[15 minutes]

a. [10 points]
Explain clearly what aspect of a relational database is the reason why it is difficult to scale out a relational database

Because a relational database is strict consistent and it is very difficult to scale out a relational database and be still available and strict consistent. It is very difficult to scale out writes to the database.

b. [10 points]
Explain clearly the consequence of scaling out a No-SQL database. In other words, what is the price you pay for the ability to scale out No-SQL databases?

The consequence is that we have eventual consistency with a No-SQL database

[15 minutes]

Describe clearly why it is not good to use an **anemic domain model**

-You do not use the powerful OO techniques to organize complex logic.
-Business logic (rules) is hard to find, understand, reuse, modify.
-The software reflects the data structure of the
business, but not the behavioral organization
-The service classes become too complex
 -No single responsibility
 -No separation of concern

[15 minutes]

In Domain Driven Design we learned to divide our domain into subdomain types.

a. [10 points]

Give the name of the different subdomain types we learned and describe in one sentence the characteristic(s) of these subdomain types.

- Core subdomain: This is the reason you are writing the software.
- Supporting subdomain: Supports the core domain
- Generic subdomain: Very generic functionality(Email sending service, Creating reports service)

b. [10 points]

Explain clearly why it is important to identify these subdomain types.

If you know these subdomains, you know

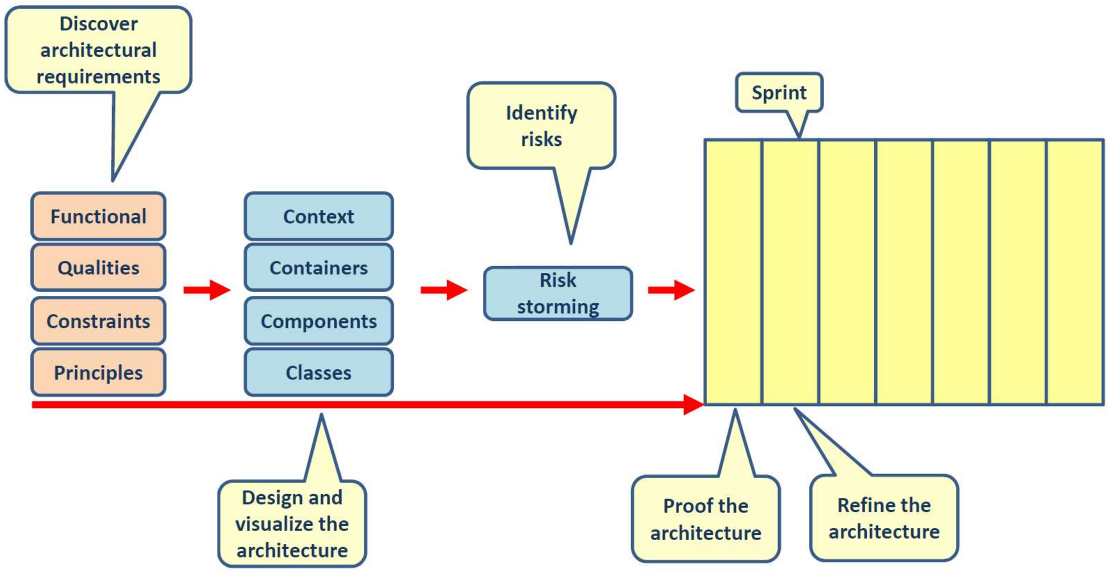
- On which part of the application should I put the focus
- Where to put your most experienced developers
- What code should be of the highest quality
- Where to apply DDD

[15 minutes]

The question is given in the attachment. **Open the PDF file in the browser.**

Type your answer in the editor on this page

Attachments
question 1.pdf



[30 minutes]

Suppose we need to design a **course scheduling application** for the Computer Science department at MIU using Components and DDD. This application has the following requirements:

The system keeps track of all courses (coursenumber, course name, course description) given by the department. The system also keeps track of prerequisites for certain courses.
The system also keeps track of all students in the department. For every student we need to know the studentnumber, name, email, phone and on-campus postbox number. For every student we also keep track of the grades this student got for the courses he/she took.
The system also keeps track of all professors in the department. For every professor we need to know the name, email, phone and on-campus postbox number. For every professor we also need to know which courses this professor can teach.
Every course will be taught a few times a year. Students can sign-up for certain course offerings. For every course offering the system keeps track of the students in the course, the professor who teaches it, the location where the course is given (building name, room number) and the start and end date of the course
If a student signs up for a course offering, the system will automatically check if this student has the required prerequisites for this particular course. If the student has not the required prerequisites then the student cannot sign up for this course. If the student has successfully signed up for a course, the system will send an email to this student.
All data will be stored in the database.
Components talk to each other using REST webservices.
You have to design this system using the best practices we learned in the course. You have to use component based design.
For every component give the following details:
- Component name
- Layers used within the component
- ALL classes used in each layer
- For the domain classes, indicate what type of class it is according to DDD (entity, value object,...)

Your answer should look like the following example:

Component A
Layer 1: Class K
Layer 2: Class L
Layer 3: Class M (Entity), Class N (Value object)
Layer 4: Class O
Component B
Layer 1: Class P
Layer 2: Class Q
Layer 3: Class R (Entity), Class S (Value object)
Layer 4: Class T

Do NOT specify class attributes or methods.

Component: Courses

web layer:CourseController
service layer: CourseService, CourseDTO
domain layer: Course(Entity)
data access layer: CourseDAO
integration layer:

Component: Students
web layer:StudentController
service layer: StudentService, StudentDTO
domain layer: Student(Entity), ContactInfo(Value object), CourseGrade(Value object)
data access layer: StudentDAO
integration layer:

Component: Professors
web layer: ProfessorController
service layer: ProfessorService, ProfessorDTO
domain layer: Professor(Entity), ContactInfo(Value object), Course(Value object)
data access layer: ProfessorDAO
integration layer:

Component: CourseOfferings
web layer: CourseOfferingController
service layer: CourseOfferingService, CourseOfferingDTO, ...(and other DTO's)
domain layer:CourseOffering(Entity), Student(Value object), Course(Value object), Professor(Value object), Location(Value object)
data access layer: CourseOfferingDAO
integration layer: EmailSender, CoursesGateway

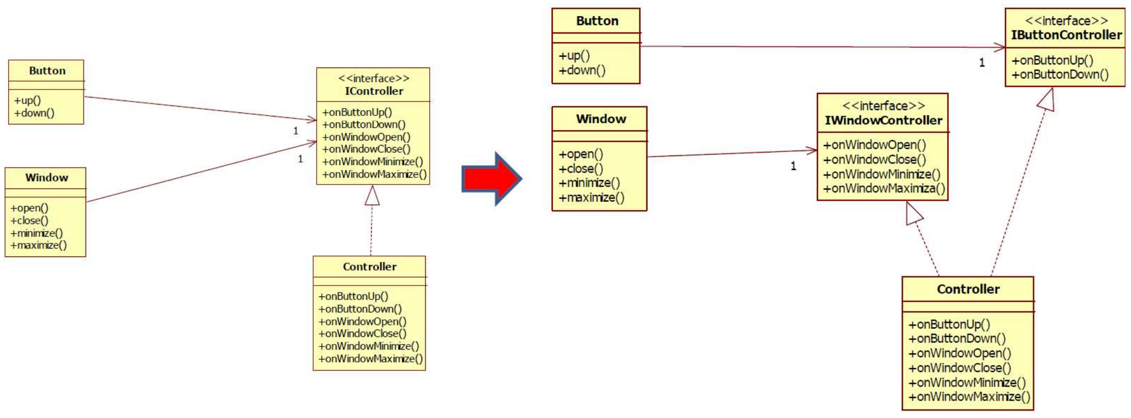
[15 minutes]

Explain what we mean with the **Interface Segregation Principle**.

Write a **clear example** that does **not** use this principle.

Write a **clear example** that does use this principle.

- Clients should not be forced to depend on methods (and data) they do not use


Button
+up()
+down()
Window
+open()
+close()
+minimize()
+maximize()
<<interface>>
IController
+onButtonUp()
+onButtonDown()
+onWindowOpen()
+onWindowClose()
+onWindowMinimize()
+onWindowMaximize()
1
1
Controller
+onButtonUp()
+onButtonDown()
+onWindowOpen()
+onWindowClose()
+onWindowMinimize()
+onWindowMaximize()
Button
+up()
+down()
Window
+open()
+close()
+minimize()
+maximize()
<<interface>>
IButtonController
+onButtonUp()
+onButtonDown()
<<interface>>
IWindowController
+onWindowOpen()
+onWindowClose()
+onWindowMinimize()
+onWindowMaximiza()
1
1
Controller
+onButtonUp()
+onButtonDown()
+onWindowOpen()
+onWindowClose()
+onWindowMinimize()
+onWindowMaximize()