and charts in a software development company
- The system must be available online
- Documents will be accessed remotely but will not be stored on local machines
- Accessed via a web browser

Functional Requirements

- Create a document/ spreadsheet
- Write and format text in a document
- Input data and formulas in a spreadsheet
- Manage document/ spreadsheet versions
- Share documents for editing or viewing
- Export documents in editable or non-editable format
- Create folder structures for managing documents

Performance Requirements

- The system should allow 100 simultaneous connections
- The maximum number of simultaneous connections on the same document is 10 users
- Available drive space must be upgradeable
- Number of folders and documents/ spreadsheets is unlimited
- Keep at most 100 major versions for each file

Constraints

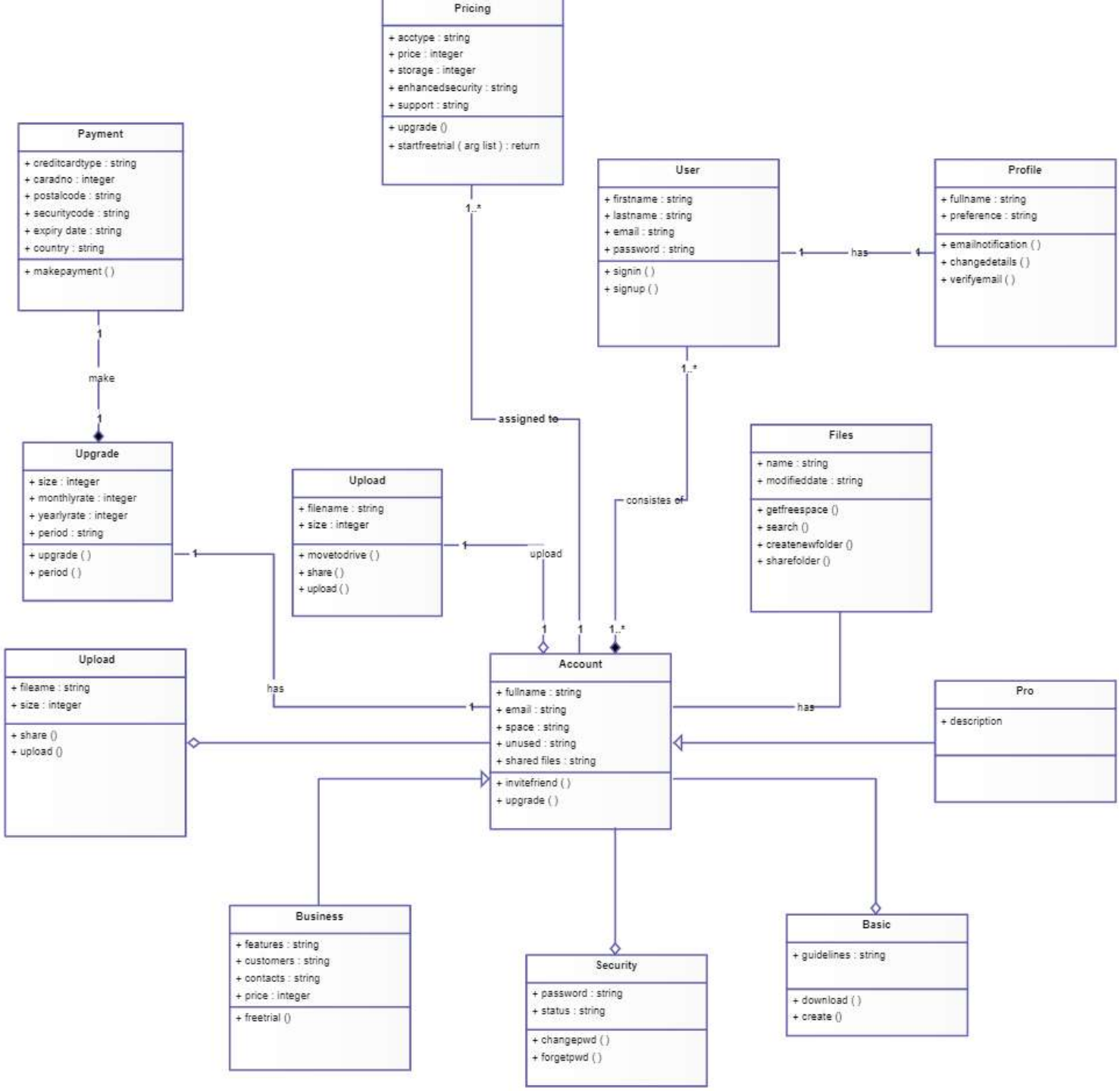
- Only logged users are permitted
- The system will only be available on the company intranet
- All user actions are logged

Actors and Roles

- Once the requirements are known, the actors and roles need to be determined
- For the exam, define the actors and roles for the given subject
- For the example
  - Logged User: wants to create, edit and share documents and spreadsheets; to organize documents in folders; to delete documents
  - Anonymous User: wants to view public documents
  - System: sends notifications regarding available drive space, shared documents, deleted documents

Architectural Design. Detailed Design

- Diagrams
  - Structure (class, package, deployment, etc.)
  - Behavioral (use case, state, activity, etc.)
  - Interaction (communication, interaction, etc.)
- For the exam, draw a class diagram for the given subject
  - The diagram for the exam might not cover all the features, since there are size constraints
- Class diagram for the given example (partial)


Pricing
+ acctype : string
+ price : integer
+ storage : integer
+ enhancedsecurity : string
+ support : string
+ upgrade ()
+ startfreetrial (arg list) : return
Payment
+ creditcardtype : string
+ caradno : integer
+ postalcode : string
+ securitycode : string
+ expiry date : string
+ country : string
+ makepayment ( )
User
+ firstname : string
+ lastname : string
+ email : string
+ password : string
+ signin ( )
+ signup ( )
Profile
+ fullname : string
+ preference : string
+ emailnotification ( )
+ changedetails ( )
+ verifyemail ( )
1..*
has
make
assigned to
consistes of
Upgrade
+ size : integer
+ monthlyrate : integer
+ yearlyrate : integer
+ period : string
+ upgrade ( )
+ period ( )
Upload
+ filename : string
+ size : integer
+ movetodrive ( )
+ share ( )
+ upload ( )
upload
Files
+ name : string
+ modifieddate : string
+ getfreespace ()
+ search ()
+ createnewfolder ()
+ sharefolder ()
Upload
+ fileame : string
+ size : integer
+ share ()
+ upload ()
has
Account
+ fullname : string
+ email : string
+ space : string
+ unused : string
+ shared files : string
+ invitefriend ( )
+ upgrade ( )
Pro
+ description
Business
+ features : string
+ customers : string
+ contacts : string
+ price : integer
+ freetrial ()
Security
+ password : string
+ status : string
+ changepwd ( )
+ forgetpwd ( )
Basic
+ guidelines : string
+ download ( )
+ create ()

Implementation

- SOLID
  - SRP – Single Responsibility Principle
  - OCP – Open/Closed Principle
  - LSP – Liskov Substitution Principle
  - ISP – Interface Segregation Principle
  - DIP – Dependency Inversion Principle
- DRY – Don't Repeat Yourself
- YAGNI – You Aren't Gonna Need It
- KISS – Keep It Simple, Stupid

Implementation

- For the exam, choose a non-trivial method from the class diagram you have drawn and fully implement it
  - For the class Files, implementation of the function shareFolder()

```
void shareFolder(ArrayList<User> users) {
    String fileId = "1sTWaJ_j7PkjzaBWtNc3IzovK5hQf21FbOw9yLeeLPNQ";
    JsonBatchCallback<Permission> callback = new JsonBatchCallback<Permission>() {
    @Override
    public void onFailure(GoogleJsonError e,  HttpHeaders responseHeaders)  throws IOException {
            // Handle error
     System.err.println(e.getMessage());
  }

     @Override
     public void onSuccess(Permission permission, HttpHeaders responseHeaders) throws
            IOException {
   System.out.println("Permission ID: " + permission.getId());
  }
};
BatchRequest batch = driveService.batch();
Permission userPermission = new Permission()
    .setType("user")
    .setRole("writer")
    .setEmailAddress("user@example.com");
driveService.permissions().create(fileId, userPermission)
    .setFields("id")
    .queue(batch, callback);}
}
```

Design Patterns

- A general repeatable solution to a commonly occurring problem in software design
- Language-independent strategies for solving common object-oriented design problems
- Types
  - Creational
  - Structural
  - Behavioral

- For the exam, select 1 appropriate DP from each category and explain
  - List of useful patterns for the given example

Creational Patterns

- Abstract Factory – not useful
- Builder
  - Used to create new files, as each files needs the following pieces to be created: file name, path, access rights
- Factory Method – not useful
- Prototype
  - Used to clone files to be then separately edited
- Singleton – not useful
- Lazy initialization – not useful
- Object pool
  - Used to manage user connections to the system. Remember the 100 concurrent user limit in the requirements
- Multiton – not useful
- Resource acquisition – not useful

Structural Patterns

- Adapter
  - Used for uploading rich text files or spreadsheets in different formats, which must then be converted to the internal format
- Bridge
  - Used for decoupling things like file content and file access rights. The access rights are available in the abstract class Files, and then inherited. The content is available in the child classes
- Composite
  - Used for the folder structure that holds the files. Folders are the containers and files are the primitives.

Structural Patterns

- Decorator
  - Used for assigning access rights. Also used for conditional formatting for spreadsheets.
- Façade
  - Provide an API for access through other methods than the web interface
- Flyweight
  - Used to encode character formatiing in the case of rich text documents in the system
- Proxy
  - Used to provide a limited access version of a file (read only, web view, etc.)

Behavioral Patterns

- Chain of Responsibility – not used
- Command
  - Used to encapsulate actions such as create (file or folder), share (file or folder), change access rights
- Interpreter
  - Used to transform formulas in spreadsheets into executable code and to provide the result of that code
- Iterator
  - Used to navigate through the file structure; also used to create custom lists for each user (shared files, recent files, large files, etc.)

Behavioral Patterns

- Mediator
  - Used for instant messaging while editing files. Also used for sending notifications regarding sharing and editing of files
- Memento
  - Used for version management, undo actions, etc.
- Observer
  - Each user can activate notifications for the account/ file with regards to receiving messages, file changes, etc. This can be done by using observers for the appropriate actions.

Behavioral Patterns

- State
  - Used for providing different types of access depending on the share status of the document (read only, can edit, can share, etc.). Each type of share status is a state.
- Strategy
  - Used for running formulas in spreadsheets. Each formula is a strategy which is solved on demand (Add, Max, Avg, etc.)
- Template Method – not used
- Visitor
  - Used for creating statistics: determine available space (visit all files in account and add their size), list shared files, list files shared with user, or custom searches such as files larger that a given size.

Software Testing

- **Professional testing** - finding the least amount of test cases which will check the most amount of system features
  - Functional
  - Non-Functional

Functional Software Testing

- Layers of testing
  - Unit testing or debugging
  - Module/Sub-System
  - Integration
  - System
  - Acceptance

Functional Software Testing

- Testing Methods
  - White Box
  - Black Box
  - Gray Box
  - Graphical user Interface Testing
  - Acceptance Testing
  - Regression Testing
- For the exam, for the method defined previously, write 3 test methods

Non-Functional Testing

- **Performance testing** or **Load Testing** checks to see if the software can handle large quantities of data or users (software scalability).
- **Usability testing** checks if the user interface is easy to use and understand.
- **Security testing** is essential for software which processes confidential data and to prevent system intrusion by hackers.
- **Internationalization and localization** is needed to test these aspects of software for which a pseudo localization method can be used.

Refactoring

- Making code cleaner, simpler and more elegant
  - Increase cohesion and decrease coupling
  - Refactored code is easier to maintain and extend
  - Simpler to use design patterns
  - Improves performance and usability of code
- The refactored program must be functionally equivalent to the initial program

Software Quality

- Software Quality
  - Safety
  - Efficiency
  - Maintenance
  - Usability

Estimation

- Estimation
  - Size of software
  - Complexity of software
  - Robustness of software
  - Amount of time required to develop some software
  - Resource allocation for development
  - Productivity of effort
  - Development costs

Exam

- 2nd of June, 8.00 to 12.00
  - One hour slots
  - Do NOT miss your time slot
- The exam will be online, one group at a time
  - Each group will have an assigned time slot and will take the exam with their lab coordinator
  - Everybody must be online in the same conference (similar to laboratories), microphone and (if possible) camera on
  - If you cannot attend because of no internet connection or any other issue, write an email to the course coordinators until 26th of May

Exam

- A theme will be given (such as the example in this course) and students will have 40 minutes to answer.
  - The answer will be written by hand on ONE sheet of A4 paper (two pages)
  - The answers expected are described here: https://profs.info.uaic.ro/~adiftene/Scoala/2020/IP/Cursuri/SablonExamenIP.pdf

Exam

- At the end of the 40 minutes make a photo for each page
  - The names of the images must follow the template YearGroupSurnameName_1.jpg (E.g. 2A8PopescuIon_1.jpg, 2A8PopescuIon_2.jpg)
  - Upload your images to the given link until the hour is over.
- During week 14 (25 – 31 May) there will be a simulated exam to rehearse the procedure
  - It will take place during the laboratory period

Bibliography

- https://creately.com/diagram/example/hy47yb2e2/Google+Drive?utm_source=pinterest&utm_medium=social&utm_campaign=pinclass