Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto, Raffaela Mirandola

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

Software Engineering II

February 10th 2015

Last Name	
First Name	
Id number (Matricola)	

Note

1. The exam is not valid if you don't fill in the above data.
2. Write your answers on these pages. Extra sheets will be ignored. You may use a pencil.
3. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
4. You cannot keep a copy of the exam when you leave the room.

Question 1 Alloy (7 points)
The operating system kernel of a multiprocessor machine (with M CPUs) organizes the processes in three disjoint sets:

- RUNNING: the processes that are being executed by some CPU.
- READY: the processes that are ready to be executed.
- WAITING: the processes that are waiting for some event from some peripheral device.

Answer to the following points:

A) Model in Alloy this kernel defining proper signatures and facts. Assume that processes are modeled through this signature:

   sig Process { }

B) Model the operation Resurrect that moves a process from WAITING to READY

C) Model also the following constraint:
   The kernel should not waste computational resources, that is, if there are READY processes, there should not be any available CPUs (that is, a CPU that is not executing any process).

Solution

```
sig Process { }

sig CPU {
  busy: lone Process
}

sig Kernel {
  cpus: set CPU,
  ready: set Process,
  waiting: set Process,
  running: set Process
}
{
  no (ready & waiting)
  no (ready & running)
  no (waiting & running)
  all c: CPU | c in cpus implies (c.busy = none or c.busy in running)
  all p: Process | p in running implies (one c: CPU | c in cpus and p in c.busy)
}

fact {
  all disjoint k1, k2: Kernel | k1.ready+k1.waiting+k1.running & k2.ready+k2.waiting+k2.running =
none
}

fact doNotWasteCPUs {
  all k: Kernel | k.ready!=none implies no c: CPU | c in k.cpus and c.busy = none
}
```

```
pred Resurrect(k, k': Kernel, p: Process) {
  //precondition
  p in k.waiting

  //postcondition
  k'.cpus = k.cpus
  k'.ready = k.ready + p
  k'.waiting = k.waiting - p
  k'.running = k.running
}

pred show[] {}

run show
```

Questions 3, 4 and 5, Planning (5 points), Design (5 points) and Testing (5 points)
Design a simple project management tool for a company. The application should:

1. Handle information about projects (name, purpose, budget, tasks) and people (name, skills, employee number, role, department, allocated projects, allocated tasks).
2. Enable people to log-in/out.
3. Allow people with role "project manager" to allocate other people to projects and tasks.
4. Allow people to visualize the tasks they are working on and input the level of accomplishment of these tasks.
5. Compute the set of people allocated to a specific project/task.
6. Support people in requesting help across the company based on skills (the user selects a set of skills and the system suggests people with these skills).

A) Estimate the effort M of the project defined above, using a combination of Function Points and COCOMO II, assuming that the application is going to be developed in JAVA (1 FP = 53 SLOC) and that all COCOMO II cost and scale drivers are nominal. To apply Function Points you can use the table below:

Function Types	**Weights**		
	Simple	Medium	Complex
N. Inputs	3	4	6
N.Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. ELF	5	7	10

To apply COCOMO II use the effort equation: 2.97 * EAI * $(KSLOCs)^E$

You do not need to compute the COCOMO II formula completely but only show that you know how to use it.

B) Define:

- The Actors involved in the application
- The Use Cases describing the main functionality of the application: it is not necessary to detail all the use cases. Rather, draw them in one or more use case diagrams and provide the details of the most important one.
- A diagram highlighting the key components of the system architecture and the way they are connected. Explain the role of each component.

C) Identify possible acceptance test cases for the system focusing on functional aspects. While defining these cases specify the preconditions (if any) that should be true before executing the tests, the inputs provided during the test, and the expected outputs. Finally, explain why you think the selected tests are important for the specific system being considered.

Solution (Part A)

First, the total amount of FPs for the problem at hand should be defined. Our problem is the following:

Design a simple project management tool for a company. The application should:
1. *Handle information about projects (name, purpose, budget, tasks) and people (name, skills, employee number, role, department, allocated projects, allocated tasks).*
2. *Enable people to log-in/out.*
3. *Allow people with role "project manager" to allocate other people to projects and tasks.*
4. *Allow people to visualize the tasks they are working on and input the level of accomplishment of these tasks.*
5. *Compute the set of people allocated to a specific project/task.*
6. *Support people in requesting help across the company based on skills (the user selects a set of skills and the system suggests people with these skills.*

ILFs: 3 (people and projects, ALL SIMPLE STRUCTURE) 2*7 = **14**
ELFs: 0 (application does employ external logical files)
EIs: 5 (login/logout, task update, project tasks allocation, task summary, i.e., the set of people allocated to a specific task. ALL MEDIUM STRUCTURE since some functions involve 2+ entities) 6*4 = **24**
EIQs: 1 (help request, MEDIUM STRUCTURE) 1*4 = **4**
EOs: 0 (application does not allow output)
TOTAL: **14+24+4 = 42**

assuming 0.053 KLOC per each FP = 42*0.053= 2.226 KSLOC for COCOMO II

COCOMO II Effort Equation = $2.94 \cdot EAF \cdot (KSLOC)^{E}$

$$E = 2.94 \cdot 1 \cdot (2.226)^{1{,}0997}$$
$$E = \mathbf{7{,}087\ PM}$$

Solution (Part B)

B.1)
Actors Involved are:
Project Manager (PM), responsible for project and task artifacts with all connected relations
Person/Developer, responsible for his/her own tasks and their progress status
Helper (anyone), responsible for providing help based on owned skills/expertise

B.2)
Use-Cases:
1. Project staff allocation
2. Task status visualization and update
3. Login-logout
4. Development network computation
5. Look for technical support / help

Arguably, the most important use-case is Look for technical support / help:

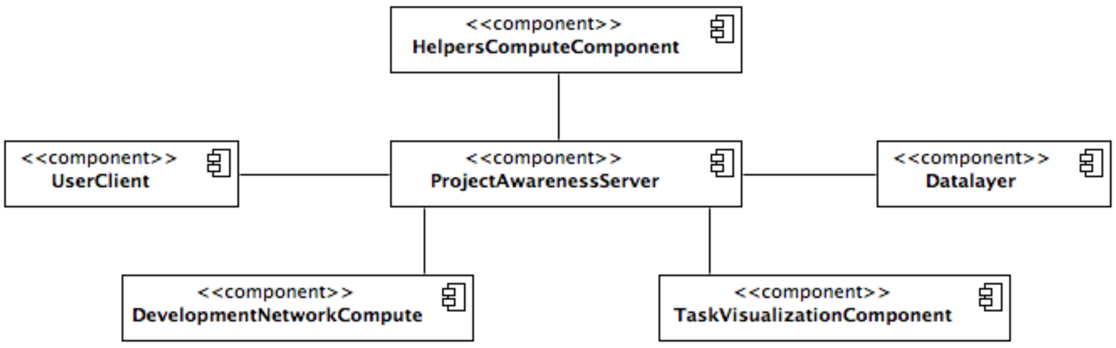
Use case name: Look for technical support
Participating actors: developer
Entry condition: none
Flow of events
- The developer polls the system to get suggestions on possible competent helping-hands
- The system evaluates tasks and completion of remaining company members to identify people who have the skills and some spare time to help.

- The system offers to the developer a list of people he/she could contact to get help.

Exit condition
The use case terminates when the system has computed the list of people. This list can be empty is none of the company members fulfills the requirements in terms of skills and free time.
Exceptions: none

B.3)

Architecture:



<<UserClient>> is used by all development network members and provides interfaces on a role basis. Project managers will be able (and will have interfaces) to create and manage projects, dividing work into tasks and allocating said tasks to other developers. Managers may also compute people working on projects/tasks to evaluate progress. Developers will be able to evaluate their status, check for other status reports, check for help-wanted requests, check their own progress and update their status.

<<ProjectAwarenessServer>> the server handles most of the critical functionality and instruments a DBMS for data pervasiveness, i.e., the <<Datalayer>> component.

The awareness server uses additional components to handle business logic, namely:

<<HelpersComputeComponent>> keeps track of status updates and evaluates people who might be available for helping colleagues;

<<DevelopmentNetworkCompute>> evaluates dependencies and allocations between members to compute a live software development network;

<<TaskVisualizationComponent>> keeps track of tasks and allows for their visualization and update.

Solution Part C

The exercise is about identifying possible acceptance test cases
- focusing on functional aspects.
- specify the preconditions (if any) that should be true before executing the tests,
- the inputs provided during the test,
- and the expected outputs.

Finally, to explain why you think the selected tests are important for the specific system being considered.

Given the most critical use-cases specified as part of exercise 3.B, it is reasonable to assume that those same use-cases should become scenarios to derive acceptance tests by means of black-box testing. This means that every of the following use-cases should be formulated into an acceptance test specification:

1. **Project staff allocation:**
   - PRECOND – at least one project is inserted and at least one task is ready to be allocated to at least one developer; also, at least one project manager is allocated to the project; finally, the project should be directly related to the project manager following an "instantiation" relation, i.e. the project manager created the project himself and has "ownership" rights;
   - INPUTS – project manager inputs the name of the person or skill needed for the task and should be able to specify the allocation;
   - OUTPUTS – system returns new development network structure that takes the allocation into account;
2. **Task status visualization and update**
   - PRECOND: at least one developer is working on a task X to which he was allocated; system is currently showing old status or <no-status> for task X;
   - INPUTS: task X.ID is used to retrieve the status related to the task and the person responsible for it, if the person coincides with the requestor then modification should be permitted;
   - OUTPUTS: a new task status should be presented via the system;
3. **Login-logout**
   - PRECOND: system is live; security check component is live;
   - INPUTS: employee credentials;
   - OUTPUTS: visualization of employee tasks and task stati;
4. **Development network computation**
   - PRECOND: there is at least one project with at least one task and at least one task allocation per tasks present;
   - INPUTS: employee credentials; development network request from a user recognized as "project manager";
   - OUTPUTS: graph representation of task allocation;
5. **Technical support / help**
   - PRECOND: task-allocations have been done, project is started (i.e., task status updates are present and progress can be computed)
   - INPUT: "need-help-on-task" request;
   - OUTPUT: list of possible candidates for lending a helping hand, matched based on skills and current availability;

These tests are critical since they ensure that key functionalities of the core business logic to be exhibited by the system are actually taken care of. User-acceptance tests performed around said functionalities will make sure that the system performs its intended function in the way the user expects it to. Also, in light of the fact that a good test is a test that fails, tests rotating around the functionalities above will reveal key shortcomings in the design and implementation of the system-to-be.