Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Indholdsfortegnelse

Anders H. Poder
Mogens Habekost

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Introduktion

Dette dokument indeholder besvarelsen på opgave H5 i kurset Software Arkitektur i Praksis. Opgaven er delt i to afsnit, hvor det første har fokus på anvendelse af skyen og den anden på arkitekturel evaluering.

Part1

Dette afsnit omhandler anvendelse af skyen og er yderligere inddelt i nogle underafsnit.

1. A short essay of the challenges/obstacles/good experiences of deploying the TM12 server (and Mongo DB) in the cloud. Evaluate benefits/liabilities of cloud deployment both in this particular case as well as in a longer time perspective.
2. An exploration of the systems QAs compared to a local systems QAs, such as performance, modifiability, availability, testability, etc. Also look for anomalies (like uploads that take long time to appear during read, stolen CPU cycles, etc.)
3. A short discussion of benefits and liabilities of using cloud deployment for tele medicine in general. Are there any "show-stoppers"? Refer to Ambrust et al.
4. A short outline of what it would take to refactor TM12 to use a PaaS approach for cloud deployment instead, for instance, using Google App Engine.

1. Oplevelser i skyen

Oplevelser med opsætning

Vi har prøvet at deploye TM12 til Amazon på en Ubuntu server. Det viste sig at være yderst enkelt at oprette serveren efter beskrivelsen. Der var dog en enkelt ting. Man skal eksekvere en ”sudu apt-get update” før man installere java og ant. Selv Putty virkede uden problemer (efter vi lige læste brugervejledningen ordentlig igennem).

Det er overraskende hvor enkelt det er at få opsat en virtuel server og få den til at virke.

Overvejelser omkring anvendelse af ”skyen”

At lægge noget op i skyen har mange fordele og ulemper. Ligesom med alle andre services hvor man overvejer at anvende en tredjeparts leverandør.

1. Kan vi stole på leverandøren?
2. Er leverandøren stabil nok?
3. Kan leverandøren levere det produkt vi har behov for? Med hvilken QoS?
4. Kan vi stole på at leverandøren vil hjælpe os hvis der er noget der ikke virker?
5. Hvis vi beslutter at skifte leverandør kan vi så det? Hvordan gemmes data? Standard, API, …

Der er andre overvejelser der er specielle for skyen.

1. Hvilken del skal vi lægge op i skyen?
2. Er der nogle juridiske aspekter omkring ejerskab af data samt fysisk placering af data?
3. Hvilke udsving i QoS kan vi risikere (determinisme)?

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Som med alle leverandører er det vigtigt at man stoler på firmaet. Hvis der f.eks. er tale om data omkring et fremtidigt produkt, skal man jo være sikker på at udbyderen ikke vil stjæle ens data og sælge dem til en konkurrent. Der er selvfølgelig mulighed for at anvende en passende kryptering, men hvis man ikke stoler på sin leverandør, skal man nok overveje om der findes en alternativ leverandør.

Der er naturligvis forskel på firmaer. Hvis man ønsker at sikre en global og konstant adgang til ens data er det nok ikke en god ide at vælge et mindre eller nystartet firma eller et firma med usikre regnskaber eller egenkapital. Selv om det ren juridiske med rettigheder til data er i orden, vil det utvivlsomt resultere i en længere nede periode hvis firmaet pludselig lukker dørene og man skal til at finde en alternativ leverandør.

Her er det vigtigt at kikke på ens behov. Udbyder firmaet det produkt man har behov for, med den QoS og stabilitet der er behov for? Kan det tilbyde den skalering i ressourcer og båndbredde som der er behov for? Et firma der har mange kunder vil ofte have meget bedre mulighed for ressource sharing, og dermed for at supportere korte peek perioder eller op-skalering efter behov.

Når man overdrager ansvaret for dele af ens system til en anden har det naturligvis den fordel at det er dennes ansvar at sikre at det kører efter aftale. Det er som regel derfor man vælger en anden udbyder; fordi man enten ikke selv har ressourcer til det eller det er ufordelagtigt. Disse ressourcer kan være i form af penge, erfaring (der er jo nogle der skal drifte det) eller tid (det tager tid at starte op). Om det er fordelagtigt er mere et spørgsmål om hvem der kan gøre det billigst og bedst. Selv om vi har ressourcer til at oprette og drifte vores eget system, så er det meget tænkeligt at et firma der ikke laver andet end at udbyde skyer vil kunne gøre det billigere og bedre end vi selv kan – også selv om de skal tjene penge på det. Det er et simpelt spørgsmål om stordrift. Men der hvor det som regel giver knuder er når der er noget der ikke virker. Hvis man selv drifter det kan man smide alt hvad man har i hænderne, hyre de bedste konsulenter, og få det til at virke. Men, hvis man anvender Amazon's sky og man udgør 0,0001% af deres total omsætning, hvor meget vil de så lægge i at få løst ens problem, koste hvad det vil?

Data kan lagres på mange måder, og ofte er en sky mere end bare data, det er et specifikt API eller nogle virtuelle ressourcer der fungerer på en bestem måde. Set fra leverandørens synspunkt er det en fordel at gøre det så svært så muligt at skifte leverandør, så det er vigtigt som minimum danne sig et overblik over hvad det vil kræve at skifte leverandør, og hvilket risici det ville indebærer efter systemet er operationelt i skyen.

Når der er tale om at have data og/eller ressourcer i skyen er det vigtigt at også tænke interface. Meget ofte har firmaer nogle legacy systemer der er nødt til at holdes kørende. Kan disse overhovedet fungere i skyen? Hvis ikke, hvad vil det så betyde for QoS hvis disse systemer skal forblive internt, men data ligger i skyen (længere netværkskommunikation)? Skal vi købe et service API i skyen (SaaS), hvis et sådanne findes. Hvis et firma udbyder en end user application som passer til vores behov, så er det sandsynligvis både den hurtigste, nemmeste, mest stabile og billigste løsning. Dette skyldes at mange andre bruger samme løsning og det derfor er disse firmaers samlede "vægt" der afgør hvor hurtigt firmaet reagere på problemer. Samtidig vil API'et og performance være skræddersyet til denne løsning og er ikke generelle interfaces som så skal bruges til at implementere end user applikationen. Dette er sammenligneligt med at en general purpose architecture (GPA) på en CPU altid er mindre effektiv end en custom application direkte i silicium.

Findes der ikke en SaaS der er et perfekt match så er der måske nogle der udbyder et framework i en sky hvorpå vi kan bygge vores applikation (PaaS). Dette kræver mere arbejde, men frameworket gør at vi kan anvende nogle kendte og afprøvede moduler til at implementere vores applikation, og disse moduler vil være optimeret i forhold til den service de udbyder. Ikke så performance optimalt som SaaS, men mere fleksibelt og hurtigere end at skrive det selv. Endelig kan det være at vores applikationer så speciel at den skal skrives fra bunden, eller at vi vil forsøge at lægge en legacy applikation op i skyen. I det tilfælde kan vi købe nogle virtuelle ressourcer (Iaas), som vi så kan implementere vores applikation på. Dette tager naturligvis længere tid (hvis ikke vi har legacy koden i forvejen), og ansvaret for at applikationen fungerer og er god nok performancemæssigt påhviler fuldt og fast os selv, ligesom ressourcerne man køber er general purpose, hvilket betyder at de skal kunne bruges til "alt", og derfor ikke kan være optimerede mod en speciel brug.

Dette kan sammenlignes med at arbejde på et OS på en GPA (IaaS). Alternativt kan man skrive applikationen direkte på stålet ud fra et library fra producenten (PaaS). Og endelig kan man købe en chip der er skræddersyet til ens behov (SaaS).

Meget ofte er data ikke bare data. Der kan være tale om nationale hemmeligheder eller personfølsomme data, som er beskyttet af specielle juridiske regler for hvor og hvordan disse oplysninger fysisk skal lagres og slettes. Når man arbejder i skyen er man som udgangspunkt ligeglad med hvor ens data lagres, men disse juridiske aspekter kan gøre det nødvendigt at sikre at data fysisk lagres et bestemt sted, og at data når den slettes gøres efter de rigtige retningslinjer (f.eks. DoDs regler for data sletning). Hvis en server i en virtuel farm går ned flytter data og server øjeblikkeligt over på en anden server, da data gemmes redundant (raid), men hvad med den server der er gået ned? Hvis den kommer op og fungere igen, vil den så blive brugt til noget andet hvor data evt. kan genskabes af den nye bruger? Eller risikere vi at den fysiske server/disk smides på lossepladsen hvor dens data evt. kan genskabes? Alt efter de juridiske krav er det vigtigt at tage højde for disse ting i kontrakten.

Med virtuel placering af data i skyen kan det være meget svært at garantere svartider af to på hinanden følgende kald, specielt med load distribution og multiple servere. Udbyderen vil måske garantere en maximum svartid, men da det går imod ideen om skyen at holde tæt styr på den fysiske placering af de forskellige virtuelle servere. Ofte er det rigeligt at kende den maksimale svartid, men i specielle tilfælde kan det være vigtigt at svartiden er deterministisk, eller ikke varierer ret meget mellem forskellige servere. I dette tilfælde kan det være problematisk at anvende skyen.

2. QA sammenligning

Availability

En af de største fordele ved at anvende skyen er at data of services er tilgængelige overalt. Hvis man vælger en større udbyder med mange fysiske servere, vil det at anvende skyen ofte øge availability, da en fejl på en server ikke vil betyde noget, da den underliggende VM executer automatisk vil flytte eksekveringen uden at brugeren oplever nogen reduktion i availability. Det er dog vigtigt at man vælger den rigtige aftale med cloud udbyderen. Samtidig vil de fleste cloud udbydere have redundante forbindelse til internettet, som sikre høj forbindelses-availability.

Det er også vigtigt at undersøge den underliggende arkitektur i skyen. Anvender de en form for caching eller et specielt API, der gør at den givne sky opfører sig anderledes end man normalt ville forvente. Er der I/O caching som gør at data der lægges op i skyen først er tilgængelig efter et stykke tid, eller måske et ikke-intuitivt fil API som gør det svært at interagere med?

Modifiability

Hvis der er tale om at man lægger hele applikationen op i skyen (IaaS) så har det mere betydning hvordan man laver sin architectur end om det er i Skyen eller ej, men hvis man anvender PaaS eller SaaS så er man begrænset af det bibliotek (PaaS) eller API (SaaS) som cloud udbyderen stiller til rådighed.

Performance

Hvis der er tale om et system som TM12 der altid tilgås udefra vil performance i form af netværkskommunikationstid ikke blive påvirket af at lægge det op i skyen, faktisk er der en vis sandsynlighed for at det kan blive hurtigere, da en større cloud udbyder sandsynligvis vil have en meget kraftig forbindelse direkte til et internet back-bone, og dermed kan tilbyde en kommunikationshastighed og ping-tid som det vil være meget dyrt at skulle opbygge selv, specielt hvis man vælger en (eller flere) udbydere i nærheden af hvor data skal bruges. F.eks. hvis det skal bruges i Danmark vil man vælge en i Danmark eller som minimum i Europa.

Da cloud systemer mere eller mindre altid arbejder virtuelt skal dette naturligvis tages i betragtning, men moderne VM-Ware tilbyder performance der kan konkurrerer med eksekvering uden VM, så dette er ikke et stort problem mere, og der er mere et spørgsmål om at vælge den rigtige mængde ressourcer.

Security

Her kan der være vise problemer med at anvende skyen, da data ikke mere fysisk ligger på en server man selv kontrollere. Kryptering kan afhjælpe meget af dette, men juridisk kan der være krav til godkendelse for at man må opbevare personfølsomme oplysninger. Der bør altid laves en risikovurdering før data flyttes op i skyen.

Testability

Testability i Skyen er generelt sværere end på en lokal fysisk server, da man ikke har kontrol over det underliggende virtuelle eksekveringsmiljø. Hvis der er tale om SaaS eller PaaS, bliver dette problem større, da den underliggende applikation eller library ikke er tilgængelig for white-box test, og er der et problem i denne kode skal man vente på at cloud leverandøren retter fejlen. Fordelen er dog at da koden anvendes af mange andre er der en mindre risiko for at der er alvorlige fejl i koden.

Usability

Hvis der er tale om SaaS er det naturligvis vigtigt at vælge en cloud leverandør der kan præcentere et logisk og let anvendeligt interface. Samtidig får man foræret en del end user test, da andre kunder har brugt interfacet før. Samtidig kan det være en fordel at man anvender et look and feel som er velkendt i markedet, da genkendelighed er utrolig vigtigt for usability. Vi har alle prøvet at skifte bank og skulle bruge tid på at sætte sig ind i et nyt interface, eller indenfor programmering er der ikke noget mere irriterende end når producenter afviger fra defakto-standarden, som f.eks. .NET Array.Copy, som anvender ”source, destination” når alle andre siden C anvender ”destination, source”.

I TM12 kan det f.eks. være et stort problem hvis læger og ikke mindst brugere er blevet vandt til det eksisterende interface og så skal til at lære et nyt. Samtidig, hvis hundredvis af klienter ude hos borgerne er blevet kodet op mod et interface, kan det være kompliceret at omkoden dem til et nyt.

Hvis der er tale om PaaS er problemet sandsynligvis mindre, da det hovedsageligt er applikationen der bestemmer usability, og ikke skyen.

Anomalities

Det er klart at når ejerskabet af den underliggende arkitektur ligger hos en tredjepart er det svært, og ofte umuligt, at vide hvilke optimeringsalgoritmer de anvender. F.eks. lazy write. Hvis der anvendes en disk sync mekanisme er det tænkeligt at der kan være en forsinkelse mellem write og disk sync. En Cloud implementation opfører sig også lidt anderledes end en fysisk server. En virtuel server (eller server i skyen) deler en meget større CPU med mange andre servere. Den givne server har så lov til at anvende en andel af CPU'en, men brugen er meget sjældent jævnt fordelt, og systemet tillader derfor at man kan anvende flere CPU-cycler end man rent faktisk har til rådighed i en kort periode, så længe det i gennemsnit ligger under det tilladte. Det kan dog betyde at der i perioder er færre CPU cycler end forventet på den givne maskine.

3. Telemedicin i skyen

Indenfor telemedicin er der mange risikoområder, fra introduktionen af ny teknologi indenfor et områder der er notorisk konservativt, til installation hos en kundegruppe der er både konservativ og ofte udsat for både fysiske og mentale udfordringer som gigt, senilitet, stofmisbrug, m.m. Disse udfordringer vil dog eksistere uafhængigt af om der er tale om deployment i skyen eller på egne servere.

Ambrust et al omtaler forskellene mellem et konventionelt data center og skyen i en tabel. Ved at tage udgangspunkt i denne tabel og sammenholde med de behov der er i Telemedicin kan vi se om der er noget vundet ved at skifte til Skyen.

Advantages	Public cloud	Conventionel Data Center	Tele-Medicine
Appearance of infinite computing resources on demand	Yes	No	Tele-medicine is a closed user group, and its growth is highly deterministic. For this reason the ability to rapidly scale the resources is not a high priority, as there would always be time to buy more physical or virtual servers when scaling up.
Elimination of an up-font commitment by Cloud users	Yes	No	The investment in Tele-medicine is huge, and the cost of the computing resources on the server end negligible in comparison.
Ability to pay for use of computing resources on a short-term basis as needed	Yes	No	The resources required will fluctuate during the day, but with

			the exception of an audit, the fluctuations are minor and predictable, and huge short-term fluctuations, like might be experienced on a public server when a new add-campaign runs, are not possible (except for errors).
Economies of scale due to very large data centers	Yes	Usually not	
Higher utilization by multiplexing of workloads from different organizations	Yes	Depends on company size	This could be interesting, primarily due to the field of interest. Higher utilization means less resource waste and therefore a greener image. Due to the deterministic and relatively limited fluctuations of the load, it is however unlikely that this will be more than a PR gimmick.
Simplify operation and increase utilization via resource virtualization	Yes	No	Scaling must be possible, even if it does not need to be fast, and the design of the application must therefore be layered and follow the general recommendations for scalable software (see Design Guidelines for Application Performance). With that said the scaling may as well be done on an in-house data center or an external conventional data center.

Baseret på ovenstående kan man konkludere at der ikke er tunge grunde til at vælge skyen, men omvendt, er der så nogle tunge grunde til ikke at vælge skyen, da ovenstående hovedsageligt konkluderer at der ikke er vundet noget, men er der tabt noget ved at vælge skyen?

Ud over at overveje om det er en god ide, Ambrust et al omtaler 10 mulige forhindringer ved at skifte til skyen. Disse vil vi også sammenholde med Telemedicin.

Forhindring	Telemedicin
Availability/Business Continuity	Det er klart at det er vitalt vigtigt at systemet er tilgængeligt når patienter og læger har brug for det. Hvis man vælger en større cloud udbyder med en god egenkapital og stabile regnskaber, vil det dog give mindst ligeså god availability/Business continuity som at vælge et data center med samme krav, eller selv oprette et med dertilhørende afhængighed af internet udbyder. Det kan dog anbefales at anvende en kombinationsløsning med et eksternt data center eller cloud udbyder kombineret med et mindre internt data

	center som kan håndtere de vitale funktionalitetet i tilfælde af af data centeret eller cloud udbyderen fejler.
Data Lock-In	Ved at anvende skyen er man underlagt udbyderens API. Der er visse regler til hvordan data skal være tilgængeligt, f.eks. OIOXML, men med data transformering og backup af data udenfor udbyderen vil dette ikke være mere problematisk end at anvende et data center. Det største problem her vil være den ekstra udvikling påkrævet for at supportere flere API'er (multiple parallelle skyer/data centre med forskellig API).
Data confidentiality and Auditability	Da der er tale om personfølsomme oplysninger skal de naturligvis beskyttes, men samtidig er der ikke tale om specielt økonomisk værdifulde oplysninger, og sikkerheden vil derfor ikke være mere kompliceret i skyen end på et data center. Et aspekt der er endnu vigtigere end sikkerhed er safety. Da der er tale om tele-medicin er det vitalt vigtigt at sammenkædningen af data er intakt, så en given patient får den korrekt medicin til tiden og i den rigtige dosis. Samtidig vil det være et krav at man vil kunne følge en given handling fra initiering til konsekvens. Disse aspekter vil dog ikke være anderledes på et data center end i skyen.
Data transfer bottlenecks	Der er p.t. ikke tale om kæmpe store data mængder til tele-medicin, men hvis f.eks. systemet skulle kunne supportere Cat-scan og CT-scan images er det en helt anden situation. Her vil det være vigtigt at overveje hvordan enormt store data-mængder kan uploades til skyen. Der er dog udbydere der tilbyder et hard-link til upload at data til skyen – i det tilfælde kan man simpelthen sende et fysisk drev med posten og så få det uploadet via et hard-link til skyen. Denne upload udføres af cloud udbyderen, og det er derfor vigtigt at sikre sig at det er muligt, hvis det er et behov.
Performance Unpredictability	Selv om der performance er vigtigt, så er der ikke tale om streaming af store data mængder, og den variation i performance der kan være alt efter contention på IO elle fysisk placering af web logic og business logic i forhold til database, vil være uden nævneværdig betydning i denne type applikation. Så længe cloud udbyderen kan garantere en acceptabel QoS så er variationen i performance uden betydning.
Scalable storage	Den form for skalering der er behov for her i form af database plads er ikke noget problem i skyen, og det vil være muligt at udvide med flere database servere hvis nødvendigt.
Bugs in large distributed systems	Det er klart at jo mere kompleks skyen er, jo større er risikoen for fejl. Dette er en reel bekymring, da fejl vil kunne påvirke ikke kun availability men også security og safety. Omvendt kan fejl også findes i data centre, selv om kompleksiteten er mindre, og der er efterhånden en del erfaring med VM og shared resouces, så hvis man vælger en konservativ udbyder med en sky der har eksisteret i noget tid, er risikoen til at overkomme. I dette tilfælde kan det også være en fordel med et redundant system (flere udbydere) evt. kombineret med et mindre private data center til at håndtere vital funktionalitet i tilfælde af nedbrud.
Scaling quickly	Der er ikke behov for denne form for skalering så det er ikke et

	problem.
Reputation fate sharing	Da telemedicin både er lægeligt, menneskeligt, teknisk og politisk så vil der vil helt sikkert være nogle der vil sikre sig at de ikke risikerer at blive hængt ud i pressen. Der er mange muligheder for skandaler: ”Kommunens data ligger side om side med børneporno”. ”Staten støtter colombianske narkobaroner” (f.eks. medejerskab i selskabet der udbyder skyen). ”Lægerne for hjælp af børnearbejdere” (server farm i Afrika med vedligeholdelse af børn). Man kan sagtens blive hængt ud selv om man umuligt kunne have vidst noget om det, så denne risiko vil altid findes, men den kan dog reduceres ved at vælge en større og velrenommeret udbyder, da disse også selv har en interesse i at holde et rent hus. Samtidig kan man indhente oplysninger omkring ejerskab og stille krav i kontrakten til arbejdsforhold og vilkår for ansatte i selskabet. Det er dog tvivlsomt at man ville komme langt med disse krav hvis vi taler Dansk telemedicin projekt op mod Amazon.
Software licensing	Der er efterhånden en del applikationer og OS’er der har fået en licens model der passer bedre til skyen, og hvis man kikket på at anvende et der ikke har en passende licens model, så skal man måske overveje om det er den rigtige applikation eller OS, da dem der ikke følger med strømmen og tilpasse sig på dette område med overvejende sandsynlighed vil ende som et ikke-supporteret applikatione eller OS. Tag f.eks. OS/2.

Baseret på ovenstående analyse kan det konkluderes at der ikke er nogle vægtige problemer med at anvende telemedicin i skyen, men omvendt er der heller ikke nogle meget store fordele. Det må derfor være meget op til hvilken økonomisk aftale man kan få på hhv. cloud og data center løsningen, samt hvilken tilpasning der er påkrævet (igen en økonomisk overvejelse, da tilpasning koster penge).

4. TM12 i PaaS

Da TM12 er skrevet i Java er det allerede designet til at eksekvere i et virtuelt miljø – Java VM’en. Det er naturligvis muligt at der findes en Cloud udbyder der giver mulighed for er PaaS hvor man simpelthen får stillet en Java VM til rådighed. Der findes microcontrollere der er designet specifikt til at eksekvere Java byte code (eller et udsnit af den), så hvorfor ikke en sky. Hvis vi derimod kikker på en anden leverandør kan der nemt være tale om et helt andet API, og applikationen vil derfor skulle redesignes til dette API. Jo mere moduleret, med logiske lag og høj lav coupling og høj cohesion, arkitekturen til TM12 er designet jo nemmere vil det være.

Google App Engine anvender Java, Python eller Go. For Python skriver google følgende:

- a complete Java 6 runtime environment in a secure sandbox environment
- based on common Java web technology standards, including servlets and WARs, JDO and JPA, java.net, JavaMail and JCache
- a plugin for the Eclipse IDE makes project creation, testing and deployment a snap

- supports other languages that compile to the JVM or use JVM-based interpreters, such as JRuby, JavaScript (Rhino), and Scala

Baseret på dette skulle det ikke være en større opgave at portere TM12 til Google App Engine – I hvert fald ikke in-memory versionen. Mongodb er ikke Java, og vil derfor ikke kunne eksekvere i Google App Engine. Det er muligt at eksekvere mongodb udenfor google app engine og bare tilgå den derfra. Hvis databasen også skal ind i under Google App Engine skal der anvendes App Engine's data repository i stedet for mongodb.

Den bedste måde at finde ud af det på er dog at prøve det. Først deployede vi et Hello World projekt til Google App Engine, og det virkede fint. Vi konstaterede at Google AppEngine også anvender Jetty, så det gør det nemmere.

API'et skal dog overholdes, og det kræver et entry point modul, ligesom at der skal anvendes App Engine's data repository – dette vil dog "bare" være en endnu et persisteringsmodul på linje med in-memory og mongodb. Endelig skal resources filerne lægges ind under web.xml, som er Googles konfigurationsfil.

For sjov prøvede vi at lægge source koden ind under vores Hello World projekt, og det kan compile og eksekvere uden problemer på localhost (uden jetty.jar og jetty-util.jar, da de er med i Google API'et), hvis man ændre entry point. Dette er dog ikke muligt i en deployet version, der skal entry point være korrekt.

Et forsigtigt estimat vil være at en der kender både TM12 og Google App Engine API vil kunne gøre det på et par timer til den første hul-igennem test, og derefter måske 3-4 timer til oprydning.

Part 2

Dette afsnit omhandler arkitekturel evaluering og er yderligere inddelt i nogle underafsnit.

ATAM

1. Discuss how ATAM may be tailored to a system such as the TM12 system.
2. Discuss which steps of ATAM have already been performed as part of your QAW in H2.
3. Draw a utility tree based upon your developed QAS from H2.
4. Pick one or two suitable QAS from TM12 and perform Architectural Approach Analysis (ATAM step 6), and describe risks, sensitivity and tradeoff points using an (adapted) form of the table shown in Bass et al., figure 11.2.

aSQA

Apply the aSQA to the following two designs of TM12:

1. The original design of TM12 as described in H1.
2. Your redesigned TM12 as proposed in H2.2.

Discussion of Evaluation Methods

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Discuss the evaluation methods both in relation to the TM12 evaluation and in relation to your own daily practice as a developer/architect.

ATAM

1. Tilpasning til TM12

ATAM procedure ville faktisk passe ganske godt til et system som TM12. TM12 er et system med en klar kundegruppe og dermed også forretningsmål (business drivers).

Det er dog vigtigt at se på hvordan organisationen bag projektet er opbygget, og hvilke stakeholders der er til rådighed.

Hvis vi tager udgangspunkt i virkeligheden så er TM12 et universitetsprojekt.

Ud fra dette er der to muligheder. Enten at den oprindelige arkitekt (underviser og/eller studerende) går ud og finder en køber som er villig til at købe produktet når det er udviklet og at investere i udviklingen (betale på forskud), eller at der søges om midler til selv at implementere projektet, for derefter at forsøge at sælge projektet.

I det første scenarie er der naturligvis en klar kunde (stakeholder), som også vil kunne hjælpe med scenarier og domænekendskab. I det andet scenarie vil det være nødvendigt at indhente den nødvendige domæne-viden udefra (læger, patienter, sygeplejere, …). Der er faktisk fordele og ulemper ved begge dele. Hvis der er én stor klar kunde kan det være problematisk at skulle afholde et åbent arkitektur review som der lægges op til i ATAM. Det er de færreste der synes godt om at skulle udstille evt. fejl og mangler overfor kunden. Og selv om kunden er repræsenteret via brugere (patienter, læger, mm.) så er det stadig folk der har adgang til kunden, og de vil derfor kunne fortælle om evt. fejl og mangler. Samtidig er det ofte tilfældet at når arkitekturen er skrevet så indeholder den enormt mange ressourcer i form af rationaler over forsøg og eksperimenter som viste sig ikke at være en god ide. Dette betyder at arkitekturen indeholder en hel del værdi, som en teknisk kyndig (og kynisk) kunde kunne spare ved ”bare” at implementere arkitekturen selv. Dette kan delvist undgås ved at anvende ”egne” domæne specialister – men her kan det være svært at adskille nice-to-have fra need-to-have, da en domæne-specialist (i modsætning til en kunde), ikke har pungen fremme, og det derfor ikke koster noget at prioritere et ønske foran et andet.

Vi vil dog se bort fra disse risici og kun fokusere på ønsket om at opnå en god arkitektur. I det tilfælde kikker vi på hvad der er nødvendigt. Første gang man høre om ATAM virker den som en relativt stor procedure, men efter vi har læst lidt op på den er vi kommet frem til at det ikke nødvendigvis behøver at være tilfældet.

ATAM ynder at have en del bogholderifunktioner (tidstager, schribe, …), og her vil vi mene at man med fordel kan skære ned på antallet. En enkelt referent burde kunne gøre det. At gøre brug af et professionelt ATAM konsulentbureau er nok også oftest at skyde over målet – så komplekst er det heller ikke, men da bogens forfatter er ATAM konsulent (og så meget tjener man heller ikke på at skrive software bøger), er det

selvfølgelig forståeligt nok. Vi vil derfor anbefale at man kan udpege en intern arkitekt fra et andet projekt til at varetage denne rolle.

I Part 1 mødes projekt leder og arkitekt m.m. og aftaler stakeholders og hvornår der kan aftales et møde og hvem der skal deltage (eksperter, domænekyndige, …) samt hvad der skal sendes ud i forvejen af arkitekturdokumentation. Dette er et nødvendigt og uundgåeligt trin, da der jo skal findes en mødedag. Det vil dog være muligt at skære lidt i mødedeltagelsen, men faktisk vil det være nødvendigt med mindst projektlederen og arkitekten.

Part 2 omhandler selve analysen. Her har ATAM valgt at splitte den op i 3 separate møder, og det kan nemt virke som rigtig meget, men efter at have tænkt lidt over det så er det faktisk ikke tilfældet.

Det første møde fokuserer på de rent tekniske aspekter (dog altid med en forretningsvinkel) hvor arkitekten og arkitekturspecialister (evaluation team) deltager sammen med projektleder og projektchef samt andre projekt ansvarlige. Dette giver god mening, da de rent tekniske aspekter er vigtige at få klarlagt, og disse aspekter er ikke interessante for kunde eller andre stakeholders. Der er måske mangler i dokumentationen, eller detaljerne omkring valg af architectual drivers skal diskuteres. Det der kommer ud af mødet er en mere klarlagt arkitektur.

Denne arkitektur renskrives og klarlægges før andet møde hvor stakeholders er med. De vil blive forlagt arkitekturen og resultatet fra første møde, og vil nu kunne komme med yderligere kommentarer og forslag.

Efter mødet bliver der samlet op på dokumentationen og der holdes et tredje møde hvor resultaterne fra andet møde præsenteres for samme team som var til mødet. Dette giver en god mulighed for at sikre at der er fuld enighed omkring resultatet og snakke det igennem en sidste gang.

Dette lyder måske som mange møder, men hvis man skære ind til benet og kun afholder 1 møde vil det betyde at stakeholders (og kunden) skal sidde med under det meget tekniske møde, og sandsynligvis blive både forvirrede og trætte inden man er kommet frem til den del de kan hjælpe med. Der er en stor risiko for at de ender med at sige: ”hvorfor er vi her?”. Samtidig hvis man dropper tredje møde og bare udsender resultatet som en rapport så er der 99,9% sandsynlighed for at rapporten aldrig bliver læst af stakeholders, og fejl og misforståelser vil derfor kunne forblive i rapporten, som arkitekten og projektlederen vil arbejde ud fra. Samtidig er der en risiko for at heller ikke arkitekten læser rapporten, da de jo var med til mødet, og så kan der også her være uoverensstemmelser mellem rapporten og det arkitekten arbejder ud fra. Af disse grunde er det en fordel at afholde de tre separate møder.

På selve møderne er der også en del trin i ATAM. Til møde 1 (i part 2) starter man med at præsentere ATAM så man ved hvad man skal i gang med. Hvis man i forvejen er bekendt med ATAM kan dette naturligvis droppes. Derefter præsenteres forretningsmål (business drivers). Dette mener vi er vigtigt, da der alt for ofte bliver fokuseret mere på design og arkitektur, men uden at tænke på om det er nødvendigt. Derefter præsenteres arkitekturen. Selv om de er sendt arkitekturdokumenter ud, er dette vigtigt så alle har samme forståelse af arkitekturen. Derefter genereres et utility tree, som giver et godt overblik over hvilke scenarier og hvilke kvalitets attributter der er og hvor vigtige de er (sorteret efter vigtighed og kompleksitet). Dette

giver god mening, og er en udmærket måde at analysere arkitekturen. Derefter analyseres den arkitekturelle fremgangsmåde (architecturel approach). Dette er en god måde at identificere patterns og til at vurdere om disse spiller godt sammen, om de modarbejder hinanden, og om de er korrekte i forhold til de business drivers der er vigtige og de quality attributes der er prioriteret. Også dette er et trin der er værd at tage med.

Til møde 2 starter man med at beskrive ATAM, hvilket giver god mening da det er sandsynligt at stakeholders ikke kender metoder. Derefter gennemgås resultatet af møde 1, hvilket betyder at alle får den renskrevne opdaterede arkitektur, quality attributes og scenarier. Dette er også vigtigt så der er samme udgangspunkt og forståelse. Derefter er der en brainstorm hvor stakeholders kan komme med yderligere scenarier, hvilket er yderst vigtigt da de har en helt anden indsigt i brug og domæne. Dernæst forklarer arkitekten hvordan arkitekturen kan klare disse scenarier, hvilket giver dem en mulighed for at kommentarer løsningen og arkitekten kan overveje om det er den bedste løsning.

Så samles dette til en rapport som præsenteres til et tredje møde, hvor stakeholders og arkitekt kan ”sign of” på arkitekturen, og alle har nu en fælles forståelse af arkitekturen, business drivers og prioritering.

Vores anbefaling er altså følgende:

1. Anvend en intern arkitekt fra et andet projekt i stedet for et konsulentbureau.
2. Følge ATAM som beskrevet ovenfor.

2. ATAM og QAW

Først og fremmest er det vigtigt at forstå at QAW og ATAM ikke er konkurrenter, men derimod komplementere hinanden. Dette skyldes at QAW og ATAM udføres på forskellige tidspunkter. QAW udføres før der er en arkitektur og anvendes til at genererer de kvalitetsattributter og forretningsmål (business drivers) der skal bruges til at drive udviklingen af arkitekturen. ATAM udføres lidt senere når det første udkast til en arkitektur er klar.

Når dette er sagt så er der dog rigtig mange ligheder mellem de to processer. Faktisk kan man med lidt god vilje sige at QAW er ATAM uden den arkitekturelle fremgangsmåde (architecturel approach), hvilket giver god mening for hvis der ikke foreligger en arkitektur er det svært at analysere den. ATAM lægger også op til en lidt mere formel struktur, hvilket også giver god mening, da der jo netop foreligger en arkitektur at tage udgangspunkt i.

Disse ligheder gør også at hvis der er udført en QAW vil det være betydeligt enklere at udføre en ATAM.

Den første del af ATAM skal stadig udføres. Der skal udvælges hvilke arkitekturdokumenter der skal sendes ud og hvem der skal deltage. Denne del vil dog være betydeligt kortere da man kender stakeholders fra QAW, så det er mest et spørgsmål om at vurdere om der skal inviteres flere eller andre stakeholders.

Da stakeholders har været med til QAW ved de godt hvad der venter og introduktioner kan derfor gøres korte. Nye stakeholders eller deltagere vil dog skulle informeres.

Efter en QAW foreligger der en masse scenarier, og disse kan man med fordel tage udgangspunkt i. Det betyder at mødet mere vil omhandler om der har været ændringer i de præmisser der lå til grund for QAW, eller dennes konklusion (hvilke business drivers er der og hvilke kvalitetsattributter og hvad er deres prioritering).

Den/de personer der skal analyserer arkitekturen har også et rigtig godt udgangspunkt da scenarier, kvalitetsattributter og business drivers er defineret. Det er naturligvis muligt at stakeholders har skiftet prioritering, men da scenarierne er skrevet vil det være betydeligt nemmere at sige: ”Nej, det er alligevel ikke vigtigt” eller ”Det må vi hellere prioritere op”.

En ATAM efter en QAW bliver altså et spørgsmål om at:

1. Gennemgå og tilrette de eksisterende scenarier, business drivers og quality attributes samt tilføje nye hvis nødvendigt.
2. Foretage analyse af architecturel approach

Dette vil sandsynligvis kunne gøres på én dag, og dermed reducere antallet af mødedage. Alt efter vurdering vil det være muligt at holde det tekniske møde og stakeholdermødet sammen. Dette er muligt da stakeholders allerede kender scenarierne, og derfor bedre vil kunne forstå det tekniske, som også er kortere af varighed. Det vil dog stadig være fordelagtigt at holde det som to separate møder (måske formiddag og eftermiddag).

3. Utility tree

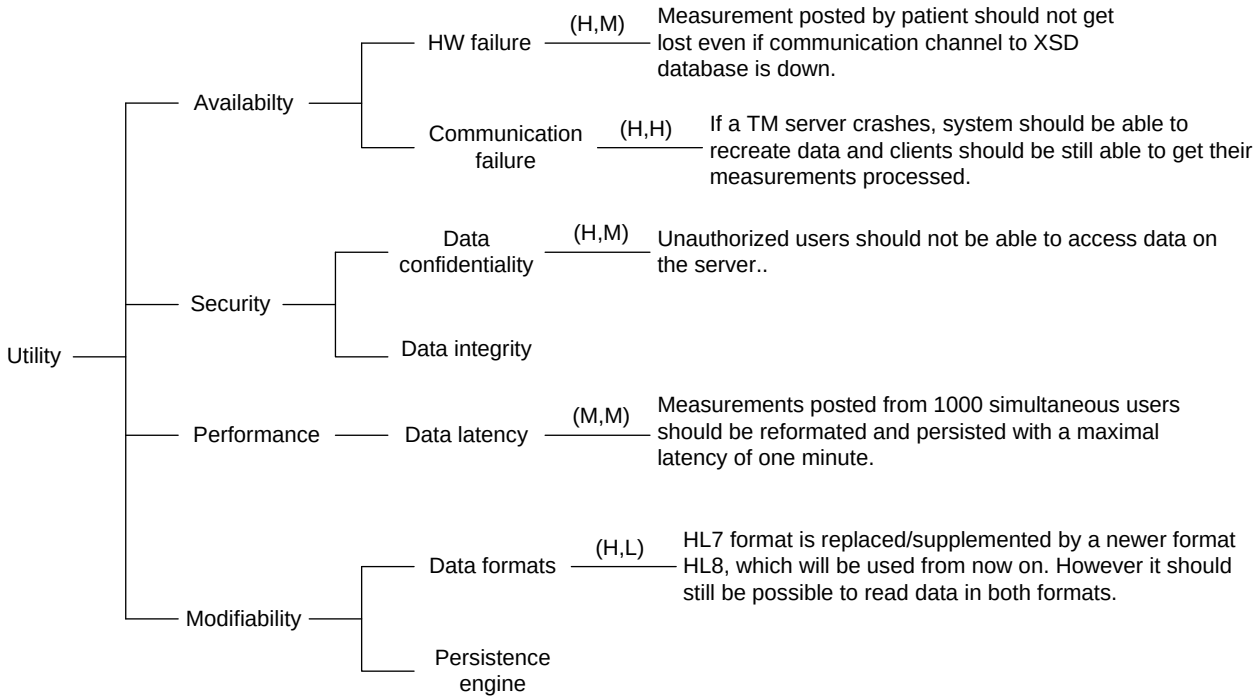
Et utility tree er en måde at danne sig et overblik over de scenarier der er samt deres indbyrdes kompleksitet.

I det nedenstående utility tree har vi taget og indsat de scenarier der er for opgave H2. Vi har valgt at undergruppere nogle af dem, selv om der med det givne antal nok ikke er behov for det. Vi har valgt den underinddeling der giver mening og hvor vi mener at der ville være andre scenarier hvis vi havde flere.

Under security har vi valgt at inkludere en tom gruppe, hvilket skyldes at data integritet er et vigtigt aspekt, men er ikke adresseret.

Det samme gælder for modifiability, hvor vi har inkluderet det underliggende persisteringslag, da det ofte er et område hvor man ønsker mulighed for at udskifte implementationen.

Prioriteringen stammer fra en vurdering af vigtigheden af de forskellige scenarier og quality attributes. Som det kan ses er de alle meget vigtige, hvilket giver god mening, da de scenarier der er tages med i H2 er de vigtigste af dem. Kompleksiteten stammer fra en vurdering af hvor komplekst det vil være at implementere den givne funktionalitet. Som det kan ses er der en enkelt H,L. Ifølge ATAM behøver vi ikke fokusere på denne, og man kan overveje om et andet scenarie skal inkluderes i stedet.



4. Architectual approach analysis

Analyse af den arktekturelle fremgangsmåde (architecturel approach) er et vigtigt skridt, som gør brug af både scenarier, arkitekturdiagrammer, patterns og en stor del erfaring. En arkitekt med erfaring indenfor det område som analysen omhandler (f.eks. databaseinteraktion, IPC, webservices, …) vil kunne bruge denne til at vurdere arkitekturens anvendelighed i forhold til de kvalitetsattributter og businness drivers der er fremsat. Arktitekten kan også trækker på andre arkitekters erfaring og kikke på architecurel patterns. Architectual patterns minder om design patterns, bare på et højere plan. Forskelle patterns har forskellige fordele og ulempler. Her er det også vigtigt at nævne anti-patterns, som en erfaren arkitekt også vil kende og derfor kunne undgå.

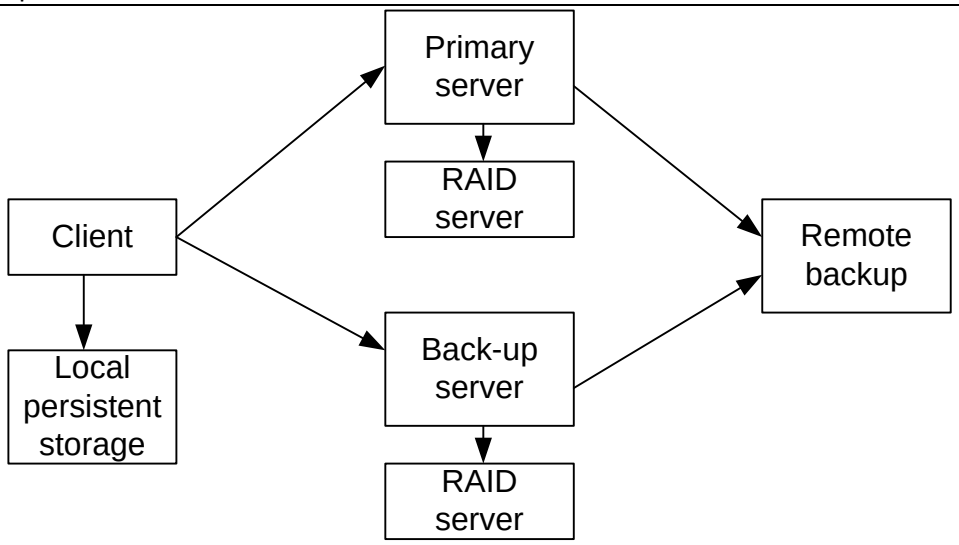
For at kunne foretage analysen kan man f.eks. anvende den template som Bass et al foreslår.

Vi har valgt at analysere følgende QAS fra H2:

#13 If a TM server crashes, system should be able to recreate data and clients should be still able to get their measurements processed.

Dette scenarie indeholder to uafhængige detaljer: 1. vi skal sikre at der er en server som klienten kan anvende og 2: vi skal sikre at det data der er sendt til serveren ikke går tabt. Vi har også valgt at beskrive analysen på engelsk da scenariet oprindeligt er skrevet på engelsk (og det er templaten også).

Scenario #: 13	Scenar*io: If a TM server crashes, system should be able to recreate data and clients should be still able to get their measurements processed.*
Attribute(s)	Availability

	(Performance)			
Environment	Normal operations			
Stimulus	Fatal server crash.			
Response	Client continues normal operations with a maximum extra delay of 30 seconds.			
Architectural decisions	Sensitivity	Tradeoff	Risk	Nonrisk
Backup server	S1		R1	
RAID implementation				N1
Remote backup	S3			N2
Client data retention	S4	T1		
Reasoning	Back-up server is continuously ready and client switch-over if no response within 20 seconds. Back-up server run same application and operating system as main server (risk-1) A RAID implementation will protect against hardware failure in the hard-drive. The remote backup will perform an incremental back-up of all data every 24 hours. All clients keep data in a local cache until server response allows deletion. This ensures that in the event of a destructive server crash the data can be retrieved from the clients. The clients are instructed to erase local data when data has been backed up.			
Architecture diagram				

S1	We could keep a routing server in front of the main server, which could ping the main server every n seconds and switch server if non-responsive, yet this may lower performance due to extra routing and will also increase complexity and price, and impose an extra load on the server from the heartbeat.
S3	By increasing the frequency of the remote back-up we can limit the required amount of client caching, yet at the cost of putting more load on the server (reducing performance).
R1	By using the same application on the same OS there is a risk of application failure also existing on the backup-failure. F.eks. a failure to accept leap-year would exist on both servers. Secondly an OS update crash (the operating system vendor may send out an update that makes the system fail) is a risk when same OS is used. The alternative is a parallel implementation on a different architecture (very expensive).
S4	By using a simple timer in the client data may be removed earlier.
T1	Keeping the latest client data on the client as a safety feature (backup) lowers security, as the client is now vulnerable to theft. This is however a limited risk and only the clients own data is at risk. Encryption may be employed to ensure confidentiality.

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

aSOA

1. Det oprindelige design

	App server (IPC pacakage)					Database (XDS pacakage)					Home Client				
	t	c	h	i	f	t	c	h	i	f	t	c	h	i	f
Avalibility	5	1	1	5	5	5	1	1	5	5	3	1	3	3	2
Performance	3	2	4	4	2	5	2	2	5	4	3	1	3	3	2
Modifiability	4	4	5	4	1	2	1	4	3	1	3	1	3	3	2
Testability	4	2	3	3	2	2	1	4	1	0	2	1	4	3	1
Security	5	1	1	5	5	3	1	3	3	2	5	1	1	5	5
Usability	2	1	4	3	1	2	2	5	1	0	5	1	1	5	5

Change Log: Startup of project.

2. Det nye design

	App server (IPC pacakage)					Database (XDS pacakage)					Home Client				
	t	c	h	i	f	t	c	h	i	f	t	c	h	i	f
Avalibility	5	1	1	5	5	5	1	1	5	5	3	1	3	3	2
Performance	3	2	4	4	2	5	2	2	5	4	3	1	3	3	2
Modifiability	4	4	5	4	1	2	1	4	3	1	3	1	3	3	2
Testability	4	1	2	3	2	2	1	4	1	0	2	1	4	3	1
Security	5	2	2	5	4	3	1	3	3	2	5	1	1	5	5
Usability	2	1	4	3	1	2	2	5	1	0	5	1	1	5	5

Change Log: Security pacakage -> User authentication added. Testability down -> No tests added for security

2. Forsat H3

	App server (IPC pacakage)					Database (XDS pacakage)					Home Client				
	t	c	h	i	f	t	c	h	i	f	t	c	h	i	f
Avalibility	5	2	2	5	4	5	1	1	5	5	3	1	3	3	2
Performance	3	2	4	4	2	5	2	2	5	4	3	1	3	3	2
Modifiability	4	4	5	4	1	2	1	4	3	1	3	1	3	3	2
Testability	4	2	3	3	2	2	1	4	1	0	2	1	4	3	1
Security	5	2	2	5	4	3	1	3	3	2	5	1	1	5	5
Usability	2	1	4	3	1	2	2	5	1	0	5	1	1	5	5

Change Log: Avalibility spike -> Possibilities to improve Avalibility explored.

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Evaluering af metoder

ATAM

Meget lig QAW, men hvor der tages udgangspunk i en arkitektur design. Struktureret og relativ verbos, men der er dog mulighed for at skræddersy den til behov. Anvender scenarier, quality attributes, business drivers and architectual approaches. Business drivers er gode til at sikre at kvalitetsattributerne (og dermed arkitekturen) er stilet efter de konkrete forretningsmæssige behov, og ikke bare hvad arkitekten synes er bedst (sjovest). Quality attributes er gode til at definere de non-functional dele af kravene til arkitekturen, og kombineret med scenarier gør det både overskueligt og forståelige for stakeholders.

QAW

Minder om ATAM, men anvendes til at definere arkitekturen. Tager også udgangspunkt i business drivers, quality attributes og scenarier. Lidt mere light-weight end ATAM, men har risiko for at der ikke blive fulgt ordentlig op på analysen.

aSQA

Meget light-weight i forhold til ATAM. Fokus er her at det skal være en process der er hurtig at udføre(3-5 timer) og simpel at kommunikere. Det er tiltængt at metoden gentages ofte(1 gang pr sprint).
I aSQA starter man med at definere hvilke QA'er man ønsker at bruge og hvilken defination de skal have. Derefter deles systemet op i bider som så bliver vurderet individuelt.
For hver del af systemet gøres følgende:
De valgte qa'er gives en værdi for: Det ønskede niveau(t), Nuværende niveau(c), Så kan Sundheds niavuet beregnes (Forskællen mellem ønsket niveau og nuværende niveau.)med formel: 5-max(0,(t-c)).
Derefter prioteres vigtigheden af QA'en (i) . Nu kan man så beregne en værdi som angiver hvor det største gab er i forhold til vigtighed (f) . Formlen er: ceil((6-h) * i/5).
Alle tildelte værdier er i skalaen 1-5 hvor 5 er best.
Metoden kommer ikke med konkrete eksempler på hvad som skal forbedres men peger mere på et område som skal forbedres eks: performance. Dette skal ses i forhold til ATAM som resultere i lister med anbefalinger (risks discovered og sensitivity points).

TM12

TM12 er et system med høje krav til availability og security (og i mindre grad performance). Arkitekturen er allerede relativt defineret, hvilket peger i retning af en ATAM analyse (specielt da der allerede er udført en QAW). aSQA har den "ulempe" at den fokuserer på komponenter, og da TM12 er baseret kraftigt på kvalitets attributter og scenarier er en komponent baseret analyse lidt et skridt i en anden retning. Der er dog ingen tvivl om at med de høje krav til availability og security vil det være yderst fordelagtigt at udføre en analyse af arkitekturen efter der er lavet et udkast, men inden den implementeres. Specielt security er utroligt svært at tilføje senere og skal designed med fra starten.

Praktisk erfaring

Da vi er to i gruppen fra forskellige firmaer har vi naturligvis også forskellige fremgangsmåder.

Software Arkitektur i Praksis (Modul 2)
H5. In the Cloud + Architectural Evaluation

Anders

Da jeg arbejder i et lille firma er det nemt at tage diskussionerne i plenum når det er nødvendigt. Samtidig er der meget kort til kunderne, og sælger, projektleder, arkitekt og udvikler har derfor alle et indgående kendskab til business drivers og domænet. De største arkitektoniske udfordringer er nok at da det er så nemt at snakke sammen har man ofte en tendens til ikke at skrive noget ned. Samtidig kan det nemt ske at der justeres i prioriteringen på en dag til dag basis efter hvilken kunde der råber højest den dag, og det kan derfor være svært altid at holde business drivers for øje. Dette håndterer vi med status møder hvor vi gennemgår de brandslukningsopgaver der er, samt de arkitektoniske og udviklingsmæssige opgaver der vil blive udskudt af denne grund. Til tider kan en brandslukningsopgave udskydes ved at kunde simpelthen må leve med det, eller at der kan laves en midlertidig løsning (men her er det meget vigtigt at sikre at den senere bliver rettet – dette er en risk som man skal være bevidst om). Et andet issue er at da der er tale om high dependability SW/HW er vi meget forsigtige med at ændre i arkitekturen, og meget ofte bypasser man hellere et problem frem for at løse det korrekt, hvilket resulterer i code degeneration, hvilket sandsynligvis vil betyde et komplet redesign på et tidspunkt.

Vi anvender derfor ikke struktureret arkitektur analyse metoder som ATAM, QAW eller aSQA. Vi lægger til gengæld meget arbejde i at sikre de rigtige business drivers og quality attributes (dette har jeg fra modul 1 da jeg tog det), og kommunikere meget med kunden for at gennemgå krav, forventninger og scenarier. Jeg må dog indrømme at der ikke er nogen egentlig arkitektur evaluering. Det bliver højest til at jeg sætter mig ned og gennemgår mine diagrammer når kunde-basen ændre sig eller vi skal supportere en ny funktionalitet. Min arkitektur er ikke noget jeg deler med kunden, med undtagelse af de diagrammer omkring security som er påkrævet af datatilsynet i forbindelse med at være Databehandler for kommunen.

Mogens

I Solar arbejder vi med programmer som bliver brugt i 6 forskellige lande. I landene har vi udpeget område ansvarlige som er primære kontagt til it i forbindelse med nyudvikling. Der er tæt kontagt mellem udviklere og område ansvarlige, hvilket bevirker at der er generel konsensus om QA's, dog uden at der er nedfældet noget på skrift. I afdelingen har vi defineret for alle programmer hvilke udviklere som er primære udvilkere og seckondære udviklere. Ved nyudvikling vil der normalt altid været en primær udvikler involveret. Ved nyudvikling oprettes RFC (Request for change) dokumenter som evalueres og godkendes først af foretningen i alle lande og dernæst af en arkitekt eller primære udvikler. Normalt sker udviklingen i cykluser af 6-8 ugers perioder, hvor der er test,UAT og release. Sideløbende med dette er der i perioder store projecter som køre et paralelt spor. Vores vigtigeste metode til at sikre at Qualitets attributterne bliver overholdt er, konstant integration og et udvikler team som har 7+ års erfaring. For de fleste programmer har primære udvikler været med fra starten.

Jeg tror vi ville kunne bruge ATAM til at trykprøve vores eksisterende programmer. Jeg kunne forstille mig at man en gang om året ville bruge ATAM til at vurdere de forskellige systemer med henblik på at få belyst problemer og risiko ved valgte design. Det kunne så genere RFC'er til forbedring af systemet.

aSQA tror jeg at vi ville kunne bruges som informations kilde til styregrupperne da det er information der er let at bryde ned for ikke programøre. Det vil kunne være med til at belyse hvis et system har nogle trends gående i negativ retning.