NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Science

Examination paper for TDT4240 Software Architecture

Academic contact during examination: Alf Inge Wang

Phone: +47 9228 9577

Examination date: Saturday June 11th 2016

Examination time (from-to): 09:00 – 13:00

Permitted examination support material:

- IEEE (2000), "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", Software Engineering Standards Committee of the IEEE Computer Society.
- Kruchten, P. (1995), "The 4+1 View Model of Architecture", IEEE Software, 12(6).
- English-Norwegian dictionary (or to your native language if your not Norwegian) and/or an English thesaurus (English-English).

Other information:

- Simple calculator or a calculator approved by NTNU allowed.
- English exam paper: pages 1-5
- Bokmål eksamensoppgave: sidene 7-11
- Nynorsk eksamensoppgåve: sidene 13 - 17

Language: English

Number of pages English exam paper: **5**

Number of pages enclosed: **17**

Checked by:

1/6-2016 Kristoffer Hagen

Date Signature

Problem 1: Various questions (10 points)

Answer these questions *briefly*:

1.1	What is Bass, Clements and Kazman's definition of Software Architecture (the definition in the textbook)?
1.2	Name the *four groups* of *modifiability tactics*.
1.3	Give *five important reasons* for why software architecture is important.
1.4	What is the *purpose* of architectural views?
1.5	What is the difference of a *Sensitivity Point* and a *Tradeoff Point*?
1.6	What is the difference of ATAM and CBAM?
1.7	Briefly describe the CBAM process.
1.8	How can the architecture of the game-loop affect the frame-rate of a game?
1.9	Name the *three main parts* needed to describe a design pattern.
1.10	Name one *Creational* design pattern.
1.11	Name one *Structural* design pattern.
1.12	Name one *Behavioral* design pattern.
1.13	What *type of models* are used to model *availability* analytically?
1.14	Name three typical *response measures* for a *modifiability* quality attribute scenario.
1.15	Name *the four steps* for reconstructing of a software architecture.
1.16	Why can it be necessary to reconstruct a software architecture?
1.17	What is the difference between a reconstructed and a designed software architecture?
1.18	Under what circumstances would a *Software Product Line* be used?
1.19	What is the *purpose* of a *Utility Tree*?
1.20	Under what circumstances can be it useful to use the *Composite* design pattern?

References:

- Bass, Clements & Kazman: "Software Architecture in Practice", 3rd edition, Pearson, 2013.
- Philippe Kruchten, "Architectural Blueprints – The 4+1 View model of Software Architecture", IEEE Software 12 (6), 1995

Problem 2: Choose the most appropriate architectural pattern (5 points)

Nominees:

a) Layered
b) Broker
c) Model-View-Controller
d) Pipe-and-Filter
e) Client-Server
f) Peer-to-Peer
g) Service-Oriented
h) Publish-Subscribe
i) Map-Reduce
j) Multi-tier

Choose the *most appropriate architectural pattern* (one) for the 5 descriptions below. Motivate for your choice in *one sentence* (give reasons for choosing the pattern):

1. Need to develop a simple operative system, which allows applications to use an API to utilize high-level functionality in component A. Component A is implemented using the core basic functionality for the operative system provided by component B. The operative system can work on different hardware, as component B interacts with the hardware indirectly through component C, which provides a universal API independent of the hardware used.
2. Need a web-based system where the centralized computing resources are divided into three parts: A *presentation* part which displays information related to various services, an *application* part – which controls the system's functionality and carries out the processing, and a *data* part where data is accessed, stored, changed and maintained.
3. Need software for a system where computing nodes can share services, processing power, disk storage, and network bandwidth to achieve dynamic load balancing and higher availability of the system.
4. Need software that makes it possible for online stores to use electronic payment services without needing to have the detailed information to access these payment services (such as URL) due to security reasons.
5. Need software for a flexible solution to visualize the same data for various configurations, such as single screen, multi-screen, multi-window, and various screen resolutions.

Problem 3: Edge-dominant system (5 points)

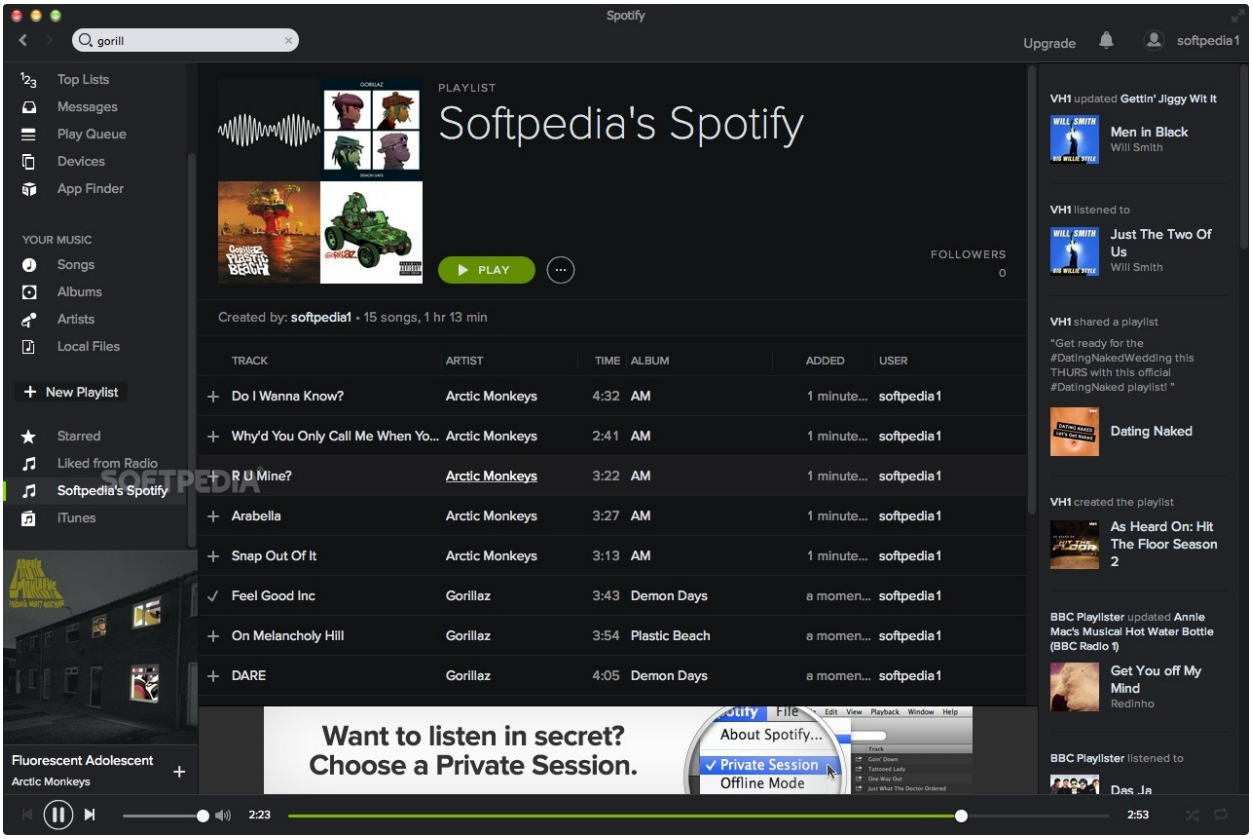
a) Create a sketch (drawing) and explain the Metrapolis model. (3 points)
b) How can a user change roles in open content systems and in open source software? (2 points)

Problem 4: Cloud Architecture (4 points)

a) Explain the difference between Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (Iaas). (2 points)
b) Explain the economic justification for cloud-based computing. (2 points)

Problem 5: Quality Attribute Scenario (6 points)

System description: The music streaming service Spotify, where the user can, among other things, browse and play songs from a large library of music, and create and share custom playlists. The client for the streaming service can be accessed on a PC, a smart-phone, a tablet, Smart-TVs, game consoles and many other digital devices.
The user interface of this system can typically look like the following:



a) Create one relevant quality attribute scenario on *usability* for the system described above. (2 points)
b) Create one relevant quality attribute scenario on *availability* for the system described above. (2 points)
c) Create one relevant quality attribute scenario on *performance* for the system described above. (2 points)

Problem 6 Design a software architecture (30 points)

Read the description below and do an architectural design. Your answer must include:

a) Architectural Significant Requirements/Architectural Drivers – 2 points
b) Architectural tactics and patterns – 3 points
c) Process view – 8 points
d) Logical view – 14 points
e) Architectural rationale – 3 points

Provide motivation for your choices and state your assumptions.

Robot Lawnmower (grass cutter)

Your goal is design the software architecture for the software controlling and managing a robot lawnmower. Robot lawnmowers are lawnmowers that can cut the grass within an area without human intervention. A *perimeter wire* is used to set up a border around the lawn that defines the area to be mowed. The robot uses this wire to locate the boundary of the area to be trimmed. The robot lawnmower comes with a *Base unit* where the battery-operated lawnmower gets charged. The lawnmower will return to the Base unit when finished cutting the grass or the battery is low. The user interface of the lawnmower can vary. Basic models let the user configure and operate the lawnmower through a simple display and some buttons. More advanced models provide touch screen interfaces on the lawnmower, and the most expensive models allow the user to control the lawnmower using a smart phone through Wi-Fi or 3G/4G. The following *operations* are available for the user: start/stop, schedule cutting, automatic cutting mode, and adjusting height of cutting. The architecture should make it easy to update the software running the lawnmower, including improving sensor interpretation and integration, detecting objects around the mower, navigation, strategy for movement, and the user-interface. These changes will not be made runtime, but rather through firmware upgrades or new software for newer models.
The robot lawnmower comes with the following sensors:

- Perimeter sensor – senses the perimeter wire.
- Radio sensor – locate the Base unit – which sends out radio frequency emissions.
- Front sensors – detects when it is about to hit an object to move around it.
- Rain sensor – lawnmower should return to Base unit if it rains.
- A camera sensor in the front – detects recognizable objects, which should be avoided such as human beings (children), animals, and other objects that can be harmed.
- A ground sensor – detects if the mower is being lifted off ground – and will be shut down immediately if being lifted as a safety precaution.

The robot lawnmower should be able to operate for weeks without any human intervention if it operates on automatic or scheduled mode.

NTNU – Trondheim
Norwegian University of
Science and Technology

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4240 Programvarearkitektur

Engelsk versjon er referanseversjon for eksamensoppgaven

Faglig kontakt under eksamen:	Alf Inge Wang
Telefon:	+47 9228 9577

Eksamensdato:	Lørdag 11.juni 2016
Tidspunkt for eksamen (fra-til):	09:00 – 13:00

Hjelpemiddelkode/Tillatte hjelpemidler:

- IEEE (2000), "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", Software Engineering Standards Committee of the IEEE Computer Society.
- Kruchten, P. (1995), "The 4+1 View Model of Architecture", IEEE Software, 12(6).
- English-Norwegian ordbok (eller fra ditt morsmål hvis det ikke er norsk) og/eller Engelsk synonymordbok (Engelsk-Engelsk).

Annen informasjon:

- Enkel eller godkjent kalkulator kan brukes.
- English exam paper: pages 1 – 5
- Bokmål eksamensoppgave: sidene 7 – 11
- Nynorsk eksamensoppgåve: sidene 13 – 17.

Språk:	Bokmål
Antall sider for bokmålsoppgave:	**5**
Totalt antall sider (alle språk):	**17**

Kontrollert av:

1/6-2016	Kristoffer Hagen
Dato	Signatur

Oppgave 1: Diverse spørsmål (10 poeng)

Svar kort på disse spørsmålene:

1.1	Hva er Bass', Clements' og Kazmanns definisjon på programvarearkitektur (i følge læreboka)?
1.2	Navngi de *fem hovedgruppene* for modifiserbarhetstaktikker.
1.3	Giv *fem viktige grunner* til hvorfor programvarearkitektur er viktig.
1.4	Hva er *formålet* med arkitektoniske perspektiver (views)?
1.5	Hva er *forskjellen* på et *Sensitivity Point* og et *Tradeoff Point*?
1.6	Hva er *forskjellen* på ATAM og CBAM?
1.7	Kort, beskriv CBAM prosessen.
1.8	Hvordan kan arkitekturen av en game-loop påvirke bildefrekvensen (frameraten) av et spill?
1.9	Navngi de *tre hoveddelene* som trengs for å beskrive et designmønster.
1.10	Navngi ett *Creational* designmønster.
1.11	Navngi ett *Structural* designmønster.
1.12	Navngi ett *Behavioral* designmønster.
1.13	Hvilke type modeller eksisterer for å modellere tilgjengelighet analytisk?
1.14	Navngi tre typiske *response measures* for et *modifiability* quality attribute scenario.
1.15	Navngi de *fire stegene* for rekonstruksjon av en programvarearkitektur.
1.16	Hvorfor kan det være nødvendig å rekonstruere en programvarearkitektur?
1.17	Hva er *forskjellen* på en rekonstruert og en designet programvarearkitektur?
1.18	Under hvilke forhold blir en *Software Product Line* brukt?
1.19	Hva er *hensikten* med ett *Utility Tree*?
1.20	Under hvilke forhold kan det være nyttig å bruke *Composite* designmønsteret?

Referanser:

- Bass, Clements & Kazman: "Software Architecture in Practice", 3rd edition, Pearson, 2013.
- Philippe Kruchten, "Architectural Blueprints – The 4+1 View model of Software Architecture", IEEE Software 12 (6), 1995

Oppgave 2: Velg arkitekturmønster som passer best (5 poeng)

De nominerte:

a) Layered
b) Broker
c) Model-View-Controller
d) Pipe-and-Filter
e) Client-Server
f) Peer-to-Peer
g) Service-Oriented
h) Publish-Subscribe
i) Map-Reduce
j) Multi-tier

Velg det mest passende arkitekturmønsteret (ett) for de 5 beskrivelsene under. Motiver for dine valg med en setning (gi grunner for valg av mønster):

1. Skal utvikle et enkelt operativsystem, som gir applikasjoner mulighet til å bruke et API for å utnytte høynivå-funksjonalitet i komponent A. Komponent A er implementert ved hjelp av kjernefunksjonalitet for operativsystemet tilbudt av komponent B. Operativsystemet kan samhandle med maskinvaren indirekte ved at komponent B benytter seg av komponent C, som tilbyr et universelt API som er uavhengig av maskinvaren brukt.
2. Trenger et web-basert system der de sentraliserte dataressursene er delt i tre deler: En *presentasjonsdel* som framviser informasjon relatert til ulike tjenester, en *applikasjonsdel* som kontrollerer systemets funksjonalitet og utfører prosesseringen, og en *datadel* hvor data blir aksessert, lagret, endret og vedlikeholdt.
3. Trenger programvare for et system der datanoder kan dele tjenester, prosesseringskraft, disklagring og nettverksbåndvidde for å oppnå dynamisk lastbalansering og høyere tilgjengelighet for systemet.
4. Trenger programvare som gjør det mulig for online-butikker å benytte seg av elektroniske betalingstjenester uten at man trenger å ha detaljert informasjon for å aksessere disse betalingstjenestene (som for eksempel URL) pga. sikkerhetsgrunner.
5. Trenger programvare for en fleksibel løsning til å visualisere samme data for forskjellige konfigurasjoner, som enkeltskjerm, flere skjermer, flere vinduer, og forskjellige skjermoppløsninger.

Oppgave 3: Edge-dominant systemer (5 poeng)

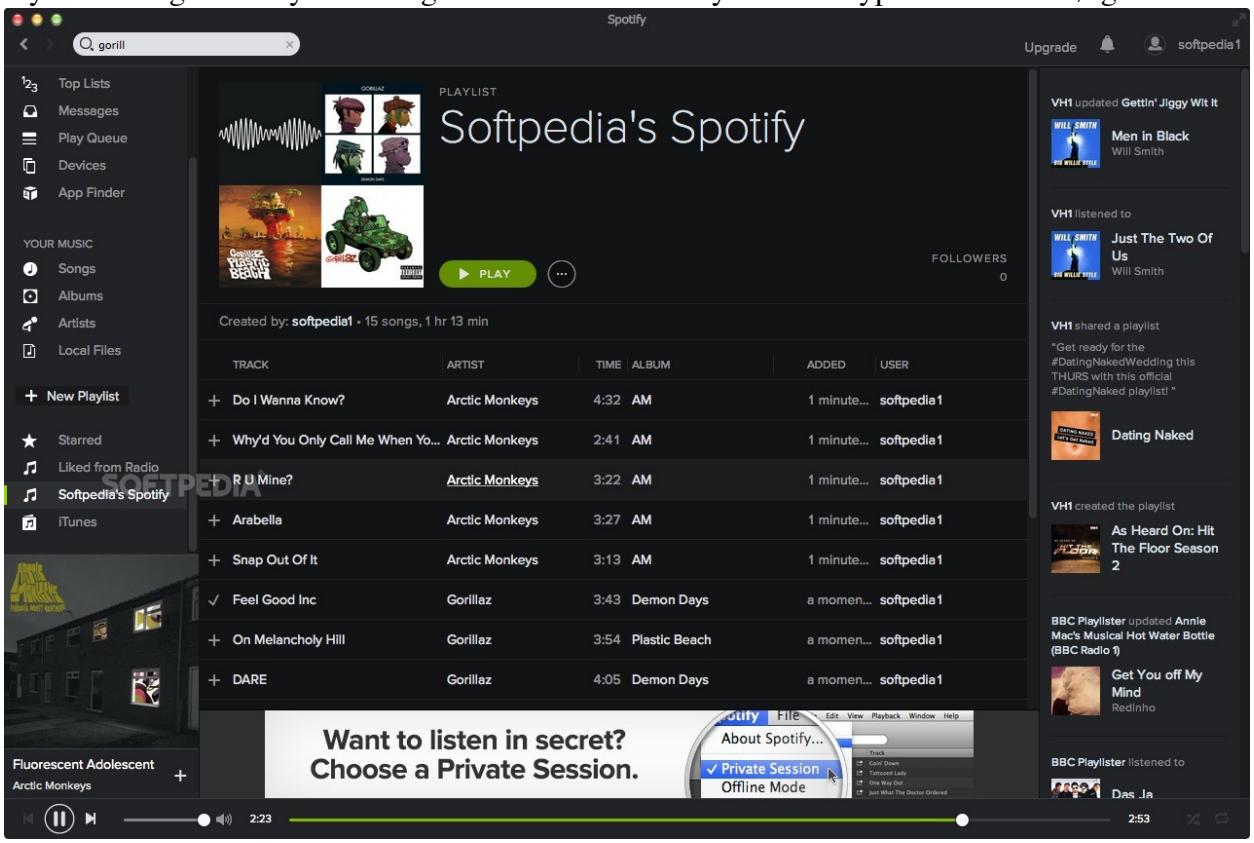
a) Lag en skisse (tegning) og forklar Metrapolis-modellen. (3 poeng)
b) Hvordan kan en bruker endre rolle i *open content* systemer og i åpen kildekode programvare? (2 poeng)

Oppgave 4: Skyarkitekturer (4 poeng)

a) Forklare forskjellen mellom *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* og *Infrastructure as a Service (Iaas)*. (2 poeng)
b) Forklar den økonomiske begrunnelsen for skybasert databehandling. (2 poeng)

Oppgave 5: Kvalitetsattributt-Scenario (6 poeng)

Systembeskrivelse: Musikkstrømmetjenesten Spotify, hvor brukeren kan bla. oppdage og spille sanger fra et stort bibliotek av musikk, samt lage og dele egne spillelister. Klienten til strømmetjenesten kan aksesseres på PC, smarttelefon, nettbrett, Smart-TV, spillkonsoller, og mye annet digitalt utstyr. Brukergrensesnittet av dette system kan typisk se ut som følgende:



a) Lag et relevant kvalitetsattributt-scenario på brukbarhet (*usability*) for system beskrevet ovenfor. (2 poeng)
b) Lag et relevant kvalitetsattributt-scenario på tilgjengelighet (*availability*) for systemet beskrevet ovenfor. (2 poeng)
c) Lag et relevant kvalitetsattributt-scenario på ytelse (*performance*) for systemet beskrevet ovenfor. (2 poeng)

Oppgave 6 Design en programvarearkitektur (30 poeng)

Les beskrivelsen under og utfør et arkitekturdesign. Svaret må inkludere:

a) Architectural Significant Requirements/Architectural Drivers – 2 poeng
b) Arkitekturtaktikker og –mønster (patterns) – 3 poeng
c) Process view – 8 poeng
d) Logisk view – 14 poeng
e) Motivasjon/Begrunnelse for arkitektur (Architectural rationale) – 3 poeng

Motiver for dine valg og beskriv dine antagelser.

Robotplenklipper

Ditt mål er å designe programvarearkitekturen til programvaren som kontrollerer og styrer en robotplenklipper. Robotplenklippere er plenklippere som kan klippe gresset innen et gitt område uten menneskelig innblanding. En *avgrensningskabel* er brukt til å sette opp grensen rundt plena. Roboten bruker denne kabelen til å lokalisere grensene for området som skal klippes. Robotplenklippere kommer med en *basestasjon* der den batteridrevne plenklipperen lades. Plenklipperen vil returnere til basestasjonen når den er ferdig med å klippe gresset eller batteriet er lavt. Brukergrensesnittet på plenklipperen kan variere. Enkle modeller lar brukeren konfigurere og operere plenklipperen via et enkelt display og noen knapper. Mer avanserte modeller tilbyr touchscreen-interface, og de dyreste modellene gir også brukeren mulighet til å styre og operere plenklipperen ved hjelp av en smarttelefon over Wi-Fi eller 3G/4G. Følgende operasjoner er tilgjengelige for bruker: start/stop, timeplanlegge klipping, automatisk klippemodus, og justering av klippehøyde. Arkitekturen skal gjøre det enkelt å oppdatere programvaren som styrer plenklipperen, inkludert forbedre sensortolking og –integrering, detektering av objekter rundt klipperen, navigering, strategi for forflytning, og brukergrensesnittet. Disse oppdateringene kan ikke gjøres run-time, men igjennom oppdateringer av firmware eller ny programvare for nye modeller.

Robotplenklipperen kommer med følgende sensorer:

- Perimetersensor – detekterer avgrensningskabelen.
- Radiosensor – lokaliserer basestasjon – som slipper ut radiofrekvenssignaler.
- Frontsensors – detekterer når den holder på å krasje i objekter, for så å kjøre rundt.
- Regnsensor –plenklipperen skal returnere til basen hvis det regner.
- Kamerasensor i front – detekterer gjenkjennbare objekter som bør unngås som mennesker (barn), dyr, og andre objekter som kan skades.
- Bakkesensor – som detekterer om klipperen blir løftet fra bakken – og vil da stenges av umiddelbart som et sikkerhetstiltak.

Robotklipperen skal være i stand til å operere i ukesvis uten menneskelig involvering hvis den opererer i automatisk eller timeplanlagt klippemodus.

NTNU – Trondheim
Norwegian University of
Science and Technology

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgåve i TDT4240 Programvarearkitektur

Engelsk versjon er referanseversjon for eksamensoppgåva

Fagleg kontakt under eksamen:	Alf Inge Wang
Telefon:	+47 9228 9577

Eksamensdato:	Laurdag 11.juni 2016
Tidspunkt for eksamen (frå-til):	09:00 – 13:00

Hjelpemiddelkode/Godkjend hjelpemiddel:

- IEEE (2000), "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems", Software Engineering Standards Committee of the IEEE Computer Society.
- Kruchten, P. (1995), "The 4+1 View Model of Architecture", IEEE Software, 12(6).
- English-Norwegian ordbok (eller fra ditt morsmål hvis det ikke er norsk) og/eller Engelsk synonymordbok (Engelsk-Engelsk).

Annen informasjon:

- Enkel eller godkjend kalkulator kan brukas.
- English exam paper: pages 1 – 5
- Bokmål eksamensoppgave: sidene 7 – 11
- Nynorsk eksamensoppgåve: sidene 13 – 17.

Språk:	Nynorsk
Tal på sider nynorsk oppgåvesett:	**5**
Totalt tal på sider (alle språk):	**17**

Kontrollert av:

1/6-2016	Kristoffer Hagen
Dato	Signatur

Oppgåve 1: Diverse spørsmål (10 poeng)

Svare kort på disse spørsmåla:

1.1	Kva er Bass', Clements' og Kazmanns definisjon på programvarearkitektur (i følje læreboka)?
1.2	Namngje dei *fem hovudgruppene* for *modifiability*-taktikkar.
1.3	Gje *fem viktige* grunnar til kvifor programvarearkitektur er viktig.
1.4	Kva er *formålet* med arkitektoniske perspektiv (views)?
1.5	Kva er *skilnaden* på eit *Sensitivity Point* og eit *Tradeoff Point*?
1.6	Kva er *forskjellen* på ATAM og CBAM?
1.7	Kort, beskriv CBAM prosessen.
1.8	Korleis kan arkitekturen av en game-loop påverke bildefrekvensen (framerate) av eit spill?
1.9	Namngje dei *tre hovuddelane* som trengs for å beskrive et designmønster.
1.10	Namngje eit *Creational* designmønster.
1.11	Namngje eit *Structural* designmønster.
1.12	Namngje eit *Behavioral* designmønster.
1.13	Kva for type modellar eksisterer for å modellere tilgjengelegheit analytisk?
1.14	Namngje tre typiske *response measures* for eit *modifiability* quality attribute scenario.
1.15	Namngje dei *fire stega* for rekonstruksjon av ein programvarearkitektur.
1.16	Kvifor kan det være naudsynt å rekonstruere ein programvarearkitektur?
1.17	Kva er *forskjellen* på ein rekonstruert og ein designa programvarearkitektur?
1.18	Under kva forhold blir ein *Software Product Line* brukt?
1.19	Kva er *hensikta* med eit *Utility Tree*?
1.20	Under kva forhold kan det være nyttig å bruke *Composite* designmønsteret?

Referansar:

- Bass, Clements & Kazman: "Software Architecture in Practice", 3rd edition, Pearson, 2013.
- Philippe Kruchten, "Architectural Blueprints – The 4+1 View model of Software Architecture", IEEE Software 12 (6), 1995

Oppgåve 2: Vel arkitekturmønster som passar best (5 poeng)

Dei nominerte:

a) Layered
b) Broker
c) Model-View-Controller
d) Pipe-and-Filter
e) Client-Server
f) Peer-to-Peer
g) Service-Oriented
h) Publish-Subscribe
i) Map-Reduce
j) Multi-tier

Vel det mest passande arkitekturmønsteret (eit) for de 5 beskrivingane under. Motiver for dine val med ein setning (gjev grunnar for val av mønster):

1. Skal utvikle eit enkelt operativsystem, som gjer applikasjonar mulegheit til å bruke eit API for å utnytte høgnivåfunksjonalitet i komponent A. Komponent A er implementert ved hjelp av kjernefunksjonalitet for operativsystemet tilbode av komponent B. Operativsystemet kan samhandle med maskinvaren indirekte med at komponent B brukar komponent C, som tilbyr eit universalt API som er uavhengig av maskinvaren brukt.
2. Treng eit web-basert system der dei sentraliserte dataressursane er delt i tre delar: Ein *presentasjonsdel* som syner informasjon relatert til ulike tenestar, ein *applikasjonsdel* som kontrollerer funksjonaliteten åt systemet og utfører prosesseringen, og ein *datadel* kor data blir aksessert, lagra, endra og vedlikehalde.
3. Treng programvare for eit system kor datanoder kan dele tenester, prosesseringskraft, disklagring og nettverksbandvidde for å oppnå dynamisk lastbalansering og høgare tilgjengelegheit for systemet.
4. Treng programvare som gjer det mogleg for online-butikkar å nytte seg av elektroniske betalingstenester utan at ein treng å ha detaljert informasjon for å aksessere desse betalingstenestene (til dømes URL) grunna sikkerheitsårsaker.
5. Treng programvare for ein fleksibel løysing til å visualisere same data for forskjellige konfigurasjonar, som enkeltskjerm, fleire skjermar, fleire vindauger, og forskjellige skjermoppløysningar.

Oppgåve 3: Edge-dominant system (5 poeng)

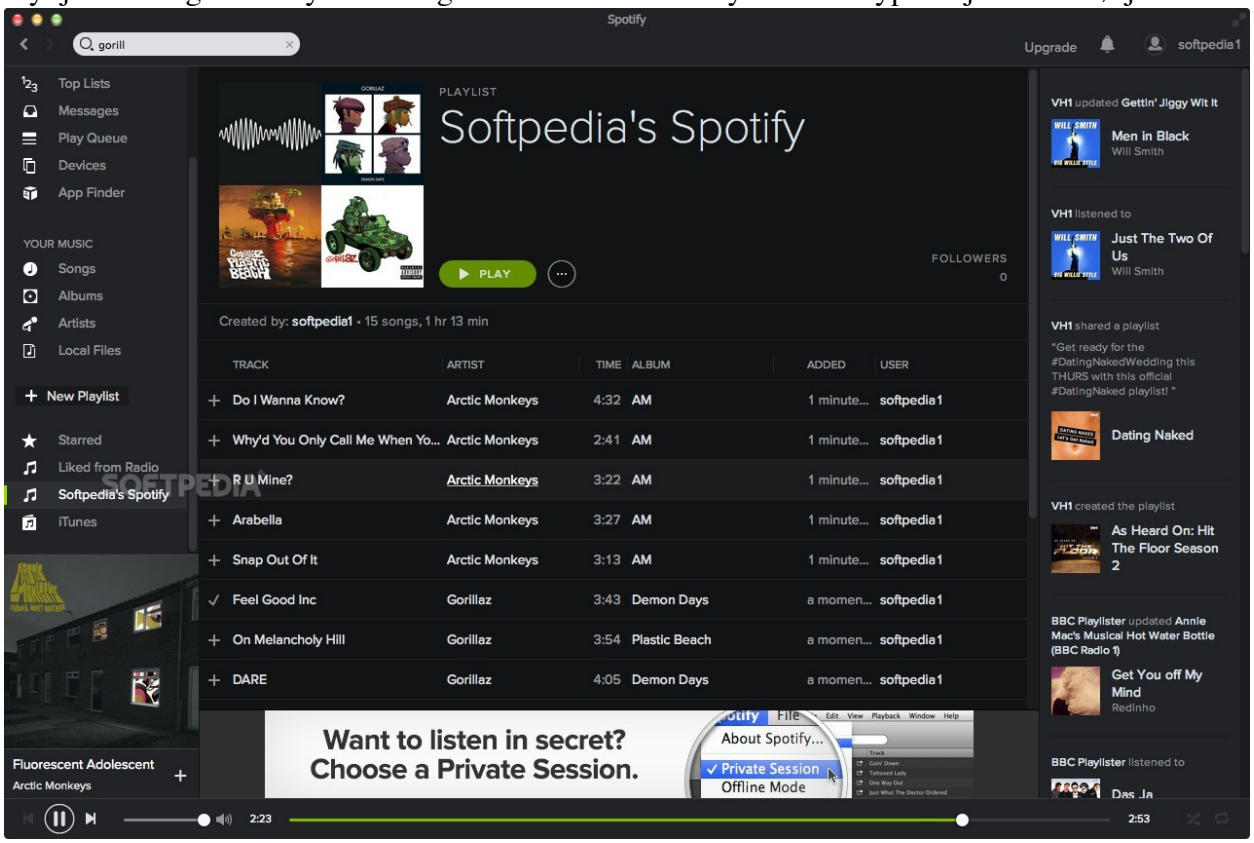
a) Lag ein skisse (teikning) og forklar Metrapolis-modellen. (3 poeng)
b) Korleis kan ein bruker endre rolle i *open content* system og i open kjeldekode programvare? (2 poeng)

Oppgåve 4: Sky-arkitekturar (4 poeng)

a) Forklar skilnaden mellom *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* og *Infrastructure as a Service (Iaas)*. (2 poeng)
b) Forklar den økonomiske grunngjevinga for skybasert databehandling. (2 poeng)

Oppgåve 5: Kvalitetsattributt-Scenario (6 poeng)

Systembeskrivelse: Musikkstrømmetenesta Spotify, kor brukeran kan t.d. oppdage og spille songar frå eit stort bibliotek av musikk, samt lage og dele egne spillelister. Klienten til strømmetenesta kan aksesseras på PC, smarttelefon, nettbrett, Smart-TV, spillkonsollar, og mykje anna digitalt utstyr. Brukargrensesnittet av dette system kan typisk sjå ut som føljande:



a) Lag eit relevant kvalitetsattributt-scenario på brukbarheit (*usability*) for system beskrive ovafor. (2 poeng)
b) Lag eit relevant kvalitetsattributt-scenario på tilgjengelegheit (*availability*) for systemet beskrive ovafor. (2 poeng)
c) Lag eit relevant kvalitetsattributt-scenario på yting (*performance*) for systemet beskrive ovafor. (2 poeng)

Oppgåve 6 Design en programvarearkitektur (30 poeng)

Les beskrivinga under og utfør eit arkitekturdesign. Svaret må inkludere:

a) Architectural Significant Requirements/Architectural Drivers – 2 poeng
b) Arkitekturtaktikkar og –mønster (patterns) – 3 poeng
c) Process view – 8 poeng
d) Logisk view – 14 poeng
e) Motivasjon/Grunngjeving for arkitektur (Architectural rationale) – 3 poeng

Motiver for dine val og beskriv dine forventingar.

Robotgrasklipper

Ditt mål er å designe programvarearkitekturen til programvaren som kontrollere og styrar ein robotgrasklipper. Robotgrasklipparar er grasklipparar som kan klippe graset innafor eit gjeven område utan menneskelig innblanding. En *avgrensningskabel* er brukt til å sette opp grensa rundt plena. Roboten brukar denne kabelen til å lokalisere grensene for området som skal klippas. Robotgrasklipparar kjem med ein *basestasjon* der den batteridrivne grasklipparen lades. Grasklipparen vil returnere til basestasjonen når den er ferdig med å klippe graset eller batteriet er lavt. Brukargrensesnittet på grasklipparen kan variere. Enkle modellar lar brukaren konfigurere og operere grasklipparen ved hjelp av eit enkelt display og noen knappar. Meir avanserte modeller tilbyr touchscreen-interface, og de dyraste modellane gjer også brukaren mogligheit til å styre og operere grasklipparen ved hjelp av ein smarttelefon over Wi-Fi eller 3G/4G. Føljande operasjonar er tilgjengelige for ein brukar: start/stop, timeplanlagt klipping, automatisk klippemodus, og justering av klippehøgde. Arkitekturen skal gjøre det enkelt å oppdatere programvaren som køyrer plenklipparen, inkludert forbetre sensortolking og –integrering, detektering av objekt rundt klipperan, navigering, strategi for å flytte på seg, og brukargrensesnittet. Disse oppdateringane kan ikkje gjerast run-time, men igjennom oppdateringar av firmware eller ny programvare for nye modellar.
Robotgrasklipperen kjem med følgjande sensorar:

- Perimetersensor – detekterar avgrensningskabelen.
- Radiosensor – lokaliserar basestasjon – som slipp ut radiofrekvens signal.
- Frontsensors – detekterar når den held på å krasje i objekt for å køyre rundt.
- Regnsensor – plenklipparen skal returnere til basen om det regner.
- Kamerasensor i front – detektarer objekt som kan kjennast igjen som bør unngåas som menneskje (barn), dyr, og andre objekt som kan skades.
- Bakkesensor – som detektarer om klipparen blir løfta frå bakken – og vil da stenges av med ein gong som et sikkerheitstiltak.

Robotklipperen skal vera i stand til å operere i fleire uke utan menneskeleg involvering om den opererer i automatisk eller timeplanlagt klippemodus.