24

CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DIT 345	Fundamentals of software architecture			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
DIT345-0009-DGP		01.11.2024	17	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	18	
2	X	25	
3	X	28	
4	X	12	
5	X	9.5	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen	92.5		

[redacted]

5

CHALMERS	Anonymous code	Points for question (to be filled in by teacher) 18	Consecutive page no. Löpande sid nr 1
	Anonym kod DIT345-0009-PGP	Poäng på uppgiften (ifylles av lärare)	Question no. Uppgift nr 1

P1

P1.A)

Architectually significant requirements are those requirements ~~that~~ that have both a high mission or business value (those requirements that are important to the stakeholders) and high architectural impact (those requirements that ~~if~~ influence the system architecture decisions)

P1.B)

Assuming that a business/product owner wants a highly performant software that is meant to be built once and not changed later.

Non - architectually significant requirement might be;

A system shall be extended with additional sub-components without affecting other, existing components in the system.

This requirement is not architectually significant because client ~~prior~~ prioritizes performance of the system over any other quality attributes and the requirement is talking about maintainability of the system.

5

P1. c)

Assuming that a product owner wants to have a highly agile system that can laverage DevOps style of development (deploying/realising every 2 weeks)

2+

An architecturally significant requirement will be:
A system shall allow developers (4 people) to add a new functionality/requirement to the system within 3 working days without affecting existing/other components.

This requirement is ~~significantly~~ architecturally significant since it aligns with the quality attribute important for the stakeholder (~~maintainability~~ modifiability) and has a high architectural impact (since it kind-of suggests an architectural style that would allow to achieve the requirement)

P1. D)

a) ~~fault~~ error → fault
b) ~~error~~ fault → ~~Error~~ failure OP
c) failure → Error

5

CHALMERS

Anonymous code / Anonym kod: DIT345-0008-PGP

Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare)

Consecutive page no. / Löpande sid nr: 3

Question no. / Uppgift nr: 1

P1. E)

I (3)

II (2)

III (6)

IV (1)

V (4)

VI ~~(1)~~ (5)

12p

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 4
	Anonym kod DIT345-0009-PGP	Poäng på uppgiften (ifylles av lärare)	Question no. Uppgift nr 2

P2

P2.A)

8/8

1. Encapsulation ✓ information hiding ☺ - assuming that both classes have private attributes, the encapsulation principle is not violated since only particular attributes are exposed to the „real world" through getters 2

2. Cohesion - the cohesion is not that great in both classes. the Account class has a „hasPaidTax" method which probably should ✓ be on ~~the Person~~ a different class and the Person class has a „getAccountBalance" which should be on ~~the Account~~ a different class instead. 2

3. Coupling - there are only two classes, but the coupling between them is ~~relatively low~~ present since ~~only~~ the Person class has to know about the Account class. (using an interface would make coupling very low ~~though~~) 2

4. Single responsibility - looking at the class methods they are following the single responsibility principle since they are very simple and focused only on a single task/functionality.

5. ~~The ge~~ Separation of concerns - separation of concerns in the system is not high. For example, the Person class deals with taxes and accounts details whom it should be only concerned with its own details and functionality. 2

CHALMERS | Anonymous code: DIT345-0009-PGP | Points for question (to be filled in by teacher) | Consecutive page no. Löpande sid nr: 5 | Question no. Uppgift nr: 2

P2. B)

8/8

a)

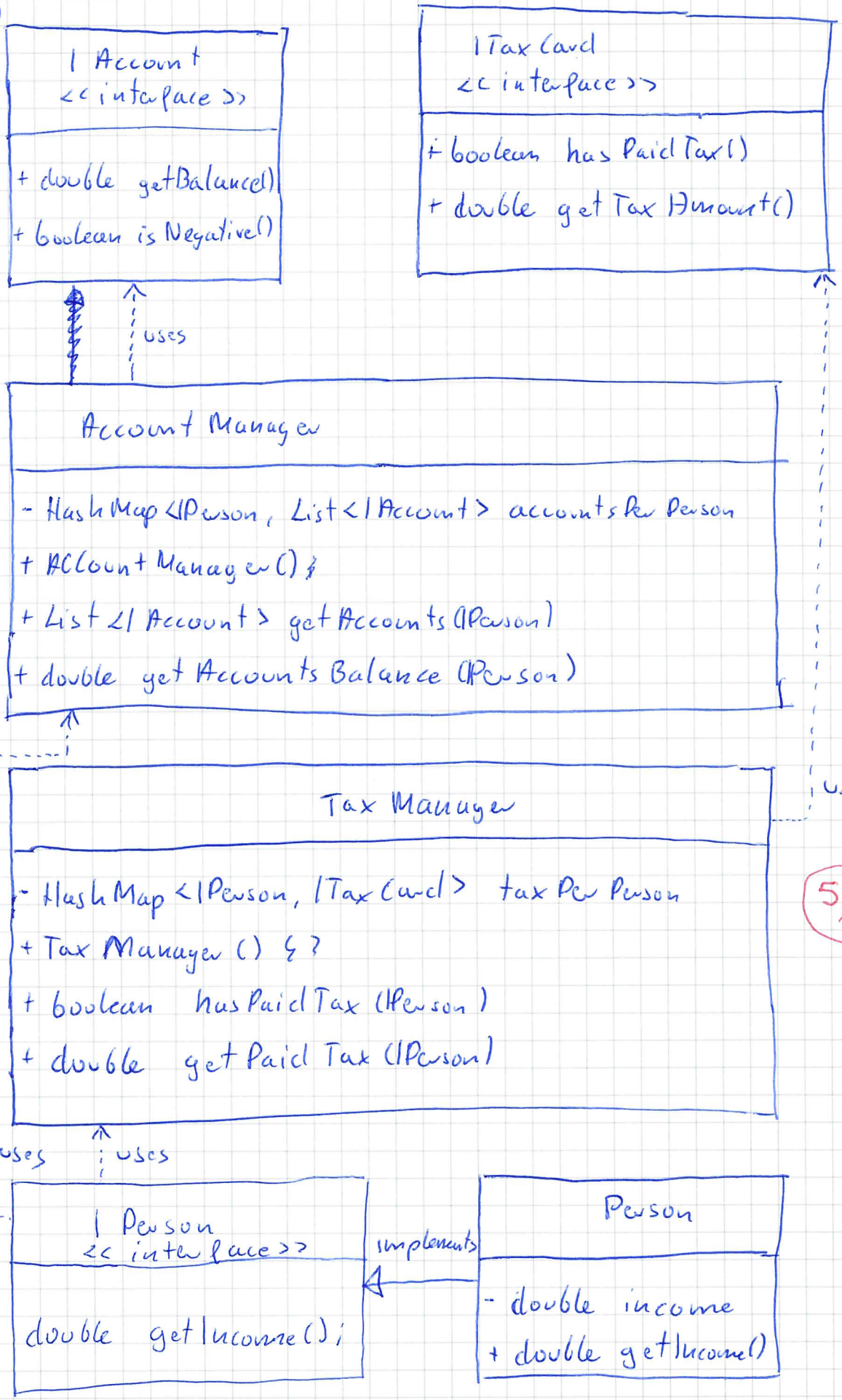
```
public interface IAccount {
    double getBalance();
    boolean isNegative();
}

public interface ITaxCard {
    boolean hasPaidTax();
    double getTaxAmount();
}

public class AccountManager {
    private HashMap<IPerson, List<IAccount>> accountsPerPerson;

    AccountManager() {}

    public List<IAccount> getAccounts(IPerson) {}

    public double getAccountsBalance(IPerson) {}
}

public class TaxManager {
    private HashMap<IPerson, ITaxCard> taxPerPerson;

    TaxManager() {}

    public boolean hasPaidTax(IPerson) {}

    public double getPaidTax(IPerson) {}
}
```

a) continue

```
public interface IPerson {
  double getIncome();

}

public Person implements IPerson {
  private double income;

  Person() {}

  public double getIncome() {}

}
```

5

6)


IAccount
<<interface>>
+ double getBalance()
+ boolean isNegative()
ITaxCard
<<interface>>
+ boolean hasPaidTax()
+ double getTaxAmount()
uses
AccountManager
- HashMap<IPerson, List<IAccount> accountsPerPerson
+ AccountManager()
+ List<IAccount> getAccounts(IPerson)
+ double getAccountsBalance(IPerson)
uses
TaxManager
- HashMap<IPerson, ITaxCard> taxPerPerson
+ TaxManager()
+ boolean hasPaidTax(IPerson)
+ double getPaidTax(IPerson)
uses
uses
IPerson
<<interface>>
double getIncome();
implements
Person
- double income
+ double getIncome()

5/5

c)

With the new version, I tried to increase the general cohesion ~~and~~ of the system. Therefore, I split the code further an introduced an Account-Manager and TaxManager classes which would deal with details and information in their specific domains. This allowed me to remove/move behavior from the Account and Person classes to the new manager classes. and I keep high separation of concerns in the system across components.

I also introduced a bunch of interfaces to decouple the system as much as possible and make it more flexible and less dependent on the actual implementations. Nice job!

I kept the encapsulation high in each class instance by using private attributes and only exposing what is needed.

Lastly, on the method level, I ~~did not~~ kept the single ~~responsi~~ responsibility high, with methods dealing with one functionality and having only a single reason to change.

4/4

5

P3

P3.A1

I think that the system would benefit from combination of two styles - client-server and microservices.

Based on the description of the system, the selected system architecture should be domain driven with many services that could act independently of one another. Microservices provide that high independence, flexibility, modularity and reliability for a system that requires different functionalities.

Since there are two customer groups (sellers and buyers) we could use client-~~and~~ server architecture to provide different presentation layers to both groups while keeping high separation of concerns between the presentation and backend (business logic, persistence, db layers) sides of the system utilizing an API layer for communication.

An architecture style that could be used instead of microservices architecture could be service-oriented architecture which would also play nicely with separating different functionalities of the system into services.

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 10
	Anonym kod DIT345-0009-PLP	Poäng på uppgiften (ifylles av lärare)	Question no. Uppgift nr 3

P3. A) continue

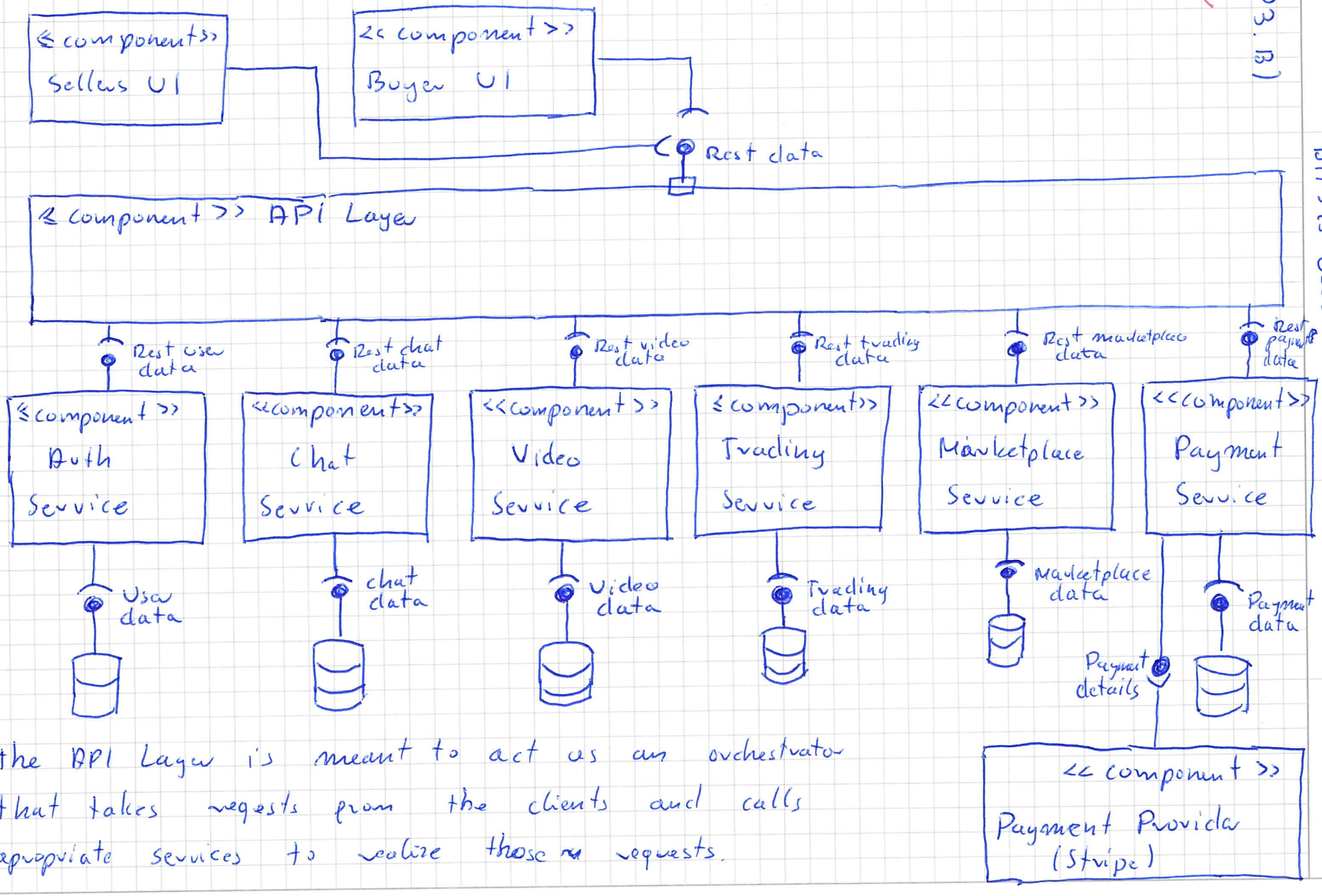
Even though SOA is less complex and cheaper than microservices, it is not as reliable in terms of fault tolerance having only a single database for the system - creating a single point of failure.

8P

Based on the platform description, it seem that functionalities are relatively independent from one another and having a database for each of those functionalities would make the system more reliable. Making microservices architecture a better choice.

12p

P3. B)



the API Layer is meant to act as an orchestrator that takes requests from the clients and calls apropriate services to realize those requests.

5

P3. c)

For the auction platfor, availability seems like the most important quality attribute since each time users cannot either buy or sell items the business is loosing (potentially) a lot of money.

Quality Attribute: Availability

Source: end-user ~~lays a bid~~

Stimulus: the end-user lays a bid for an existing item which crashes the service

Artifact: Trading Service

Environment: Normal operations

Response: The system detects the crash of the sevice, pauses all requests to that service, switches to a redundancy, and resumes the requests while restarting the main service.

Response measure: ~~within~~ Less than 4 seconds since the detection of the fault.

5

P4.

I participated in the workshop.

I do remember the decisions made during the workshop

P4.A)

My group selected availability as the most important quality attribute. The main idea was that a functional, ~~world~~ operating and accessible system will generate more profits in the long run. 2p

P4.B)

To support/promote availability, we introduced an active redundancy ~~when~~ where the system had a single point of failure. In our case, we kept a separate broker for drones communication as the redundancy. 2p

P4.C)

My group selected microservices for the core parts of the application and publish-subscribe for drone communication.

Microservices provided modularity, ~~flexibi~~ high fault tolerance, testability and maintainability which were all quality attributes that were considered important for the system that is working with different domains and different contexts, ~~to be important~~

5

P4.c) continue

Publish-subscribe turned out to be a standard used in drone communication which was an obvious choice for the secondary (additional) architecture style.

Some of the tactics that we used were:

- heartbeat - for availability
- sandbox - for testing
- active redundancy - availability
- encryption - security
- authentication - security

In general, using microservices with publish-subscribe as architecture styles and the listed tactics were a good choice. Other groups during the workshop also thought in a simmilar way.

What I would do differently next time is to think more about how components in the system connect and communicate with one another once the initial version of the architecture is complete. More iteration-based diagram creation could show more flaws and create a foundation for discussion more easily.

6p

CHALMERS	Anonymous code	Points for question (to be filled in by teacher)	Consecutive page no. Löpande sid nr 15
	Anonym kod DiT345-0009-PGP	Poäng på uppgiften (ifylles av lärare)	Question no. Uppgift nr 4

PG.D)

It was not easy to think about and take into consideration all the quality attributes to be represented equally in the system. It got obvious very quickly that trade-offs between different quality attributes are a must. The workshop also showed me that creating a system architecture requires a lot of broad knowledge in the domain of the system to come up with valuable conclusions and provide a good solution to the problem.

2p

5

P5

P5.A)

Module views are informal and are useful to describe the system at design time. -1

P5.B)

Component-and-Connector views are informal and are used to describe the system at run-time. -1

P5.C)

In quality attribute scenarios, a stimulus is a condition that requires a response.

P5.D)

In utility tree, the business value and the architectural impact are ranked with High, Medium, and Low

P5.E)

Heartbeat is an availability tactic that can be used to detect faults

P5.F)

A disadvantage of microservices is that it costs a lot.

P5.G) -0,5

In the BAPO model, B stands for business, A stands for architecture, P stands for process, and O stands for operation.

CHALMERS	Anonymous code Anonym kod DIT345 - 0009 - PGP	Points for question (to be filled in by teacher) Poäng på uppgiften (ifylles av lärare)	Consecutive page no. Löpande sid nr 17 Question no. Uppgift nr 5

P5 continue

P5. +1)

The quality attribute that is concerned with making future changes easier is called modifiability.

9,5