**Question Bank | Software Engineering | IV1300 | 2016**

Sista frågan i det här dokumentet kommer att ställas bara på första ordinarietentan.

**OBS REGLER:**

På tentan måste dina svar skilja sig ifrån svaren skriva här.

Radera inte någons svar, håller du inte med så skriv en kommentar så här.

# **1. Basics of Software Engineering**

**1.1 QUESTION: Definition of software (1 point) (based on Mira’s material, Chapter 1.3)**

What is software? (1 point)

Det är mer än bara källkod och objektkod, det är även följande saker:

Kravspecifikation

Deisgnspecifikation

Implementation

Testspecifikation

Användarmanualer

Konfigurationsfiler

Operationsprocedurer/Instruktioner (Sätta up och operera systemet)

Instruktioner (Hur man ska reagera på systemfel)

**1.2 QUESTION: Definition of software engineering (2,5 points)**

What is the definition of software engineering (0,5 points) and what two key phrases does this definition include? (2 points)

Software engineering är en ingenjörsmetodik som berör alla aspekter av programvaruutveckling, från de tidigaste stadierna (krav/systemspecifikation) fram till att systemet går i drift och ska underhållas.

***Engineering disciplin:*** Ingenjören använder sig av lämpliga teorier, metoder och verktyg för att lösa problem med givna organisations- och ekonomiska restriktioner.

***All aspects of software production*** *(Alla aspekter av mjukvaruutveckling)****.***

**1.3 QUESTION: Relationship to other disciplines (1 point)**

How is software engineering related to computer science and system engineering?

*Computer science* berör teorier och metoder som ligger bakom datorer och mjukvarusystem. Kunskap om detta är nödvändigt för en mjukvaruingenjör.

*System engineering* berör alla aspekter av utvecklandet av ett komplext mjukvarusystem, alltifrån hårdvaruutveckling, processdesign och distribution, och software engineering.

**1.4 QUESTION: Diversity of software application (3 points)**

One of the most significant factor in determining software engineering methods is the type of application that is being developed. Sommerville lists eight such types. List (1,5 points) and explain (1,5 points) six of them.

* **Stand-alone applications**

Program som körs på en lokal maskin, innehåller all nödvändig funktionalitet och behöver inte vara uppkopplad mot ett nätverk. Ex: Photoshop, MS Word, VLC.

* **Interactive transaction-based applications**

Exekveras på en server som är tillgänglig från användarens egen maskin. Ex: Amazon, Swish, Sociala medier, Onlinespel.

* **Embedded control systems**

Det är mjukvarusystem som styr och kontrollerar hårdvaruutrustning. Ex: Bil,  
mikrovågsugn, flygplan.

* **Batch processing systems**

Affärssystem som är designade för att bearbeta stora mängder data. Tar emot stora mängder input och skapar motsvarande output. Ex: Periodiska faktureringssystem så som mobilfakturor och löner för hos företag.

* **Entertainment systems**

System som primärt är för personligt bruk med syfte för att underhålla användaren. Majoriteten är system för spel. Kvalitet på systemets användargränssnitt är väldigt viktigt för entertainment systems.

* **Systems for modelling and simulation**

System utvecklade av forskare och ingenjörer för att modelera fysiskaprocesser eller simulera situationer som innehåller många separata, samspelande objekt. Model- och simuleringssystem är ofta väldigt beräkningsintensiva och kräver högprestanda-paralella system.

* **Data collection systems**

Datakollektionssystem är mjukvara som kommunicerar med sensorer för att samla data som sedan kan sändas vidare till andra system för bearbetning. Sensorerna används ofta inom fientliga områden, så som insidan av en motor eller på en väldigt avlägsen plats

* **Systems of systems**

System som är uppbygd av ett nummer av olika andra mjukvarusystem. Några av dom kan vara generiska mjukvaruprodukter (t.ex spreadsheet program), andra kan vara specifikt skapade för just ett ändamål.

**1.5 QUESTION: Relationship to other disciplines (3 points)**

There is no universal software engineering method or technique that is applicable for all different types of software. However, there are three general issues that affect different types of software. These are *heterogeneity*, *business* *and social change* and *security and trust*. Explain in what way these issues provide challenge to software developers?

*Heterogeneity is a word that signifies diversity. A classroom consisting of people from lots of different backgrounds would be considered having the quality of heterogeneity.*

**Heterogeneity**

System ska fungera med en rad olika andra system, allt ifrån äldre system till system som kör på andra platformar eller system skrivna i ett annat programmeringsspråk. Utmaningen här ligger i att skapa system som är flexibla nog att funktionera med den här heterogeneity.

**Business and Social Change**

Affärsmarknaden och samhället förändras snabbt när nya ekonomier utvecklas och ny teknologi blir tillgänglig. Nuvarande mjukvara ska kunna ändras fort och man ska snabbt utveckla ny mjukvara. Många traditionella mjukvaruingenjörstekniker är tidskrävande och kan ta lång tid att utveckla och leverera. Det måste förbättras så att tiden för att utveckla ny mjukvara som ger nytta åt kunden blir förminskad.

**Security and Trust**

Nu när mjukvara är så sammanflätat med alla våra aspekter av våra liv är det viktigt att vi kan lita på mjukvaran. Det här är speciellt sant för fjärrstyrda mjukvarusystem som är tillgängliga genom en hemsida eller liknande. Det är vädlgit viktigt att skadliga användare inte kan förstöra vår mjukvara samt att säkertheten av information uppehålls.

**1.6 QUESTION: Types of software products (2 points)**

There are two types of software products. What are they? (0,5) Provide an example of each type. (0,5) Who is in the control of the specification of each product type? Motivate? (1 point)

**Generiska produkter** och **skräddarsydda/specialbeställda produkter.**

**Generiska produkter** är självständiga system som är skapade av utvecklingsteam för att sedan säljas på öppen marknad för vem som helst. T.ex mjukvara för PC’s så som Word, Photoshop, spel. Här ligger all kontrol hos utvecklingsteamet.

**Skräddarsydda produkter** är beställda av en viss kund och utvecklade av en annan entitet under ett bindande kontrakt av kundens krav. T.ex banksystem eller en hemsida för ett företag. Här är det kunden som bestämmaer vad som ska vara med.

**1.7 QUESTION: Professional and ethical responsibilities (4 points)**

Like other engineers, software engineers must accept that their job involves wider responsibilities than simply the application of technical skills, the professional and ethical responsibilities.

Some of the professional and ethical responsibilities are: Confidentiality, Competence, Intellectual property rights, Computer misuse.

Describe briefly in what way the software engineers may misuse their responsibilities.

**Confidentiality**

Mjukvaruingenjörer ska respektera tilliten av arbetsgivaren samt kunden oavsett om ett formellt sekretesskontrakt har signerats eller ej.

**Competence**

Mjukvaruingenjörer ska inte missrepresentera sin kunskap och kompetens. Samt inte medvetet acceptera arbete utanför ens kunskaper.

**Intellectual Property Rights**

Mjukvaruingenjörer ska ha bra uppsikt på lagar över immartiel äganderätt, så som patent och upphovsrätt. Samt försäkra den immartiela äganderätten för arbetsgivaren och kunden.

**Computer Misuse**

Mjukvaruingenjörer ska inte använda deras tekniska skickligheter för att missbruka andra personers datorer. Datormissbruk är allt ifrån triviala saker (spela på någon annans dator) till extremt seriösa missbruk (virus/malware).

# **2. Organizational Levels**

**2.1 QUESTION:** **Organizational levels and strategies** **(6 points) based on Mira’s material in Chapter 5**

Give an account of organizational levels and the strategies that apply on each level.

**Business Strategic Level**

Den här nivån berör alla strategier berörande affärer, kunder och produkter på ett långsiktigt plan. Varje strategi har sina egna “Strategy visions”, “Strategy missions”, “Strategy Goals” och “Strategy Values”.

**Tactical Level**

Strategier för att uppfylla affärsplanen. Berör strategier så som “Product Management Strategy”, “Lifecycle Process Strategy”, “Technology Strategy” och “Support Strategy”.

**Implementation Level**

Implementations nivån sköter allt som har med utveckling, underhållning, testning och fortsatt utveckling.

**3. Software Processes**

**3.1 QUESTION: Process versus process model (1 point)**

What is a process and process model? (1 point)

**Process**

En process är alla aktiviteter involverade i att skapa ett mjukvarusystem.

**Process Model**

En process model är en abstrakt representation av processerna. Kan vara i text, diagram, lista eller andra format. Innehåller ibland villkor som man kan titta efter processen är klar för att se om de uppfylldes eller ej.

**3.2 QUESTION: Reasons for modelling processes (4 points)**

Lari Lawrence Pfleeger (see Mira’s slides) lists reasons for modelling processes. List and explain them.

* Ger en allmän förståelse för processen.
* Uppnå strukturerad och konsekventa aktiviter.
* Leder handlingarna så man vet vad man ska göra härnäst.
* Man kan undersöka processen bättre och på så sätt få en bättre förståelse för det hela samt hitta överflödiga eller försummade delar som inte behöver vara med. Eller om det är något som saknas.

**3.3 QUESTION: Process Model (4 points)**

Give an account of the waterfall model.

*Vattenfallsmodellen* är en sekventiel designprocess som delar upp de olika processerna av mjukvaruutevecklig i olika faser. Det kallas för vattenfallsmodellen för man tar en fas i taget för att sedan forstätta vidare, när man bestämt sig för att gå vidare finns det ingen återvåndo bakåt till föregående faser. Faserna går oftast så här, Krav → Design → Implementation → Verifikation/Testning → Underhållning.

**3.4 QUESTION: Process Model (4 points)**

Give an account of the incremental model.

Inkrementalmodellen är en mjukvarumetod där programmet är designat, implementerat och testat i olika delar. För varje iteration läggs lite ytterligare saker till programmet tills dess att produkten är klar.

**3.5 QUESTION: Process Model (4 points)**

Give an account of the reuse-oriented model.

Återanvändnings-orienterad-modell är metoden att återavända färdiga komponenter, integrera deras funktion med varandra till ett nytt system. Så fokuset ligger på att implementera de färdiga komponenterna till systemet istället för att skapa allt från grunden.

**3.6 QUESTION: Process Phases (4 points)**

There are many different process models, but all must include four fundamental software engineering activities. These are:

1. Software specification
2. Software design and implementation
3. Software validation
4. Software evolution.

Give an account of software specification phase and its inclusive requirements engineering process.

Mjukvaruspecifikationfasen är processen av att sätta sig in och förstå sig på systemet. Här definierar man alla krav och identifierar alla gränser för systemets funktionalitet och utveckling. Det finns fyra huvudaktiviteter inom kravspecifikationsfasen.

1. **Feasibility study/Förstudie:** En förstudie utförs för att se om utvecklaren kan skapa systemet med existerande mjukvara och hårdvara. Förstudien tar också hänsyn till om det föreslagna systemet kommer att vara kosteffektivt från ett affärsperspektiv och om det är möjligt att genomföra på den existerande budgeten.
2. **Kravinsamling och analys:** Processen att ta fram systemkrav genom observation av befintliga system, diskussioner med framtida användare och kunder, och uppgiftsanalys. Ibland behövs en eller flera systemmodeller och prototyper tas fram för att hjälpa en att förstå systemet och dess krav.
3. **Kravspecifikation:** Aktiviteten att översätta intagen information från analysen och sätta upp användarkrav samt systemkrav.
4. **Kravvalidering:** Kontrollera kraven för realism, konsistens och fullständighet.

**3.7 QUESTION: Process Phases (4 points)**

Give an account of Boehm’s spiral model – the risk-driven software process framework.

Spiral modellen är lik inkerementalmodellen men med mer fokus på riskanalys. Spiral modellen har fyra faser: Planering, Riskanalys, Engineering och Utvärdering. Ett projekt repeterar de här fyra faserna under dess utveckling, som en spiral. Det hela startar med planeringsfasen där krav samlas in och risker bedöms. Varje varv runt i spiralen bygger på det första varvet. I riskanalysfasen identifieras risker och alternativa lösningar tas fram. Ofta produceras en prototyp i slutet av riskanalysen. Engineeringsfasen utvecklas och testas mjukvara. I utvärderingsfasen så bedöms produkten av kunden i dess nuvarande stadie innan det fortsätter till nästa varv i spiralen.

**4. Requirements**

**4.1 QUESTION: Requirement levels (3 points)**

Sommerville describes two levels of requirements. What are they? (1 point) Why do you need to write requirements on different levels of detail? (1 point) How are these requirements structured into hierarchies? (1 point)

**User requirements** och **System requirements.**

Användarkrav är påståenden på ett naturligt språk med diagram, som beskriver vilka tjänster systemet ska uppfylla och vilka begränsningar systemet ska klara av att funktionera under.

Systemkrav är mer detaljerade beskrivningar av systemets funktioner, tjänster och begränsningar. Systemkraven ska dokumentera exakt det som ska implementeras, det kan vara en del av kontraktet mellan kund och utvecklare.

Olika nivåer av krav är användbart därför de förmedlar information om systemet till olika sorters läsare. Läsarna av användarkrav är vanligtvis inte intresserade av hur systemet ska implementeras eller små detaljer om systemet.

Hierarkiskt sätt är användarkrav rangordnade högre än systemkrav och är därför viktigare.

**4.2 QUESTION: Requirement types (4 points)**

Software requirements are often grouped into functional and non-functional requirements. Describe briefly each requirements type and exemplify it.

Functional requirements är vad systemet ska kunna göra och hur det ska reagera på viss input-data och fungera i olika situationer. I vissa fall kan funktionellakrav explicit säga vad systemet inte ska göra.

Non-functional requirements är begränsningar på tjänster eller funktioner erbjudna av systemet. Det inkluderar tidsbegränsningar, begränsningar på utvecklings processen. Icke-funktionellakrav berör oftast systemet som en helhet istället för individuella systemtjänster. Icke-funktionellakrav kan beröra saker så som pålitlighet, svarstid, säkerhet eller prestanda.

**4.3 QUESTION: Non-functional requirements (2 points)**

Describe three main different types of non-functional requirement which may be placed on a system. Give at least one example of each of the types.

**Product Requirements**

Dessa krav beskriver eller begränsar beteendet av mjukvaran. Exempelvis kan vara ett prestandakrav på hur snabbt systemet måste exekvera eller hur mycket minne det kräver.

**Organizational Requirements**

Det här är breda systemkrav som härledds från företagspolicy eller procedurer inom kundens eller utvecklarens organisation. T.ex utvecklings processen har krav på vilket programmerings språk som ska användas. Eller vilken utvecklingsmiljö och vilka processtandarder som ska användas.

**External Requirements**

Externakrav är alla sorters krav som kommer ifrån saker som inte är en del av systemet eller dess utvecklingsprocess. Exempelvis kan det vara lagar som måste följas eller etiska krav som måste uppfyllas för att slutanvändarna och allmänheten ska godkänna det.

**4.4 QUESTION: Defining requirements in natural language (2 points)**

Discuss the problem of using natural language for defining user and system requirements and show, using small examples, how structuring natural language into forms can help avoid some of these difficulties.

Naturligt språk kan potentiellt vara vagt, mångtydigt eller så kan meningen skilja sig beroende på vilken bakgrund läsaren har. För att underlätta och undvika sånna här problem kan man använda sig av följande riktlinjer: Använd ett standardiserat format. använd dig av textmarkering så som färg, fettstilt eller kursivt. Förutse inte att läsaren förstår alla tekniska termer, undvik därför att använda förkortningar, slang, etc.

**4.5 QUESTION: Requirements engineering process phase (4 points)**

After initial feasibility studies, the next stage of the requirements engineering process is requirements elicitation and analysis.

Describe the process and each of its phases. (2 points)

Give four reasons for why the elicitation and analysis phase is difficult. (2 points)

**Requirements Discovery**

Är processen av att kommunicera med beställarna av systemet för att upptäcka deras krav. Domänkrav och dokumentation är också framtagna under denna processen.

**Requirements Classification and Organization**

I den här fasen organiseras, klassificeras och grupperas alla de olika kraven för systemet. Resultatet blir organiserade kluster av sammanhängande krav.

**Requirements Prioritization and Negotiation**

Under denna fas så kommer krav som skapar konflikter att bli prioriterade och utredade genom förhandling.

**Requirements Specification**

Kraven blir dokumenterade för att användas i fortsättningen av projektet. Både formell och informella kravdokument kan tas fram.

1. Intresenter vet ofta inte vad dom vill ha utav ett datorsystem bortsett från de mest generella termerna. Det kan vara ett problem för dom att formulera deras vilja av vad systemet ska göra. Samt att de kanske stället orealistiska krav eftersom dom inte har koll på vad en dator kan och inte kan göra.
2. Intressenter kan ibland ställa krav på deras egna termer och språk som utvecklaren inte begriper.
3. Olika intressenter har olika krav som framställs på diverse sätt. Utvecklarna måste finna alla samt identifiera likheter och konflikter.
4. Politiska faktor kan påverka krav av systemet. En manager kan be om krav som hade givit dem större makt inom deras organisation.
5. Affärsplanen eller ekonomin kan ändras under analysen och krav kan skifta i vikt och prioritet. Nya krav kanske dyker upp från intressenter som inte tidigare varit med.

**4.6 QUESTION: Requirements engineering discovery process phase (4 points)**

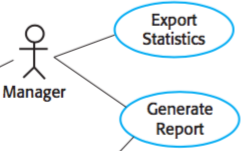
When discovering requirements, you may use different techniques such as

1. Use cases
2. Interviewing
3. Scenarios
4. Ethnography

Describe (2 points) and compare two of them (2 points).

**Use cases**

Use cases identifierar aktörer involverade i en interaktion och namger interaktionen. Det är sedan kompletterat med ytterligare information som beskriver interaktionen med systemet. Den tillägda information kan vara i en beskrivning text eller ett diagram som visar interaktionen.



På grund av att use cases fokuserar på interaktioner med systemet är dom inte lämpade för att upptäcka begränsningar, hög nivåns affärskrav, icke-funktionellakrav eller domänkrav.

**Interviewing**

Formella och informella intervjuer av intressenter är en del av de flesta kravinsamlingsprocesser. Det finns två typer av intervjuer: Öppen intervju där det inte finns någon förutbestämd agenda och sen finns där Stängda intervjuer där intressenterna svara på ett par bestämda frågor. Det kan vara svårt att få domänkunskap utav intervjuer på grund av två anledningar:  
1. Alla programspecialister använder terminologi och slang som är specifikt för en domän. Det är näst intill omöjligt att diskutera domänkrav utan att använda domänterminologin. De använder terminolign på ett exakt och svårförstått sätt vilket är väldigt enkelt för ingenjören som utför intervjun att missförstå.

2. Viss domänkunskap är så välbekant med intressenter så de finner det svårt att förklara eller tycker det är så fundamentalt så det inte är värt att nämna.

**Scenarios**

Ett scenario är ett exempel taget direkt från verkligheten. Det är beskrivning av händelser och interaktioner med systemet oftast i textformat men kan ha bilder och diagram till hjälp. Ett scenario brukar innehålla följande saker:

1. En beskrivning av vad systemet och användaren förväntar sig när scenariot börjar.
2. En beskrivning av ett normalt flöde av event i scenariot.
3. En beskrivning av vad som kan gå fel och hur det ska hanteras.
4. Information om andra aktiviter som kan pågå samtidigt.
5. En beskrivning och systemets tillstånd efter scenariot är avslutat.

Scenarion och use cases är sig väldigt lika och scenarios har därför samma styrkor och svagheter som use cases när det kommer till vad man upptäcker och vad man missar.

**Ethnography**

Etnografi är en obsverations teknik som används för att förstå operationella processer och hjälpa härledda krav för dessa processer. En analytiker fördjuper sig själv i arbetsmiljön där systemet ska komma att bli använt. De normala arbetsdagarna studeras och de uppgifter som dom olika rollerna utför noteras. Etnografi är användbart för man upptäcker underförstådda systemkrav som reflekterar sättet som folk arbetar på, istället för den formella processen som är definierad av organisationen. Lite mer om entograi i 4.7.

**4.7 QUESTION: Requirements engineering discovery process phase (2 points)**

Describe (1 point) and motivate (1 point) the role of ethnography within requirements engineering process?

Se 4.6 för ytterligare information om etnografi.

Etnografi är särskilt effektivt för att upptäcka två typer av krav:

1. Krav som är framtanga från sättet folk faktiskt arbetar på itsället för den fördefinierade processen som de skall följa. Till exempel flygtrafikskontrollanter kan ibland stänga av konlift alarm systemet som upptäcker flygplan med korsande vägar. Även när normal procedur säger att det ska vara igång. De sätter medvetet flygplan på korsande väg för att hjälpa kontrollera luftrummet. Deras stratgei är att flytta undan flygplanen innan en krasch händer och de finner att alarmet stör deras arbete.
2. Krav som är framtagna från samarbete och vetskap om anställdas aktiviteter. Till exempel luftkontrollanterna igen är medvetna om andra kontrollanters förutsägningar om antalet flygplan som finns i deras luftsektor. De kan sedan modifiera deras strategi utifrån de här förutsägningarna. Från det här kan man då se kravet för ett automatiserat system som tillåter kontrollanterna ha uppsikt över närliggande luftsektorer.

**4.8 QUESTION: Requirements engineering discovery process phase (3 points)**

During the requirements validation phase, different types of checks should be carried out on the requirements in the requirements document. Sommerville suggests five types of checks. List and describe three of them.

1. Validity checks: En användare må tro att systemet behöver utföra särskillda funktioner men vidare analys kan visa att ytterligare eller andra funktioner är nödvändiga. Ett system har många intressenter med olika behov och vilken mängd som helst av funktioner kommer oundvikligen äventyra andra intresenters funktioner.
2. Consistency checks: Krav inom ett dokument ska inte skapa konflikter. Det ska inte finnas motsägande begränsningar eller olika sorters beskrivningar av samma systemfunktion.
3. Completeness checks: Kravdokumentet ska bara innehålla krav som definierar funktioner och begränsningar avsedda för systemanvändaren.
4. Realism checks: Kan kraven verkligen bli implementerade med den nuvarande tekniken, räcker budgeten till eller finns där tillräckligt med tid för att implementera kravet.
5. Verifiability: För att minska potentiella dispyter mellan kund och utvecklare ska systemkrav vara skrivna så att de är verifierbara. Det här betyder att det ska vara möjligt att skriva tester för att demonstrera att det levererade systemet uppfyller kraven.

**4.9 QUESTION: Requirements engineering discovery process phase (3 points)**

There are a number of requirements validation techniques which can be used. These are requirements reviews, prototyping, and test-case generation. Describe each of them.

**Requirements Reviews**

Kraven är analyserade systematiskt av ett team som kollar efter fel och motsägelser.

**Prototyping**

En exekverbar model av systemet demonstreras för slutanvändarna och kunderna vilket dem kan experimentera med och se om den uppfyller deras behov.

**Test-case Generation**

Det ska vara möjligt att testa krav. Om det är svårt eller omöjligt att testa så betyder det oftast att det kommer bli svårt att implementera kravet och det borde bli omtänkt.

**4.10 QUESTION: Requirements change management (3 points)**

The requirements for software systems are always changing. Hence, one must define a process of understanding and controlling changes to system requirements. Describe the requirements management process.

Det hela börjar med grundläggande steget, plannering. Planneringstadiet etablerar de olika detalj nivåerna av kravhantering som krävs. Under requirements management måste man bestämma ett par saker.

Requirements Identification: Varje krav måste vara unikt identifierbart så att man kan krosreferensera med andra krav och användas i traceability assessment.

En “Change management process” ska finnas. Det är ett set av aktiviteter som faststället impakten och kostnaden av ändringar.

Traceabiliy policies: Dessa policys definierar hur varje förändring av krav och systemdesignen ska dokumenteras hur dessa dokument ska underhållas.

Tool support: Kravmanagement betyder bearbetning av stora mängder information om krav. Det finns verktyg som kan användas för att underlätta den här processen, allt ifrån specialiserade kravmanagementsystem till kalkylbrad eller simpla databas system.

Requirements change management är en process som bör appliceras på alla systemkravs ändringar efter att kravdokumentet har blivit godkänt. Change management är fundamentalt på grund av att du behöver bestämma om fördelarna med att implementera de nya kraven är berättigade av kostnaderna av implementationen. Fördelen med att använda en formell process till förändringar är att alla föreslagna ändringar är behandlade överensstämmande och alla ändringar i kravdokumenten sköts på ett kontrollerat sätt. Det finns tre steg inom change management processen.

Problem analys och kravspecifikation, processen startar med att identifiera krav problem. Problemet eller lösningen analyseras för att kontrollera så det är giltigt. Analysen skickas sedan tillbaka till den som rapporterade problemet som i sin tur kan svara med ett specifikt krav förändringsförslag eller dra tillbaka förfrågan om förändrig.

Change analysis and costing, effekten av ett föreslaget ändring är fastställt med traceability information och allmänkunskap av systemetskrav. Kostande av att genomföra ändringen beräknas båda i termer om modifiering av kravdokumentet, systemdesignen och implementationen. När analysen är klar så görs ett beslut om man ska fortsätta med ändringen eller ej.

Change implementation, kravdokumentet, systemdesignen och implementationen är modifierad. Dokumenten bör vara organiserade så att ändringar inte kräver stor omskrivning eller omorganisering.

**5. Testing**

**5.1 QUESTION: Verification and validation (2 points)**

Verification and validation are not the same thing. What is the difference? Explain why validation is a particularly difficult process.

Verifiering siktar sig mot att kontrollera att mjukvaran möter sina angivna funktionella krav samt icke-funktionella krav.

Validering är strävan efter att försäkra sig om att mjukvaran möter kundens förväntningar. Det går bortom att bara kolla överrenstämmelse med kravspecifikationen till att demonstrera att mjukvaran gör vad kunden förväntar sig.

Validering är svårt på grund av det inte finns konkreta sätt att bevisa att systemet uppfyller det som ska uppfyllas. Till skillnad från verifiering där det räcker med att man godkänner alla tester för att systemet ska bli verifierat.

**5.2 QUESTION: Confidence level in verification and validation (3 points)**

The ultimate goal of verification and validation is to establish confidence that the software system is “fit for purpose”. However, the level of confidence depends on (1) software purpose, (2) user expectations, and (3) marketing environment. Motivate how the level of confidence varies with respect to the three above-listed reasons.

**Software Purpose**

Beronde på vilket ändamål systemet ska ha, till exempel om det är mjukvara för ett flygplan måste man ha högt förtroende för systemet jämfört en liten app på mobilen.

**User Expectations**

Eftersom användare sen tidigare har haft dåliga erfarenheter med buggiga, opålitliga mjukvarusystem kan många användare ha låga förväntningar på mjukvarukvaliteten. När ett nytt system ska installeras kan ny användare tolerera kraschar eftersom fördelarna med programmet väger ut nackdelarna med kraschåterhämtning. Allt eftersom systemet mognar förväntar sig användarna att de senare versionerna är stabilare än tidigare versioner.

**Marketing Environment**

När ett system ska marknadsföras måste säljaren ta till hänsyn andra konkurrerande produkter, priset som kunden är villig att betala för systemet, och det krävda deadlinen för att leverera systemet. I en konkurrerande miljö kan mjukvaruföretag släppa sin produkt tidigt innan det blivit fullt testat och debuggat därför att de vill vara först ut på marknaden. Är en produkt väldigt billigt kan användare tolerera en lägre nivå av pålitlighet.

**5.3 QUESTION: Program inspections (3 points)**

Sommerville provides three advantages of program inspections. What are they?

1. Under testning kan ett fel gömma andra fel. När ett fel leder till oväntad output så kan du aldrig vara säker på om senare output avvikelser är på grund av ett nytt fel eller och det är en bieffekt av orginal felet. Eftersom inspectionsprocessen är statisk så behöver man inte oroa sig för interaktioner mellan fel. En inspektion kan upptäcka flera fel i systemet.
2. Ofullständiga versioner av systemet kan inspekteras utan extra kostnader. Om programmet är ofullständigt kan det behövas utvecklas speciella testfall för att testa de delar som är fullständiga, vilket självklart ökar utvecklingskostnaderna.
3. Utöver att söka efter brister i programmer kan en inspektion leda till högre kvalitet i koden. Eftersom man kanske upptäcker saker som inte följer standarden eller är ineffektivt eller dålig programmeringsstil, vilket kan leda till att systemet blir svårt att underhålla och uppdatera.

**5.4 QUESTION: Equivalence partitioning (2 points)**

Give an account of equivalence partitioning within testing.

Equivalence partitioning är en teknik för att testa mjukvara som går ut på att dela upp input data för en mjukvaruenhet till olika partitioner av ekvivalenta data vilket som man sen härledder test cases för de olika partitionerna. Den här tekniken försöker definiera testcases som avslöjar klasser av fel och därmed minskar det totala antalet test cases som måste utvecklas. En fördel med den här metoden är att tiden det tar att testa mjukvaran minskar eftersom antalet test cases är mindre.

**5.5 QUESTION: Component testing (4 points)**

There are different types of interface between program components. These are (1) parameter interfaces, (2) shared memory interfaces, (3) procedural interfaces, and (4) message passing interfaces. Explain them.

**Parameter Interfaces**

Parameter gränssnitt passerar data eller funktion referenser mellan två olika komponenter. Metoder hos ett objekt har ett parameter gränssnitt..

**Shared Memory Interfaces**

Dessa är gränssnitt som delar block av minen mellan komponenter. Data är placerat i minnet av ett subsystem och hämtat från minnet av ett annat subsystem. Den här typen av gränssnitt är ofta använda inom inbyggda system, där sensorer skapar data som sedan hämtas av andra systemkomponenter för att processeras.

**Procedural Interfaces**

Det här är gränssnitt där en komponent inkapslar ett set av procedurer som kan bli tillkallade av en annan komponent. Objekt och återanvändbara komponenter har den här typen av gränssnitt.

**Message Passage Interfaces**

Dessa är gränssnitt där en komponent kan efterfråga en service från en annan komponent genom att skicka ett meddelande till den. Ett svarsmeddelande inkluderas i resultatet av exekveringen av servicen. Några objekt-orienterade system har den här typen av gränssnitt, så väl har client-server system.

**5.6 QUESTION: Component testing (4 points)**

There are different classes of interface errors. Sommerville lists (1) interface misuse, (2) interface misunderstanding, and (3) timing errors. Explain them.

**Interface Misuse**

En komponent anropar en annan komponent och gör ett fel i användandet av gränssnittet. Den här typen av fel är vanligt med parameter gränssnitt, där parametrar kan vara av fel typ eller passerade i fel ordning eller fel antal passerade parametrar.

**Interface Misunderstanding**

En komponent som anropar en annan komponent kan missförstå specifikationen av dess gränssnitt och gör antaganden om dess beteende. Eftersom den kallade komponenten inte beter sig som förväntat skapar det också uförväntat beteende hos komponenter som gjorde anropet. Till exempel så kan en binarysök metod bli tillkallad med en oordnad array vilket får sökningen att misslyckas.

**Timing Errors**

Det här förekommer i realtidssystem som använder ett delat minnen eller message passing interfaces. Skaparen av data och förburkaren av data kan operera på olika hastigheter. Om inte det här behandlas i gränssnitts designen kan förbrukaren få tillgång till föråldrad information eftersom skaparen inte har hunnit med att updatera den delade gränssnitts informationen.

**5.7 QUESTION: Interface testing (2 points)**

Why is interface testing difficult?

Testa för gränssnittsfel är ofta svårt eftersom vissa gränssnittsfel uppstår bara under ovanliga tillstånd. T.ex ett objekt som implementerar en kö med fixad längd. Ett anropande objekt kan anta att kön är implementerad med en oändlig längd och tittar därför inte efter overflow på kön när den lägger till något. Det här tillståndet kan bara bli upptäckt under testning av ett test designat att forcera overflow på kön och få overflow händelsen att korruptera objektets beteende.

Ännu ett till problem kan uppstå på grund av interaktioner mellan felaktiga moduler eller objekt. Andra fel i objekt kan bara upptäckas när ett annat objekt beteer sig på ett oförväntat sätt. Exempelvis ett objekt som anropar ett annat objekt för att ta emot ett svar och antar att svaret är korrekt. Om det svarande objektet är felaktigt på något sätt kan returvärdet vara giltigt men inte korrekt. Det här är inte omedelbart upptäckbart men blir rätt tydligt senare när en beräkning blir fel.

**5.8 QUESTION: Test-driven development (4 points)**

Given an account of test-driven development. What are its fundamental steps? (2 points). Mention at least its two benefits. (2 points)

Test-driven development betyder att man utvecklar kod inktrementellt och vid sidan om gör man tester för varje steg i utvecklingen. Man går inte vidare till nästa steg förrän alla kod har passerat sina tester.

Test-driven utveckling följer dessa 5 steg:

1. Börja med att indentifiera de olika delarna av funktionaliteten som ska implementeras i varje steg. Det är ofta små och implementerade i få rader kod.
2. Skriv tester för den funktionaliteten och implementera det som ett automatiserat test. Vilket betyder att testet kan exekveras samt rapportera om testet blev godkänt eller ej.
3. Kör sen alla tester som har blivit implementarade. I början kommer du inte ha implementerat funktionalitetn så de nya testerna kommer att misslyckas. Det här är med meningen för att visa att testet utökar ditt test set.
4. Implementera sedan funktionaliteten och kör testerna än en gång. Det här kan inkludera refactoring av existerande kod för förbättring eller lägga till ny kod till vad som redan finns.
5. När alla tester blir godkända fortsätter man implementera nästa bit av funktionalitet.

Ett starkt argument för test-driven development är att det hjälper programmeraren klargöra sina ideer om vad ett kodsegment ska åstadkomma. För att kunna skriva ett test måste man förstå intentionen, och förstår man det blir det enklare att skriva koden.

**Code coverage:** I princip ska varje kodsegment som skriv ha minst ett associerat test. På så sätt kan du vara säker att all kod i systemet har faktiskt blivit exekverad. Kod testas samtidigt som den skriv vilket gör att fel upptäcks tidigt i utvecklingsprocessen.

**Regression testing:** Tester är framtagna samtidigt som programmet utvecklas. Det är därför möjligt att alltid gå bakåt för att köra tester och se om förändringar till programmet har introducerat buggar.

**Simplified debugging:** När ett test misslyckas ska det vara uppenbart var problemet ligger. Den nyligen skrivna koden ska kontrolleras och modifieras. Det finns inget behov av att använda debugging verktyg för att finna problemet.

**System documentation:** Testerna i sig själv fungerar som en sorts dokumentation som beskriver vad koden har för funktion. Läsa testerna kan göra det enklare att förstå koden.

**5.9 QUESTION: User testing (1,5 points)**

Sommerville distinguishes three types of user testing. What are they and what do they imply?

1. Alpha testing: Användare av mjukvaran arbetar tillsammans med utvecklingsteamet för att testa mjukvaran på utvecklarnas plats.
2. Beta testing: En release av mjukvaran som är tillgänglig för användarna för att de ska kunna experimentera och ta upp problem de finner till systemutvecklarna.
3. Acceptance testing: En kund testar ett system för att bestäma om det är redo för användning i kundens miljö eller ej.

**5.10 QUESTION: Acceptance testing (6 points)**

Sommerville identifies six stages in the acceptance testing process. These are (1) define acceptance criteria, (2) plan acceptance testing, (3) derive acceptance tests, (4) run acceptance tests, (5) negotiate test results, and (6) reject/accept system. Give an account of these phases.

**Define Acceptance Criteria**  
Det här ska göra så tidigt som möjligt. Det ska vara en del av systemkontraktet mellan kunden och utvecklaren. I praktiken kar det vara svårt att definiera kriterier så tidigt i utvecklingsprocessen. Detaljerade krav kanske inte är tillgängliga och det kan undergå stora ändringar i kraven under utvecklingsprocessen.

**Plan Acceptance Testing**

Planera hur resurser, tid och budgeten för acceptance testning ska se ut samt ta fram ett testschema. Acceptance test planen ska också ta upp nödvändiga täckningen av krav och ordningen som systemfunktioner ska testas. Risker ska definieras för testprocessen så som systemkrasch eller bristfällig prestation, samt ta upp hur dessa risker kan undvikas.

**Derive Acceptance Tests**

När acceptance kriterier har etablerats ska tester deisgnas för att kontrollera vare sig ett system är godkänt eller ej. Acceptance tester bör sikta in sig på att testa både funktionalitet och icke-funktionalitet (reliablity, performance) inom systemet. Testerna ska helst täcka alla systemkrav.

**Run Acceptance Test**De överrenskomna acceptance testerna exekveras på systemet. Helst ska det här göras i miljön som systemet kommer att använda i. Det kan dock visa sig vara opraktiskt och störande, därför sätter man ofta upp ett användartestmiljö för att köra dessa tester. Det är svårt att automatisera den här processen eftersom delar av acceptance tester kan innehålla tester med interaktioner mellan slutanvändare och systemet. Ibland krävs det utbildning för slutanvändaren.

**Negotitate Test Results**

Det är väldigt osannolikt att alla definierade acceptance tester kommer godkännas utan några problem. Men om så är fallet är acceptance testerna avklarade och systemet kan lämnas över. I de flesta fall kommer problem att upptäckas och i sådana fall får utvecklaren och kunden förhandla om systemet är bra nog för att användas eller ej.

**Reject/Accept System**

Ett möte mellan utvecklare och kund hålls där kunden bestämar om systemet ska bli accepterat eller ej. Är systemet inte tillräckligt bra för användning, är utvecklaren bunden till att vidar utveckla systemet. När det är klart repeteras acceptance testningsfasen.

**6. Project Management**

**6.1 QUESTION: Differences with other engineering projects (3 points)**

Most projects have important goals such as

1. Deliver the software to the customer at the agreed time.

2. Keep overall costs within budget.

3. Deliver software that meets the customer’s expectations.

4. Maintain happy and well-functioning development team.

These goals are not unique to other engineering projects. However, software engineering differs in a number of ways from other engineering projects. List and explain three such differences.

**The product is intangible.**

Mjukvara är inte begripbart på samma sätt som till exempel konstruktionen av ett hus. Halkar man efter i tidplaneringen kommer det att synas på konstruktionen. Eller om man missar något stort kommer det vara väldigt uppenbart för vem som helst. Men med mjukvara kan man inte see eller röra. Det är inte lika simpelt att se framstegen som att bara titta på det utan mjukvaruutvecklare måste förlita sig på bevis.

**Large software projects are often ‘one-­off ’ projects.**

Stora mjukvaruprojekt är ofta annorlunda på många sätt än tidigare projekt. Därför, även för en veteran som har stor erfarenhet från tidigare projekt kan tycka det är svårt att förutse problem. Utöver det så ändras teknologin inom datorer och kommunikation som kan göra ens kunskaper meningslösa. Lärdommar från tidigare projekt kan inte alltid tas över till nya projekt.

**Software processes are variable and organization­specific.**

Mjukvaruingenjörsprocessen är inte lika väldefinierad och underförstådd som i andra fält. Tillskillnad från mjukvaruutveckling där processen kan skilja sig stort från en organisation till en annan. Dock har stora framsteg inom process standardiseringen gjorts men vi kan fortfarande inte förlitligt förutse problem inom utvecklingen. Det här är specielt sant för mjukvarusystem som är en del av ett större projekt.

**6.2 QUESTION: Risk management (4 points)**

Give an account of a risk management process and its phases.

1. Risk Identification: Identifiera möjliga projekt-, produkt- och affärsrisker
2. Risk Analys: Fastställ sannolikheten och konsekvenserna av dessa risker
3. Risk Planning: Planera för riskerna, antingen undvik risken helt eller minimisera dens effekt på projektet.
4. Risk Monitoring: Gör regelbundna bedömningar av riskerna och dina planer för riskminimering och revidera dessa när du veta mer om riskerna.

**6.3 QUESTION: Risk types (4 points)**

Sommerville categorizes risks into three groups:

1. Project risks
2. Product risks
3. Business risks.

Explain and provide an example of each risk category (3 points). Why do the risk types overlap sometimes? (1 point)

**Project Risks:** Risker som påverkar projektschemat eller projektresurser. Ett exempelt är förlusten av en erfaren designer. Att hitta en ersättare med lämpliga kunskaper och erfarenheter kan ta lång tid och som konsekvens kan mjukvarudesignen ta längre tid att göra klar.

**Product Risks:** Risker som påverkar kvaliteten eller prestationen av mjukvaran. T.ex dåliga algoritmer som påverkar hastigheten av mjukvaran och gör den långsammare än förväntat.

**Business Risks:** Risker som påverkar organisationen som utvecklar mjukvaran. Exempelvis att en konkurrent lanserar en ny produkt. Introduktionen av en konkurrerande produkt kan leda till att förutsägelserna om försäljning blir väldigt optimistika då de gjordes när marknanden var tom.

Det är enkelt att en typ av risk påverkar en annan typ. Till exempel om en erfaren programmerare skulle försvinna kommer det påverka tidsschemat eftersom det tar tid att hitta en ny programmerare och för programmeraren att bli produktiv. Det räknas då som en projekt risk, men skulle den nya programmeraren inte vara lika skicklig som hans företrädare kan han producera lägre kvalitet på koden vilket skulle räknas som en produkt risk. Det skulle även kunna visa sig vara en affärsrisk då den föregångna programmerarens skickligheter kanske skulle vunnit dom ett nytt kontrakt.

**6.4 QUESTION: Risk strategies (3 points)**

The risk planning process considers each of the key risks that have been identified and develops strategies for managing these risks. Sommerville identifies three strategies. What are they? Provide an example for each strategy.

**Avoidance Strategies:** Genom att följa dessa strategies minskar sannolikheten att risker kommer uppstå. Till exempel en risk avoidance strategy är att ersätta potentielt defekta komponenter med nyinköpta komponenter som är pålitliga.

**Minimization Strategy:** Genom att följa dessa strategier blir effekten av risker mindre. Exempelvis genom att överlappa olika teams arbeten så att folk förstår vad ens kollegor arbetar med kan någon plocka upp en persons arbete om någon skulle bli sjuk.

**Contingency Plans:** Genom att följa dessa strategier är du beredd på det värsta och har en strategi för sånna situationer. Exempelvis på en Contingency plan skulle vara att förbereda ett kortfattat dokument för högt uppsatt anställda som visar hur projektet gör en viktigt insats för organisationens affärsmål och varför det inte skulle vara kost-effektivt att minska projektets budget.

**6.5 QUESTION: Planning stages (3 points)**

Project planning takes place at three stages in a project lifecycle. Describe these stages.

1. Vid the proposal stage, när man budgiver för ett kontrakt för att utveckla eller tillhandahålla ett mjukvarusystem. Behöver man en plan vid det här steget för att hjälpa en bestämma om du har tillräckligt med resurser för att avsluta arbetet och för att bestämma priset som du ska sätta för kunden.
2. Under projekt start-up fasen, när man planerar vem som ska arbeta på projektet, hur projektet ska brytas ner i inktrement-steg, hur resurser ska fördelas i företaget, etc. Här har man tillgång till mer information än i proposal stadiet, och kan därför inte förfina de första kostnads bedömningarna som gjordes.
3. Periodiskt utgenom projektets gång, när man modifierar planerna utifrån den kunskap och information man samlat under projektets arbete. Du lär dig mer om systemet som implementeras och förmågorna hos utvecklingsteamet. Den här informationen tillåter dig att göra noggrannare bedömningar av hur lång tid arbetet kommer att ta. Dessutom är det troligt att mjukvarukraven kommer att förändras vilket i de flesta fallen betyder att arbetsfördelningen måste ändras och schemat förlängas. Inom traditionella utvecklingsprojekt betyder det att planen skapad under start-up fasen måste modifieras. Däremot vid en agile-metod är planerna kortare och ändras kontinuerligt med mjukvarans utveckling.

**6.6 QUESTION: Project scheduling (5 points)**

Give an account of a project scheduling process.

Projekt schemaläggning är processen av att bestämma hur arbetet i projektet ska organiseras som olika arbetsuppgifter samt när arbetsuppgifterna ska utföras. Man bedömer hur lång tid som behövs för att utföra varje arbetsuppgift, vilken insats som krävs, och vem som jobbar med arbetsuppgifterna som blivit identifierade. En bedömning om hur mycket resurser som behövs för att göra klart vardera arbetsuppgift. Schemat som tas fram här blir sedan förfinat och modifierat under utvecklingens gång.

**6.7 QUESTION: Agile planning (5 points)**

Most of the agile approaches such as Scrum and extreme programming have a two-stage approach to planning. What are the two stages (2 points) and how do they correspond to traditional planning (2 points)?

1. Release planning: Tittar fram flera månader och bestämmer vad som ska inkludras i utsläppet av systemet.
2. Iteration planning: Tittar fram bara 2-4 veckor och fokuserar på att planera nästa inkrement av systemet.

**7.**

**7.1 QUESTION: Managing people (5 points)**

The people working are the greatest assets in a software organization. Therefore, to ensure that the organization gets the best possible return on investment, it is important to manage people with respect.

Explain and motivate why it may not always be optimal to assign software engineers to the roles of project managers? (2 points)

There are four critical factors that must be considered while managing people. These are *consistency, respect, inclusion* and *honesty*. (3 points). Explain and motivate three of them.

Mjukvaruingenjörer har ofta starka tekniska skickligheter men saknar ofta de mjukare skickligheterna som tillåter dem att motivera och leda ett projektutvecklingsteam. Som en projektledare ska man vara medveten om potentiella problem inom “people management” och borde försöka utvecklad sina “people management” skickligheter.

**Consistency:** Folk i ett projektteam bör bli behandlade på ett likvärdigt sätt. Ingen förväntar sig att alla ska få samma belöningar men folk ska inte känna att deras bidrag till organisationen ska kännas undervärderad.

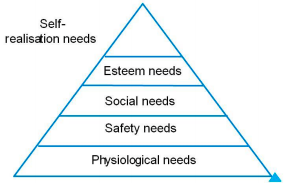
**Respect:** Olika personer har olika skickligheter och managers ska respektera dessa skillnader. Alla medlemmar i ett team ska få en chans att bidra.

**Inclusion:** Folk bidrar mer effektivt när de känner att någon lyssnar på deras synvinkel och frågar om deras förslag. Det är viktigt att utveckla en arbetsmiljö där alla synvinklar, även hos de nyaste anställda.

**Honesty:** Som en manager bör du alltid vara ärlig om vad som går bra och vad som går dåligt i ett team. Du bör också vara ärlig om din tekniska kunskap och villig att hänvisa till personal med mer kunskap om nödvändigt. Om du försöker dölja okunskap eller problem kommer det så småningom avslöjas och du förlorar respekt inom gruppen.

**7.2 QUESTION: Motivating people (5 points)**

Project managers need to motivate their people. Using Maslow’s hierarchy of human needs, explain what employee needs project managers should focus on and how they may be satisfied.



Folk som jobbar inom mjukvaruutvecklingsorganisationer är ofta inte hungriga, törstiga eller fysiskt hotade av deras miljö. Därför är det viktigt att folks sociala, värdering, och självförverkligande behov möts från ett people management perspektiv.

1. För att uppfylla sociala behov måste du ge personer tid att möta deras kollegor och tillhandahålla plats att träffas på. Det är relatit enkelt när alla medlemmar av ett utvecklingsteam arbetar på samma stället men mycket svårare när medlemmar inte är lokerade i salla byggnad eller ens samma stad. De kan ibland arbeta för en annan organisation eller hemmifrån. Bra sätt att börja projekt med är att arrangera möten där folk träffas ansikte mot ansikte och agerar direkt med varandra. Genom den här interaktionen blir folk del av en social grupp och accepterar mål och prioriteringar inom den gruppen.
2. För att uppfylla värderings behov, måste man visa att folk är värdefulla för organisationen. Publika beröm av folks prestation är ett simpelt sätt men fortfarande effektivt sätt att gör det på. Självklart måste folk känna att deras lön motsvarar deras insats och skicklighet.
3. För att uppfylla folks självförverkligande behov, måste man ge folk ansvar för deras arbete och tilldela dem utmanande uppgifter samt erbjuda träningsprogram där folk kan utveckla deras skickligheter. Träning är en viktigt motiverande influens eftersom folk gillar att lära sig nya saker och utöka deras kunskaper.

**7.3 QUESTION: Personality types (5 points)**

Frågan är försvunnen? Finns inte i PDF.

**7.4 QUESTION: Personality types (4 points)**

There are three different personality types that must be considered when choosing team members.

* What are they? (1 point)
* Explain how each personality type is motivated while doing the work? (1 point)
* What personality types would you choose when creating a team? (1 point). Motivate why? (1 point).
* In case it is impossible to choose people with the right personality types, how should you manage people so that the organizational and group objectives are met? (1 point)

**Task-Oriented:** Är motiverade av arbetet de utför. Inom mjukvaruutveckling är dessa folk motiverade av den intellektuella utmaningen av mjukvaruutveckling.

**Self-Oriented:** Är motiverade av deras personliga framgång och ära. De är intresserade av mjukvaruutveckling som ett verktyg att uppfylla deras egna mål. Det betyder inte att self-oriented personer är själviska och tänker bara på deras egna intressen. Snarare har de långsiktiga mål så som karriär framsteg, som motiverar dom och de önskar vara framgångsrika inom deras arbete för att uppnå dessa mål.

**Interaction-Oriented:** Motiverade av närvaron och handligar med kollegor. Som en mjukvaruutvecklare blir användarcentrerade och interaktionsorienterade individer mer inblandade inom mjukvaruingenjöring.

Jag hade valt personer av alla typer då de alla har sina fördelar och nackdelar, på så sätt kompletterar dom varandra.

Skulle det inte var möjligt att få med alla personlighetstyper är det viktigt för projektledaren att ha kontroll över gruppen så att individuella mål inte går i förhand över organisations- eller gruppmål. Sån kontroll är enklare att uppnå om alla gruppmedlemmar deltar i varje steg av projektets gång. Individuella initiativ är mest likt att hända när grupp medlemmar blir tilldelade uppgifter utan att vara medvetna om deras större roll i projektet.

**7.5 QUESTION: Cohesive groups (5 points)**

It is important to create a team that has the right balance of personalities, experience and skills. This however does may not always lead to successful and cohesive groups.

What does it mean that the group is cohesive? (2 points).

What are the benefits of a cohesive group? (3 points)

I en sammanhängande grupp är gruppens vision och mål viktigare än individerna som utgör gruppen.

1. Gruppen kan etablera deras egna kvalitetsstandarder. Eftersom dessa standarder är grundade av ett konsensus är det mer sannolikt att dom observeras istället för externa standarder som påtvingas av gruppen.
2. Individer lär och stöder varandra.
3. Kunskap är delad. Om en gruppmedlem lämnar kan arbetet pågå, eftersom andra i gruppen kan överta kritiska uppgifter och försäkra att projektet inte avbryts.
4. Refactoring och kontinuerlig förbättring är uppmuntrat. Gruppmedlemmar arbetar gemensamt för att leverera hög-kvalitets resultat och fixa problem, oavsett individen som skapade designen eller programmet.

**7.6 QUESTION: Group communications (5 points)**

It is essential that group members communicate effectively and efficiently with each other and other project stakeholders.

However, the effectiveness and efficiency is influenced by (1) group size, (2) group structure, (3) group composition, (4) the physical work environment, and (5) the available communication channels.

Motivate why these issues impact group communication?

1. **Group Size:** Allt eftersom en grupp blir större blir det svårare för medlemmar att kommunicera effektivt. Redan på en grupp av storlek 8 finns det 56 olika envägs-kommunikations länkar.
2. **Group Structure:** Personer i en mer informellt strukturerad grupp kommunicerar mer effektivt än grupper i en formel, . I en hierarkiskt struktur tenderar kommunikation att gå upp eller ner i hierarkin. Personer på samma nivå kan komma att knappt prata med varandra.
3. **Group Composition:** Folk med samma personlighetstyper kan komma i konflikter vilket leder till att kommunikation förhindras. Kommunikation är oftast bäst i könsblandade gruppern, än singel-köns grupper.
4. **The Physical Work Environment:** Arbetsplatsen är en väldigt stor faktor när det kommer till kommunikation. Om inte alla har gemensam arbetsyta och sitter och jobbar avskillt eller platser där de kan träffas på blir ingen bra kommunikation inom gruppen.
5. **The Available Communication Channels:** Det är många olika former av kommunikation, ansikte mot ansikte, e-mail, formella dokument, telefon eller sociala nätverk. Allt eftersom projektteam blir mer och mer utspridda med gruppmedlemmar som jobbar på distans är det viktigt att använda den teknologin som finns för kommunikation.

**8. Quality Management**

**8.1 QUESTION: Quality in manufacturing (3 points)**

What are the fundamentals of quality management in manufacturing industry (1 point) and why are they not directly comparable with software quality (2 points)?

En av de fundamentala sakerna med quality management inom tillverkningsindustrin är slutprodukten är jämförbar med dess satta specifikationer. Självklart kommer inte produkter exakt stämma överrens med deras specifikationer så en nivå av tolerans finns och produkter som stämmer helt överens blir godkända. Men inom mjukvara är slutprodukten inte alltid direkt jämförbart och ideen om tolerans finns inte på samma sätt.

På grund av följande anledningar är det omöjligt att komma till ett objektiv beslut om mjukvarusystemet möter specifikationerna.

1. Det är svårt att skriva fullständiga och klara mjukvaruspecifikationer. Utvecklare och kunder kan tolka krav på olika sätt och det kan bli omöjligt att komma överrens om systemet möter kraven eller ej.
2. Specificationer tar ofta in krav från flera olika klasser av intressenter. Det är svårt att undvika kompromiser och få med alla intressenters krav. De som inte fick med sina krav kan tycka att systemet är av dålig kvalitet även fast det innehåller alla överrenskommna krav.
3. Det är omöjligt att mäta olika särdrag (t.ex maintainability) direkt och kan därför inte specifieras på ett klart sätt.

**8.2 QUESTION: Process-based quality (3 points)**

Give an account of process-based quality and its phases in the manufacturing context (2 points). Is there any clear link between process and product quality in manufacturing? Motivate! (1 point).

En tillverkningsprocess involverar definera processen, konfigurera, sätta upp och operera maskiner involverade i processen. När maskinerna väl fungerar som de ska produceras produkter och deras kvalitet blir bedömd. Är inte kvaliteten på den nivån man vill uppnå ändrar man i processen tills man uppnår önskad kvalitet.

Länken mellan process och produkt kvalitet är att processen är relativt enkel att standardisera och bevaka. När ett tillverkningssystem är kalibrerat kan dem köra igen och producera hög-kvalitetsprodukter.

**8.3 QUESTION: Software standards (3 points)**

There are three reasons why software standards are important. What are they? (3 points)

1. Standarder fångar visdom som är värdefullt för organisationen. Dom är baserade på kunskap om det bästa och mest lämpliga arbetssättet för företaget. Den här kunskapen är ofta erhållen efter en stor bit av “trial and error”. Förvandla det till en standard hjälper företagen att återanvända den här erfarenheten för att undvika tidigare misstag.
2. Standarder tilldelar ett ramverk för att definera vad “kvalitet” betyder i ett visst sammanhang.
3. Standarder hjälper kontinuitet när arbete utförs av en person som sen fortsätts av en annan person. Standarder försäkrar att alla ingenjörer inom organisationen upptar samma praktiker. På så sätt krävs det mindre ansträngning att lära sig när man startar nytt arbete.

**8.4 QUESTION: Quality management & software development (2 points)**

Should or should not quality management and software development be separated? Motivate your answer! (1 point) When is it practically impossible to separate them in smaller companies? Motivate your answer! (1 point)

Quality management borde vara självständingt och rapportera till management över projekt management nivån. Därför att projekt ledaren måste underhålla projekt budgeten och schemat. Om problem uppstår kan projekt ledaren vara frestad att äventyra produkt kvalitet så att de möter tidschemat eller budgeten. Ett självständigt “quality management team” försökrar att organasitons mål om kvalitet är inte äventyrat av kortsiktiga budget och schema problem.

Inom små företag blir väldigt svårt att separera de båda då det ofta är samma person/personer som håller i projekt management och quality management.

**8.5 QUESTION: Reviews (5 points)**

Reviews are quality assurance activities that check the quality of project deliverables. Give an account of a software review process and its phases in a traditional software development context (3 points).

What does the review process look like in agile development (1 point)?

In what context (traditional or agile) is it more appropriate to use reviews, motivate your answer (1 point).

1. Pre-review Activities: Förberedande aktiviteter som är grundläggande för att en review ska vara effektiv. En pre-review aktivitet är planering som involverar att sätta ihop ett review team, arrangera en tid och plats för review och dela ut dokument som ska bli reviewed.
2. Review Meeting: Ett möte hållls med review teamet och skaparen av dokumentet eller mjukvaran som ska granskar. Skaparen går igenom dokumentet/mjukvaran och kommentarer och beslut dokumenteras.
3. Post-review Activites: Efter ett review möte har slutat måste frågorna och problemen bli addresserade. Det kan betyde att fixa ett bug, refactor mjukvara eller skriva om dokument. Ibland krävs det ännu en review för att kontrollera att alla ändringar täcker de tidigare review kommentarerna.

Review processen i agil utveckling är oftast informell. I Scrum hålls ett review möte efter varje iteration av mjukvara som blivit klart (Sprint Review), där kvalitetsfrågor och problem disskuteras. I extreme programming så säkerställer par programmering att koden granskas konstant av en annan team-medlem.

Avsaknaden av formella kvalitets procedurer i agila metoder betyder att det kan bli problem när man använder ett agilt tillvägagångssätt i företag som har utvecklat detaljerade quality management procedurer. Quality review kan sakta ner hastigheten på mjukvaruutveckling och är bäst använt inom plan-driven utvecklingsproces. I en plan-driven process kan reviews bli plannerade och annat arbete bli schemalagt parallelt med dem. Vilket är opraktiskt i agila tillvägagångssätt där fokus ligger på kod-utveckling.

**8.6 QUESTION: Inspections (2 points)**

Despite of the well-known cost effectiveness, many software development companies are reluctant to use inspections or peer reviews.

Motivate why inspections are effective (1 points) and explain why companies are unwilling to have inspections (1 point).

Program inspektioner är peer-review där gruppmedlemmar sammarbetar för att hitta buggar i programmet som utvecklas. Inspektioner komplementerar testning då dom inte kräver att programmet körs. Det betyder att icke färdiga versioner av systemet kan bli verifierade och kontrollerade så de stämmer överrens med representaioner så som UML modeller.

Mjukvaruingenjörer med erfarenhet av programtestning är ibland ovilliga att acceptera att inspektioner kan vara effektivare för att upptäcka defekter än testning. Managers kan vara misstänksamma för inspektioner kräver extra kostnader under design och utveckling. De önskar inte ta risken att det inte blir några besparingar i programtestningen.

**9. Configuration and Version Management**

**9.1 QUESTION: CM defintion (2 points)**

What is configuration management and why is it important?  
Configuration management berör policys, processer och verktyg för att hantera ändringar inom mjukvarusystem. Det är nödväntigt att hantera utvecklande system eftersom det är enkelt att tappa bort vilka ändringar och komponent versioner som har tagits med i varje systemversion. Versioner implementerar förändringar, fel korrigeringar och support av ny hårdvara eller operativsystem. Det är viktigt eftersom flera versioner kan vara under utveckling samt användas samtidigt. Om man inte har ett bra konfigurations management kan man spendera tid och resurser på att modifiera fel version, leverera fel version av systemet till kunden eller glöma var källkod eller en komponent för en viss version är lagrad.

**9.2 QUESTION: CM activities ( 6 points)**

There are four principal configuration management activities:

· Change management

· Version management

· System building

· Release management

Choose and describe two of these activities.

1. **Change Management:** Håll koll på förfrågningar om ändringar av mjukvaran från kund och utvecklare, ta reda på kostnaden och effekten av dessa förändringar samt bestämma om och när förändringarna ska implementeras.
2. **Version Management:** Håll koll på multipla versioner av systemkomponenter och försäkra att ändringar till komponenterna av de olika utvecklarna inte skapar konflikter.
3. **System Building:** Processen av att bygga ihop program komponenter, data och bibliotek för att sen kompilera och länka ihop dessa till ett exekverbart system.
4. **Release Management:** Förbereda mjukvara för externa publiceringar och hålla koll på system versioner som har blivit släppt för publiktanvändande.

**9.3 QUESTION: Change management process ( 5 points)**

Describe the change management process.

Processen börjar med att någon (kund, användare eller utvecklare) skickar in en “change request” som beskriver vad som behövs ändras på i systemet. Det kan vara en bugrapport eller ny funktionalitet. Där efter kontrolleras förfrågan om den är valid eller ej. Det behövs för varje “change request” behöver inte alltid vidta några åtgärde. T.ex om det är någon som missförståt hur man använder programmet och rapporterar det som ett bug eller om någon föreslår den funktion som redans finns med utan deras vetskap. Om det är en valid “change request” blir nästa steg att bedöma kostnaden och effekten som förändringen innebär samt vilka komponenter som förändringen berör. Eftern analysen är klar är det upp till en annan grupp att bestämma om ändringen är kost-effektiv från en affärssynvinkel. Den här gruppen kallas Change Control Board (CCB).

**9.4 QUESTION: Change impact ( 3 points)**

Sommerville lists five factors that should be taken into account in deciding whether or not a change should be approved. List and describe three of them.

1. *The consequences of not making the change:* När man bedömer en change request måste man överväga vad som kommer hända om ändringen inte blir implementerad.
2. *The benefits of the change:* Är förändringen något som kommer att nytta många användare av systemet eller är det bara ett förslag som primärt kommer att nytta personen som gjorde förslaget?
3. *The number of users affected by the change:* Om ändringen bara berör ett få antal personer kan det få en låg prioritet. Det kan till och med bli oklokt att genomföra en förändring om det kan ha negativ effekt på majoriteten av systemanvändare.
4. *The costs of making the change:* Om förändringen påverkar många systemkomponenter eller tar lång tid att implementera kan det bli nekat, givet den stora kostnaden.
5. *The product release cycle:* Om det nyligen släppts en ny version av mjukvaran kan det vara förnuftigt att vänta med implementationen av förändringen till nästa planerad release.

**9.5 QUESTION: ( 1 point)**

What is Change Control Board and what is its function?

Change Control Board är en grupp som ansvarar för beslut om hur mjukvaran ska utvecklas. De granskar och godkänner alla förfrågningar om ändring, så vidare det inte är en liten okostsam korrigeringsändring. De överväger effekten ändringen har från en strategisk och organisationell synvinkel istället för från en teknisk synpunkt. De beslutar om ändringen i fråga är ekonomiskt rättfärdigad och prioriterar accepterade ändringar för implementation.

**9.6 QUESTION: (2 points)**

Is it ethical for a company to quote a low price for a software contract knowing that the requirements are ambiguous and that they can charge a high price for subsequent changes requested by the customers? Motivate your answer.

Nej det är inte speciellt etiskt gjort. Då man utnytjar kundens oerfarenhet inom utveckling av mjukvarusystem när man själv vet mycket bättre bara för att tjäna mer pengar.

**9.7 QUESTION: (8 points) Den här frågan kommer att ställas bara på första ordinarietentan.**

During the course project, you have been assigned a specific role, such as for instance, tester, developer, business manager, etc. Your task was to follow the responsibilities assigned to your role. Among many tasks, you were responsible for one or several process models.

From the perspective of your role and from the perspective of **only one** process, do the following:

· Present what role and responsibilities you have had (0 points).

· Describe the process model for which you have been responsible (1 points).

· List and describe the good sides of your process model (1 point).

o at least 2 good sides

· Motivate why these good sides have been achieved (1 point).

· List and describe the bad sides of your process model (1 point).

o at least 2 bad sides

· Motivate why these bad sides have been encountered (1 point)!

· Describe and evaluate the solutions that you have used for remedying the bad sides (1 point).

· Evaluate the book from the perspective of the process model you have been responsible for (2 points).

o Does the book cover the necessary details of the process? Motivate your answer!

o Does any other literature cover the necessary details of the process? Motivate your answer!

o Does the book correctly describe the process? Motivate your answer!

o Does any other literature describe the process? Motivate your answer!

o What suggestions would you provide to Ian Sommerville to improve his process descriptions? Motivate your suggestions!

o Finally, does the book properly describe the relationship between your process and other processes? Motivate your answer!

If you happen to be CEO, then you are welcome to choose any role.

Under kursprojektet, har du tilldelats en specifik roll, till exempel testare, utvecklare, affärschef osv. Din uppgift var att följa de ansvarsområden som givits till din roll. Bland många uppgifter, var du ansvarig för en eller flera processmodeller.

Ur din rolls och utifrån endast en process perspektiv, gör följande:

* Presentera vilken roll och ansvarsområden du har haft (0 poäng).
* Beskriv den processmodell för vilken du har varit ansvarig (1 poäng).
* Lista och beskriv de goda sidorna av din processmodell (1 poäng).
  + minst 2 goda sidor
* Motivera varför dessa goda sidor har uppnåtts (1 poäng).
* Lista och beskriv de dåliga sidorna av din processmodell (1 poäng).
  + minst 2 dåliga sidor
* Motivera varför dessa dåliga sidor har uppstått (1 poäng)!
* Beskriv och utvärdera de lösningar som du har använt för att åtgärda de dåliga sidorna (1 poäng).
* Utvärdera kursboken utifrån den process du har varit ansvarig för (2 poäng).
* Har boken täckt de nödvändiga detaljerna i processen? Motivera ditt svar!
* Har annan literatur täckt de nödvändiga detaljerna i processen? Motivera ditt svar!
* Har boken korrekt beskrivit processen? Motivera ditt svar!
* Har en annan literatur korrekt beskrivit processen? Motivera ditt svar!
* Vilka förslag skulle du ge till Ian Sommerville för att förbättra sina process­beskrivningar? Motivera dina förslag!
* Slutligen, har boken beskrivit relationen mellan din process och andra processer ordentligt? Motivera ditt svar!

Om du råkar vara VD, så är du välkommen att välja vilken roll som helst.

**Architect - 110% bullshit**

**Presentera vilken roll och ansvarsområden du har haft (0 poäng).**

Jag har haft rollen som architect om mitt ansvar har varit att designa arkitekten och strukturen för systemet som vi har utvecklat.

**Beskriv den processmodell för vilken du har varit ansvarig (1 poäng).**

Jag har skapat fyra olika architect vyer, en logiskvy, en processvy, en utvecklingsvy och en fysiskvy. De här vyerna har använts under utvecklingsprocessen.

Bestämt hur arkitekturen av systemet ska dokumenteras samt hur den ska evalueras.

**Lista och beskriv de goda sidorna (minst 2) av din processmodell (1 poäng)**

De olika vyerna har hjälp oss under utvecklingen av systemet.

Vi har fått en överrensstämmande struktur på det utvecklade systemet.

**Motivera varför dessa goda sidor har uppnåtts (1 poäng).**

Alla har tittat på samma vyer vilket har underlättat allas förstående för systemet och det har hjälpt oss att få en överrensstämmande struktur.

**Lista och beskriv de dåliga sidorna (minst 2) av din processmodell (1 poäng).**

Vet inte om vår lösning på strukturen är bästa lösningen.

Det har varit lite problem med dokumentera arkitekturen.

**Motivera varför dessa dåliga sidor har uppstått (1 poäng)!**

Det har varit svårt att evaluera strukturen av systemet på grund av liten erfarenhet och kunskap om design av arkitektur.

Ovana och liten erfarenhet har lätt till svag dokumentation av systemets arkitektur.

**Beskriv och utvärdera de lösningar som du har använt för att åtgärda de dåliga sidorna (1 poäng).**

Vi har haft möten och diskussioner om strukturen och arkitekturen för att åtgärda problemen med evalueringen av arkitekturen.

Dokumentationen har vi försökt så gott vi kan med men inte haft någon alternativ lösning.

**Utvärdera kursboken utifrån den process du har varit ansvarig för (2 poäng).**

Kursboken har varit användbar för att beskriva hur man skapar och gör det olika arkitektvyerna. Dock tar boken inte upp så mycket om evaluera eller dokumentera arkitektur.

**Har boken täckt de nödvändiga detaljerna i processen? Motivera ditt svar!**

Ja, allt som allt har boken varit till nytta för att hjälpa oss uppnå en fungerande struktur för vårat system. Då den har tagit upp hur man ska gå tillväga för att uppnå en bra systemarkitektur.

**Har annan literatur täckt de nödvändiga detaljerna i processen? Motivera ditt svar!**

Har inte läst någon annan literatur.

**Har boken korrekt beskrivit processen? Motivera ditt svar!**

Ja eftersom den har tagit upp de nödvändiga detaljerna i processen har den beskrivits korrekt.

**Har en annan literatur korrekt beskrivit processen? Motivera ditt svar!**

Har inte läst någon annan literatur.

**Vilka förslag skulle du ge till Ian Sommerville för att förbättra sina process­beskrivningar? Motivera dina förslag!**

Kanske inte just för Sommerville men boken var mer inriktat för stora företag vilket gjorde det lite problematiskt eftersom man bara var ett litet team.

**Slutligen, har boken beskrivit relationen mellan din process och andra processer ordentligt? Motivera ditt svar!**

Min arkitekturprocess har varit starkt sammanbunden med själva utvecklingsprocessen. Då mina strukturvyer har använts för att utveckla programmet.