Exercise 1 counts 25%

Which is the relationship between the ”usability” quality attribute and the software architecture of a system. Give your definition of “usability” and of “software architecture”. Motivate with examples, preferably from the eCourse project.

Oppgave 1, 25%

Gi en definisjon av “brukervennlighet?” (usability) og “programvare arkitektur” (Software architecture”. Beskriv også eksempler på begge, gjerne fra eCoruse prosjektet.

Diskuter relasjonen mellom kvalitetsattributtet brukervennlighet og programvare arkitekturen for et system, angi ogsaa her eksempler, gjerne fra eCourse.

Solution

We define usability as: The capability of the software to be understood, learned, used and liked by the user, when used under specified conditions. Usability can be broken down into:

- Understandability: The capability of the software product to enable user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use. This will depend on the documentation and initial impressions given by the software.
- Learnability: The capability of the software product to enable the user to learn its application.
- Operability: The capability of the software product to enable the user to operate and control it.
- Likeability: The capability of the software product to be liked by the user.

We define the software architecture of a system as the structure of the system, in terms of components, connectors, and their visible interfaces.

The usability of a system depends on the functionality of the system, i.e., if the system provides the right functionalities, the user will like it, and he will be able to operate and control it. Understandability, and learnability strongly depends on the user interface. The user interface can be regarded as an architectural component so changes to this components should not influence the global architecture. We can conclude that there are weak relationships between usability and architecture as opposite to other quality attributes, such for examples performance and reuse.

As an example we can consider the usability work done in the context of the eCourse project. Students propose both a new functionality (nested categories) and some presentation level changes. The design of the introduction of the new functionality does not impact the architecture as no new class is introduced but only new methods. The changes to the presentations are mainly changes to jsp pages which are low level design/implementation level changes.

If a software system, which has been conceived for a device type, e.g., a WEB system conceived for PC, has to be ported on other device based applications, e.g., WAP, so the architecture of the system may have to be revised.

Exercise 2 counts 25%

Consider the following quality attributes: maintainability, performance, reuse, and security. Which of these attributes are observable at run time? Explain why a good software architecture description is crucial to achieve the quality attributes, which are not observable at run time, and explain why.

Oppgave 2, 25%
Oppgaven gjelder följende fire kvalitetsattributter: vedlikeholdbarhet (maintainability), ytelse (performance), gjenbruk (reuse) og sikkerhet (security).
Hvilke av disse kan observeres mens systemet kjöres?
Angi hvordan de fire forskjellige kvalitetsattributtene kan observeres, og hvis mulig hvilke kvantitative mål man kan gi for hver av dem.
Svar: Maintainability = kroner/KLOC/year i vedlikeholdskostnad (feilretting)

Ytelse = ytelses tester, dvs. Tester under realistisk og höy last. Man kan da sjekke f.eks. antall parallelle brukere, transaksjoner per sekund, svarstider, etc.

Gjenbruk = antall moduler gjenbrukt I andre systemer (med eller uten modifikasjoner)

Sikkerhet = antall timer för 10 innleide hackere har brutt seg inn I systemet ☺

Forklar hvorfor en god beskrivelse av programvarearkitekturen er viktig for å tilfredstille de kvalitetsatributtene som ikke er observerbare når systemet kjöres.

Solution 2

Performance and security are observable at run time.
Maintainability and reuse are not observable at run time but depends from the static descriptions of the system and their traceability.
Reuse and maintainability depend from goodness of the structure of design and code, e.g., the software architecture of the system.

Exercise 3 counts 50%

Consider the following fragment of a java servlet specification, which is extracted from eCourse source code.
Fölgende fragmet fra en java servlet spesifikasjon er gitt (ekstrahert fra eCourse kildekoden):

```
public abstract class FrontServlet extends HttpServlet {
	protected ControllerManager controllerManager;
	protected EventGenerator eventGenerator;
	private String tempFileDirectory = "c:/temp/";//NB!
	private int maxFileSize = 10;


	public void init() throws ServletException {
		controllerManager = initControllerManager();
		eventGenerator = initEventGenerator();
	}



	public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		handleRequest(request, response);
	}



	public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		PostRequestParser postRequestParser = new
PostRequestParser(tempFileDirectory, maxFileSize);
		postRequestParser.parametersToAttributes(request);
		handleRequest(request, response);
	}



	protected void handleRequest(HttpServletRequest request,
HttpServletResponse response) {
		try {
			HttpSession session = request.getSession(true);

			EApplicationEvent event =
eventGenerator.generateEvent(request);
			String nextUrl = sendEventToController(event, session);
			updateSession(session,event);
			sendResponse(nextUrl, request, response);
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}



	protected String sendEventToController(EApplicationEvent event,
HttpSession session) throws Exception {
```

```
            return
controllerManager.getController(session).handleEvent(event);
      }



      protected void sendResponse(String nextUrl, HttpServletRequest
request, HttpServletResponse response) throws Exception {
            request.getRequestDispatcher(nextUrl).forward(request,
response);
      }


      protected void updateSession(HttpSession session, EApplicationEvent
event) {}



      protected abstract EventGenerator initEventGenerator();

      protected abstract ControllerManager initControllerManager();


}
```

Answer to the following questions and motivate your answers.
Besvar følgende spörsmål, og motiver dine svar:

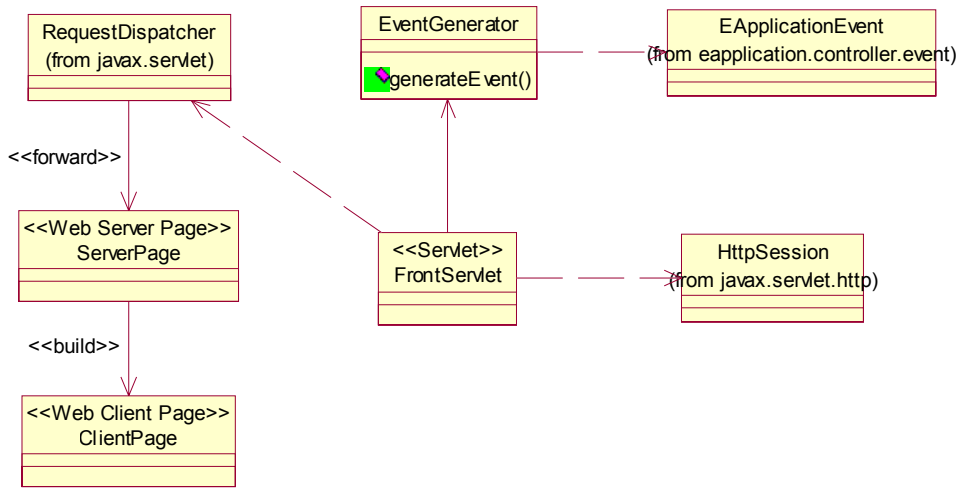
1. ECourse is organized according to the Model View Controller architectural pattern. Is FrontServlet part of the Model, View, or Controller?
   Ecourse er organisert etter model view controller arkitektur mönstret. Er FrontServlet en del av model, view eller controller?
2. Is FrontServlet an abstract or concrete class? Which is the purpose of this class?
   Er FrontServlet en abstrakt eller konkret klasse? Hva er oppgave til denne klassen?
3. Reconstruct (reverse engineering) the UML diagram which describe the connections between FrontServlet and other classes in the system. Describe only the connections and the classes which can be extracted from this code fragment as you did not have access to eCourse documentation. For each of the classes above, describe if it is part of the Model, View, or Controller part.
   Gjenskap (reverse engineer) UML diagrammet som beskriver sammenhengen mellom FrontServlet og de andre klassene I koden over. For hver av klassene beskriv om de hörer til model, view eller controller.
4. FrontServlet is designed according to the Façade pattern. Describe the advantages and disadvantages of this pattern and give examples preferably from the eCourse system.
   FrontServlet er konstruert ut fra Façade mönsteret. Beskriv fordelene og ulempene med dette mönsteret, og angi eksempler, helst fra eCourse systemet.

Solution 3

1. FrontServlet is part of the View. FrontServlet is a Servlet that acts a front component for the internet application.
2. FrontServlet is an abstract class intended to be subclassed by application-specific classes, one subclass for each client-type (web, WAP, etc.). The abstract FrontServlet is responsible of receiving requests from clients, transforming them to generic events, sending the events to the controller and returning response back to the client. The only task that is necessary for the subclasses to do, is to initialise the EventGenerator and

ControllerManager to be used. The *FrontServlet* is the *front component* in the In the eCourse application there are two subclasses - one for the WAP interface and one for the Web interface. In *FrontServlet* forwards to *ServerPages* (JSP pages). In addition it communicates with the controller through the *ControllerInterface*,

3. The central component of the view is the *FrontServlet*.



The *FrontServlet* uses the *EventGenerator* that generates an *EApplicationEvent* based on the request the *FrontServlet* received from the browser.
EApplicationEvent is not a part of the view, but it is shown in diagram to give a better perception of the view.
The *HttpSession* stores a reference to the controller, which the *FrontServlet* needs when it sends the events.
RequestDispatcher is used by the *FrontServlet* in order to forward client requests to a *ServerPage*.
A *ServerPage* is a JSP page that builds a *ClientPage*.
ClientPage represents the information that is shown to the user in either HTML or WML format. To be completed. Note that the diagram is a superset since not all relationships can be deduced by the class.

4. The façade pattern should improve the maintenability and security of a system. It can introduce performance problems since the component can become a bottle neck in the system.

maintenability
The *front component* decreases the dependencies in the system since all the client pages requests are transmitted to the *front component*. The dependencies are therefore implicit since they go through the *front component*. This makes it easier to add functionality and new pages to the system. With a lot of dependencies between pages and between the pages and the rest of the system, modifications can lead to unexpected errors in the system. It also leads to a more logical separation of responsibilities. The JSP pages deal only with the presentation and do not need to implement how the requests are handled. This could have been done in several other ways too, without the use of a front component. It is a danger however that the *front component* gets too big and complicated resulting in decreased modifiability. The *front component* uses helper classes to distribute the complexity and making it easier to modify and maintain.

security

The *front component* is one of the key elements in the security architecture. It represents a single point of entry for the system, and therefore makes it easy to enforce security policies. The eCourse system architecture forces all request to go through the *front component*. The request eventually leads to a response from the *front component* that opens a new page. A page in the system cannot be opened unless it has been redirected from the front servlet. This means that we have full control of how pages in the system are accessed.