Hand in H4

Software Architecture in Practice

Architectural Evaluation

Architectural Prototyping

Department of Computer Science, University of Aarhus

Aabogade 34, 8200 Århus N, Denmark

Gruppe: ?

20074842, Lars Kringelbach, lars@kringelbach.com

20064684, Marjus Nielsen, Marjus.nielsen@gmail.com

20074877, Morten Herman Langkjær, morten.herman@gmail.com

20054680, Peter Madsen, pm@chora.dk

<<Dato: 20. april 2008>>

1 H4a - Architectural Evalutation 4
1.1 ATAM 4
1.1.1 Tilpasning af ATAM til TS07 4
1.1.2 ATAM trin allerede udført I H2 5
1.1.3 Architectural approaches and sensitivity and tradeoff points 5
1.2 CBAM 7
1.2.1 Udvælgelse af scenarier 7
1.2.2 Uddybelse af scenarier 7
1.2.3 Prioriterring 8
1.2.4 Tildel værdi 8
1.2.5 Udvikl strategier 10
1.2.6 Bestem værdi 10
1.2.7 Beregn afkast 11
1.2.8 Vælg strategi 11
1.2.9 Bekræft resultater 11
1.3 aSQA 13
1.3.1 The original design of HS-07 as described in H1 13
1.3.2 Your redesigned HS-07 as proposed in H2”: 16
1.4 Diskussion 17
2 H4b - Architectural Prototyping 18
2.1 Lars 19
2.1.1 Prototype description 19
2.1.2 Classification 19
2.1.3 Characteristics 19
2.2 Marjus 21
2.2.1 Klassifikation 21
2.2.2 Characteristics 21

2.3 Morten 23
2.3.1 Prototype description 23
2.3.2 Classification 24
2.3.3 Characteristics 24
2.4 Peter 26
2.4.1 Prototype description 26
2.4.2 Klassifikation 26
2.4.3 Caracteristics 27
2.5 Diskussion 28
3 References 29

1 H4a - Architectural Evalutation

1.1 ATAM

1.1.1 Tilpasning af ATAM til TS07

"Discuss how ATAM may be tailored to evaluate a system such as the HS-07 system, i.e., how would you use ATAM if you were to perform an evaluation of HS-07 given its business characteristics? Do not perform a full evaluation, but just describe the tailoring":

Architecture Tradeoff Analysis Method (ATAM) er velegnet til at evaluere store systemers arkitektur på en struktureret måde og resulterer typisk i at man får klarlagt risikoer, tradeoffs og sensitivity points. Derudover tvinges/hjælpes udviklingsvirksomheden til at dokumentere systemets arkitektur og at forholde sig til hvordan business drivers mappes til konkrete tekniske afgørelser.

[Bass et al., 2007] beskriver ATAM som en forholdsvis grundig metode, som involverer mange aktører. I den henseende er HS-07 systemet et forholdsvis lille system med få architectural drivers og med en arkitektur af begrænset kompleksitet. Derfor er det relevant at tilpasse ATAM til opgaven og trimme processen så overhead mindskes til et acceptabelt niveau.

Alle skridt i ATAM er nødvendige, også for små systemer, men det ville være tilstrækkelig med én person (højst to) i evaluation team i forhold til de 3-5 som foreslået i [Bass et al, 2007]. Evaluation team skal bestå af personer, som ikke har været involveret i gennemførelsen af det projekt, som bliver evalueret. Hvis udviklingsvirksomheden er meget lille kan det derfor være nødvendig at evaluation team består af eksterne resourcer som har kompetence i evaluering af systemer via ATAM. Project decicion makers vil bestå af projektlederen, softwarearkitekten (de to er evt. én og samme person) og produktets ejer. Og endelig kommer architecture stakeholders til at bestå af en udvikler og en sælger i forhold til de 12-15 som er en tommelfingerregel for antallet af architecture stakeholders i ATAM.

Udover at skære ned i antallet af involverede personer er det også relevant at trimme processen i tid, men det sker delvist automatisk i og med at omfanget af business drivers og arkitekturen som skal dokumenteres og præsenteres er begrænset. [Bass et al., 2007] beskriver f.eks. en præsentation af arkitekturen, som skal kunne præsenteres på ca. en time ved hjælp av ca. 20 slides. I HS-07's tilfælde kan dette gøres på 10-15 minutter og ved hjælp af 6-8 slides.

Punkt 5-8 bliver også helt naturligt mindre tidskrævende for et lille system end for et stort, da der er færre kvalitetsattributter og færre scenarier at forholde sig til.

1.1.2 ATAM trin allerede udført I H2

"Discuss which steps of ATAM have already been performed as part of H2":

En del af business drivers såsom de primære krav og en primær use-case var allerede defineret i H1 og var derfor baggrund for H2. I HS-07 blev architectural drivers klarlagt via kvalitetsattribut-scenarier. Disse drivers er alle del af det som skal præsenteres i punkt 2.

I H1 blev systemets arkitektur beskrevet ved hjælp af et modul view, et component-and-connector view og et allocation view. Disse blev delvist modificeret i H2 og disse beskrivelser kan direkte benyttes i præsentationen af arkitekturen.

Der blev også lavet en brain storm med henblik på at afdække alle relevante aktørers ønsker og krav til systemet, som så blev brugt til at formulere nogle quality attribute scenarios. Dette dækker en stor del af punkt 5 og 7 i ATAM, men det ville naturligvis være bedst at brain stormen og udvælgelsen af de vigtigste forslag blev lavet sammen med de virkelige aktører.

1.1.3 Architectural approaches and sensitivity and tradeoff points

"Describe architectural approaches and sensitivity and tradeoff points of HS-07. You do not need to describe all architectural approaches (and related sensitivity and tradeoff points), but a fair set of them":

Arkitekturen i HS-07 bygger på forskellige arkitektoniske principper og efter udvælgelse af tre vigtige scenarier ved at opsætte et quality attribute utility tree, er vi kommet frem til tre yderligere architectural approaches. Vi præsenterer først to approaches i HS-07.

1.1.3.1 Architectural approaches i HS-07

Build.xml filen, som bruges når HS-07 kompileres, indeholder oplysninger om adresserne på hhv. gateway, termometre og radiatorer, og hvor mange styks der er af de to sidst nævnte. Dette er en modifiability taktik, der gør det enkelt at f.eks. ændre i antallet af sensorer (rekompilering er dog nødvendig). Så konfigurering via konfigurationsfiler er et sensitivity point i forhold til sværhedsgraden af at modificere basale egenskaber i systemet.

Abstract common services taktikken bliver brugt i HS-07, f.eks. i GatewayService, ThermometerService og RadiatorService, som alle har en Service basisklasse som de nedarver fra. Her en taktik som reducerer antallet af linier kode krævet for at modificere systemet. Så denne taktik er et sensitivity point i forhold til modifiability.

1.1.3.2 De vigtigste scenarier

De forskellige aktører er blevet enige om at disse tre scenarier er de vigtigste for systemet:

1. En korrekt identificeret person ændrer på ønsket temperatur. Der laves et audit som logges på persistent medie.
2. Et termometer holder op med at svare. Fejlen logges og systemet overgår til begrænset tilstand.
3. Brugeren ønsker at temperaturen kan indstilles individuelt per rum.

1.1.3.3 Architectural approaches in redesigned HS-07

For at håndtere de tre scenarier har vi valgt tre forskellige approaches:

1. Audit trail. Et audit trail kræver at der logges hver gang en bruger-hændelse indtræffer. I et system med begrænsede hardwareresourcer kan dette være en krævende opgave, som kan gøre systemet langsommere, udover at et persistent medie skal være tilgængeligt. Det sidste vil sandsynligvis gøre systemet dyrere. Til gengæld gør det systemet mere sikkert (i kraft af at brugeren skal identificere sig) og man kan finde ud af hvilken person lavede ændringer i systemet.
2. Exception and removal from service. Disse er to taktikker til at forbedre availability. Desuden kan det gavne performance at gatewayen ikke skal stå og vente på termometre der ikke fungerer.
3. Implementerer ny algoritme.

Architectural approach	Sensitivity point	Tradeoff
Configuration files	Modifiability	
Abstract common services	Modifiability	
Audit trail	Security	Performance, Cost
Exception and removal from service	Avaliability, performance	
Ny algoritme		

1.2 CBAM

" Apply the CBAM method to the architectural approaches uncovered in the ATAM exercise. You are free to come up with additional scenarios":

1.2.1 Udvælgelse af scenarier

Vi arbejder videre med alle de tre scenarier der blev berørt i ATAM-øvelsen:

1. En korrekt identificeret person ændrer på ønsket temperatur. Der laves et audit som logges på persistent medie.

2. Et termometer holder op med at svare. Fejlen logges og systemet overgår til begrænset tilstand.

3. Brugeren ønsker at temperaturen kan indstilles individuelt per rum.

1.2.2 Uddybelse af scenarier

Her uddybes scenarierne med hensyn til respons og respons-mål:

1. Respons i dette scenarie er at der laves et audit trail på at temperaturen er blevet ændret af en korrekt identificeret bruger. Respons-målet er hvor lang tid der går inden dette audit trail er lavet så det kan ses af fx en administrator. Vi ønsker at dette skal være muligt inden for 10 sekunder efter hændelsen er sket. Der er ikke implementeret et audit trail i det nuværende system, så vi kan ikke leve op til det ønskede niveau på 10 sekunder.

2. Respons i dette scenarie er at systemet skifter til begrænset tilstand og dermed fortsætter med at håndterer de resterende sensorere. Respons-målet er hvor lang tid der går inden dette skift i system tilstand foretages efter at sensoren er holdt op med at fungere. I det nuværende system polles sensorer hvert sekund, men eventuelle fejl bliver ikke håndteret, så systemet vil ikke skifte til begrænset tilstand.

3. Respons i dette scenarie er at temperaturen i hvert rum er som brugeren har ønsket det. Mere præcist kan man sige, at vi ønsker at kunne styre temperaturen i et rum med en bestemt præcision. Respons-målet er at vi skal kunne holde temperaturen i et rum med en præcision på ca. 0,5 grader celsius.

I følgende tabel er respons-målene summeret med værste, nuværende, ønskede og bedste niveau angivet:

Response goals				
Scenario	**Worst**	**Current**	**Desired**	**Best**
1	Never	Never	10 sekunder	1 sekund
2	Never	Never	1 minut	20 sekunder
3	>4 || <0,1 grader celcius	<0,1 grader celcius	0,5 grader celcius	0,5 grader celcius

1.2.3 Prioriterring

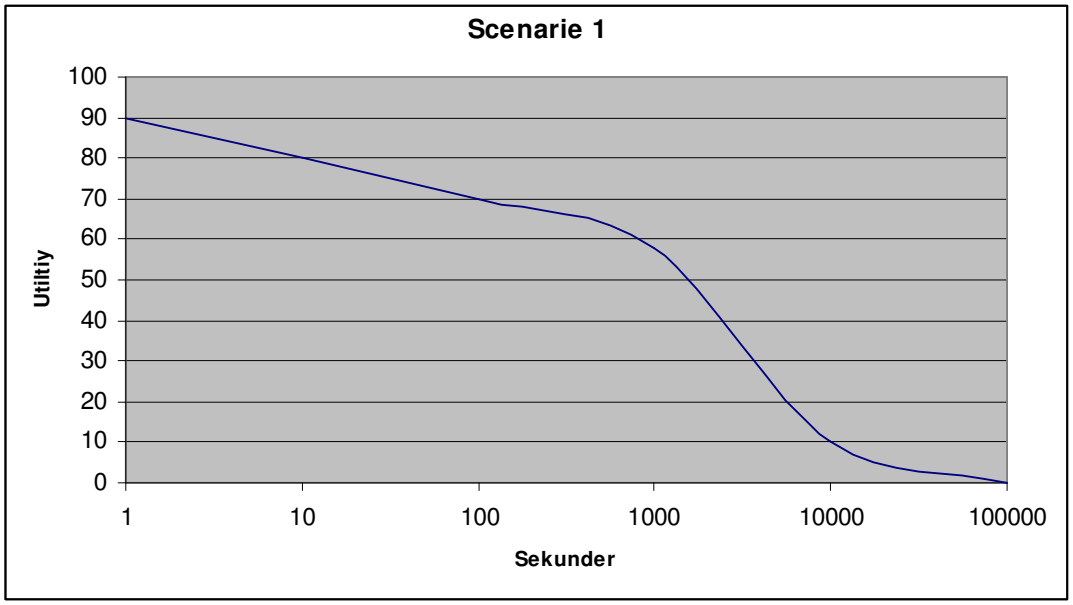
4 stakeholders har tildelt 100 stemmer hver til de 3 scenarier:

Response goals – Prioritization					
Scenario	**Votes**	**Worst**	**Current**	**Desired**	**Best**
1	75	Never	Never	10 sekunder	1 sekund
2	200	System fails	System fails	1 minut	20 sekunder
3	125	>4 || <0,1 grader celcius	<0,1 grader celcius	0.5 grader celcius	0,5 grader celcius

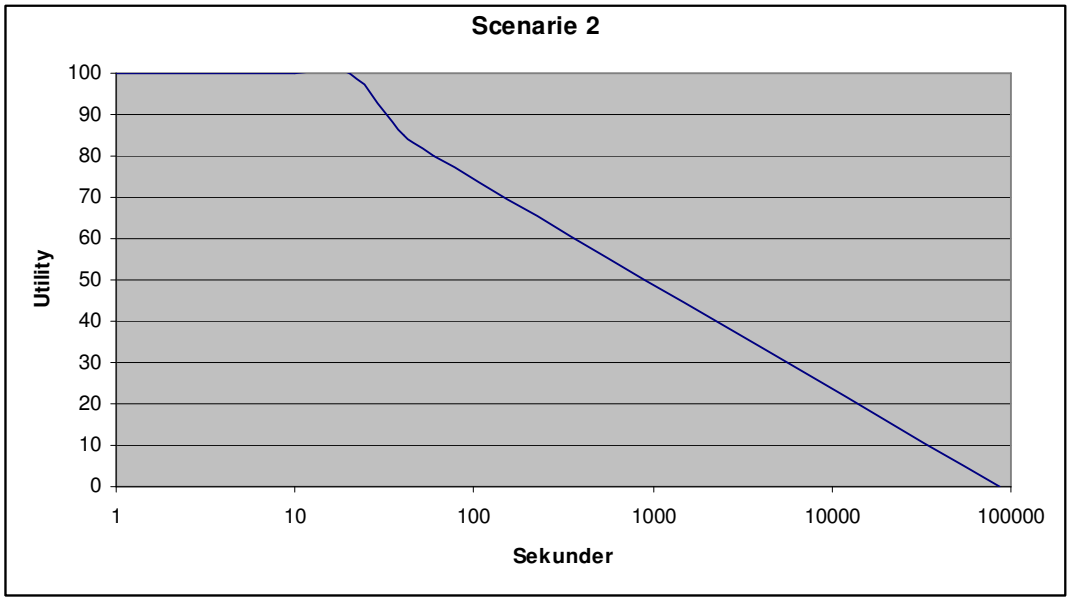
1.2.4 Tildel værdi

Følgende Utility grafer viser hvor meget værdi et respons har i forhold til hvor godt responsmålet opfyldes.

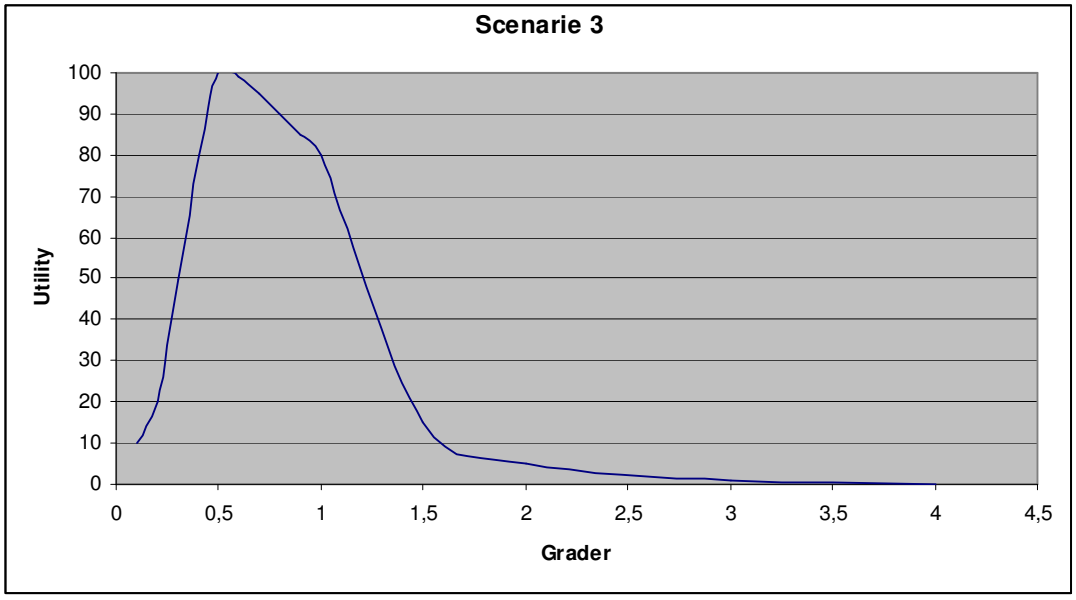
Grafen for scenarie 1 viser at værdien af et audit trail bliver mindre værd jo længere tid der går inden det kan ses i loggen. Hvis der går mere end 15 minutter falder værdien hurtigt og når der går mere end 3 timer har det stort set ingen værdi mere.



Grafen for scenarie 2 viser værdien af at der skiftes til begrænset tilstand inden for kort tid. Jo længere tid der går, jo mindre værdi har responset.



Grafen for scenarie 3 viser en speciel værdifordeling i forhold til hvor præcist systemet kan styre temperaturen mellem rum. Vi ønsker at systemet kan styre temperaturen med 0,5 graders præcision, og derfor er værdien højest der. Hvis vores system styrer temperaturen for præcist falder værdien hurtigt, da det vil overbelaste radiatorerne. Hvis vores system regulerer temperaturen mindre præcist falder værdien først markant, når unøjagtigheden bliver større end 1 grad.



Det giver følgende værdier i vores responsmål:

Response goals – Utility Scores					
Scenario	**Votes**	**Worst**	**Current**	**Desired**	**Best**
1	75	0	0	80	90
2	200	0	0	80	100
3	125	0	10	100	100

1.2.5 Udvikl strategier

Strategies				
Strategy	**Name**	**Scenarios affected**	**Current response**	**Expected response**
1	Audit trail	1	Never	<10 sekunder
2	Exception and Removal from service	2	Never	5 sekunder
3	Implement new algorithm	3	<0,1 grader celsius	~0,5 grader celsius

1.2.6 Bestem værdi

Udfra det forventede respons kan den forventede værdi findes på graferne i afsnit 1.2.4:

Strategies – Utilities				
Strategy	**Name**	**Scenarios affected**	**Current utility**	**Expected utility**
1	Audit trail	1	0	80
2	Exception and Removal from service	2	0	100
3	Implement new algorithm	3	10	100

1.2.7 Beregn afkast

Det beregnede afkast for hver strategi kan ses i følgende tabel:

Calculated benefit					
Strategy	**Scenarios affected**	**Scenario weight**	**Benefit**	**Normalized benefit**	**Total benefit**
1	1	75	80	600	600
2	2	200	100	20000	20000
3	3	125	90	11250	11250

1.2.8 Vælg strategi

Estimerede priser for implementation af strategier er brugt til at beregne Return-of-Investment:

Strategies – Utilities				
Strategy	**Strategy Cost**	**Total Benefit**	**Strategy ROI**	**Strategy Rank**
1	400	600	1,5	3
2	250	20000	80	2
3	100	11250	112,5	1

1.2.9 Bekræft resultater

"Consider alignment with business goals":

De oprindelige business goals for HS-07 systemet lyder som følger (fra H1):

"For our purposes there is one main use case for the HS07 system:
Control Temperature: The gateway collects measurements from thermometers and reports this to radiators that then control the temperature."

The major driving qualities attributes of the HS07 system are:

- *Performance.*
  HS07 should be performant so that a large number of thermometers and radiators may be part of the system.
- *Modifiability.*
  It must be possible to modify HS07 to include new types of sensors and actuators."

Hovedusecasen er mao. at samle målinger fra termometre og rapportere disse til radiatorer. Den strategi (3) som giver størst ROI er netop en som effektiviserer denne reguleringssløjfe. Denne er derfor fuldstændigt i tråd med *businnes goals*.

Strategi 2 som giver næsten lige så stor ROI sikrer *availablity* og i mindre grad *performance*. Availablity er ikke en af de definerede hoved kvalitetsatributter, men er under arbejdet med ATAM og CBAM blevet vurderet vigtigt af detagerne i processen. Dette er et udemærket eksempel på at teknikkerne giver taletid til folk som måske ellers ikke ville blive hørt. Det må i øvrigt anses for sandsynligt at *availability* er en vigtig kvalitetsatrribut for HS-07 (måske så vigtig at den er blevet taget for givet og derfor ikke blevet eksplicit nævnt).

Strategi 1 viser en meget lav ROI. Dette stemmer godt overens med det faktum at *security* ikke er defineret som en drivende kvalitetsattibut og dermed heller ikke har scoret ret højt når der stemmes om vigtighed.

Til sidst skal det bemærkes at der til denne øvelse kun er udvalgt tre scenarier, hvor en tilbundsgående ATAM og CBAM undersøgelse ville give væsentligt flere cenarier og dermed må forventes at være mere dækkende for fx. *modifiability.* Selv med det beskedne antal scenarier synes metoden at give en god indikation på hvor det kan betale sig at sætte ind.

1.3 aSQA

"Apply the aSQA to the following two designs of HS-07":

1.3.1 The original design of HS-07 as described in H1

For at benytte aSQA på HS-07, skal følgende skridt udføres:

Skridt	Forventet aktivitet	Kommentar og udført aktivitet
1.	Beskriv komponenter i systemet	Komponenterne er beskrevet i opgaven, og dette udelades her
2.	Definer metrikker	Er udført
3.	Definer niveauer	Er udført
4.	Definer niveauer for slut mål	Er udført
5.	Definer vigtighed	Er udført
6.	Mål nuværende tilstand	Er udført
7.	Fastslå systemets "health"	Er automatisk beregnet via regnearksskabelon til aSQA
8.	Fastslå fokusområde	Er automatisk beregnet via regnearksskabelon til aSQA

Tabel 1.3-1 aSQA fremgangsmåde

1.3.1.1 Beskriv komponenter i systemet

Da komponenterne og deres sammenhænge er beskrevet i opgaven samt besvarelsen af H1 er her kun noteret en liste over komponenter der evalueres.

aSQA Component Names		
1	Gateway	
2	Actuator	
3	Sensor	

1.3.1.2 Definer metrikker

Metrikkerne brugt til at evaluere arkitekturen vha. aSQA, er defineret ud fra et scenariespecifikt udgangspunkt. Da denne opgave baserer sig på genbrug af scenarierne fra H2, og kun et subset af scenarierne i denne opgave er dokumenteret efter metoden med source, stimulus, artifact, environment, response og response meassure, er alle scenarier taget med i deres oprindelige form efter brainstorm fasen af evalueringen. Skulle man udføre aSQA på et system burde man have dokumenteret alle sine scenarier på en måde så de beskrev noget målbart, og dermed kunne entydigt besvares om de er opfyldt eller ej.

Scenarierne som er taget med i evalueringen er følgende:

Performance

- The users want the system to be able to raise the temperature at least 5 degrees of Celsius per 10 minutes.

Availability

- The users wish that the remaining system continues to work when one or more actuators or sensors fail.
- The users want the system to indicate failing or deviating sensors and actuators.
- The developers want the system interfaces to be stable even when adding new soft- and hardware components
- The developers want the system to work correctly in situation where new actuators are added
- The users wish that the system has a maximum downtime of no more than 1 minute per day.

Modifiability

- The users want the system to be able to automatically update in case of new system releases.
- The developers want the system to be able to scale to having a high amount of actuators and sensors.
- The sales department wants the system to be expandable with new features.
- The developers want the system to be able to release software for new actuators and sensors (incl. logic) without doing a release of the base framework
- The sales department wants the system to have a low time to market.
- The sales department wants the system to allow third party integrators to expand it with new types of hardware sensors and actuators (along with the application logic needed).

Testability

None

Usability

- The sales department wants the system to be easy to deploy and maintain at the custumor site.

Security

- The users wish to use the system from a remote location under secure conditions.
- The users insist that no one but they can collect information about his or her home through hs07

1.3.1.3 Definer niveauer

Det følgende afsnit tager udgangspunkt I at de I H2 identificerede scenarier er de same som benyttes til at evaluere den oprindelige udgave af HS-07. Scenarierne er tage fra H2 af Gruppe Bravo. Software kvalitetsattributscenarierne som tages i betragtning i den følgende aSQA er :

De I det følgende brugte metrikker er defineret ud fra følgende skala:

Level 1: Unacceptable

• Important stakeholders find system unacceptable because of quality level of the attribute in question

– Level 3: Acceptable

• No relevant stakeholder find the system unacceptable because of quality level of the attribute in question

– Level 5: Excellent

• All relevant stakeholder are highly satisfied by the quality level of the attribute in question

1.3.1.4 Definer niveauer for slut mål

aSQA Target Levels			
	Gateway	Actuator	Sensor
Performance	3	4	4
Availability	5	5	5
Modifiability	4	3	3
Testability	3	3	3
Security	4	3	3
Usability	4	2	2

1.3.1.5 Definer vigtighed

aSQA Importance Levels			
	Gateway	Actuator	Sensor
Performance	3	3	3
Availability	5	5	5
Modifiability	5	2	2
Testability	1	1	1
Security	3	3	3
Usability	2	2	2

1.3.1.6 Mål nuværende tilstand

aSQA Current Levels			
	Gateway	Actuator	Sensor
Performance	2	4	4
Availability	2	2	2
Modifiability	2	2	2
Testability	3	4	4
Security	1	1	1
Usability	2	2	2

1.3.1.7 Fastslå systemets "health"

aSQA Health Levels			
	Gateway	Actuator	Sensor
Performance	4	5	5
Availability	2	2	2
Modifiability	3	4	4
Testability	5	5	5
Security	2	3	3
Usability	3	5	5

1.3.1.8 Fastslå fokusområde

aSQA Focus Levels			
	Gateway	Actuator	Sensor
Performance	1	1	1
Availability	4	4	4
Modifiability	3	1	1
Testability	0	0	0
Security	2	2	2
Usability	1	0	0

1.3.2 Your redesigned HS-07 as proposed in H2":

Under refaktoreringen af H2, fokuseredes I den oprindelige opgave primært på modifiability og availability.

Hvis man følger aSQA metoden, ville det efterfølgende stykke arbejde være at udføre skridt 5-8 i Tabel 1.3-1 aSQA fremgangsmåde. Da vigtigheden er den samme, grundet at opgaven er en evaluering af tilstanden efter refaktoreringen ud fra de samme scenarier, er vigtigheden den samme som i den oprindelige evaluering, og dette skridt udelades derfor.

1.3.2.1 Mål nuværende tilstand

aSQA Current Levels			
	Gateway	Actuator	Sensor
Performance	2	4	4
Availability	4	3	3
Modifiability	3	3	3
Testability	3	4	4
Security	1	1	1
Usability	2	2	2

1.3.2.2 Fastslå systemets "health"

aSQA Health Levels			
	Gateway	Actuator	Sensor
Performance	4	5	5
Availability	4	3	3
Modifiability	4	5	5
Testability	5	5	5
Security	2	3	3
Usability	3	5	5

1.3.2.3 Fastslå fokusområde

aSQA Focus Levels			
	Gateway	Actuator	Sensor
Performance	1	1	1
Availability	2	3	3
Modifiability	2	0	0
Testability	0	0	0
Security	2	2	2
Usability	1	0	0

1.4 Diskussion

"Discuss the evaluation methods both in relation to the HS-07 evaluation and in relation to your own daily practice as a developer/architect."

Det første indtryk i gruppen var at specielt ATAM i dens oprindelige form er gearet til langt større systemer og organisationer end det er tilfældet med HS-07. Selv om flere af gruppens medlemmer arbejder i større organisationer var det også en umiddelbar kommentar at metoden var alt for omfattende til brug på de systemer som gruppemedlemmerne udvikler. Da aSQA metoden arbejder på komponentniveau, var den forholdsvis enkel at benytte på HS-07. En af fordelene og samtidigt ulemperne ved aSQA er at man selv skal opfinde og definere sine metrikker og måleenheder, hvilket kan skabe risiko for at man måler forkert og ikke måler noget som er fuldstændigt kvantificerbart. Samtidigt giver det mulighed for at kunne tilpasse målingerne helt præcist til det formål og system som er impliceret i evalueringen. Hvis man har kvalitetsattribut-scenarier og strategier tilgængelige er CBAM relativt nem at gå til. Udarbejdelsen af blandt andet værdi-grafen, samt antagelser om forventet respons og pris for strategierne virker dog som en store usikkerheder i metoden. Disse usikkerheder vil dog nok være minimeret, hvis det er relevante stakeholders og arkitekten der udfører arbejdet, og følger op på resultatet som beskrevet i pkt. 9 af CBAM.

Efter at have arbejdet med at tilpasse ATAM til organistationernes størrelse har det vist sig at metoderne kan have relevans for selv relativt små organisationer, i det de tjener nogle vigtige formål:

1. Arkitekten tvinges til at kommunikere designet til stakeholders, hvilket bla. sikrer at en fornuftig dokumentation findes (eller som minimum at det bliver en identificeret risiko at det ikke gør)

2. De enkelte stakeholders får en formel metode til at kommunikere deres bekymringer. I gruppens erfaring bliver andre stakeholders end arkitekten sjældent hørt i mindre organisationer.

3. Output fra de forskellige metoder giver et godt overblik over risici og problematikker som skal angribes. Dette på en simpel form (tabeller og grafer) som gør output let forståeligt i store dele af organisationen.

Ingen i gruppen har brugt teknikkerne i deres daglige arbejde, men visse af de beskrevne elementer benyttes på uformel vis. Der er i gruppen ingen tvivl om at det daglige udviklingsarbejde kan forbedres ved at indføre flere af teknikkerne.

2 H4b - Architectural Prototyping

" Each member of the group should choose (at least) one example of architectural prototyping from his/her place of work. For each example describe and discuss the following":

Description: Describe the instance of architectural prototyping. What was the concern, who was the stakeholders, how was the issue resolved etc.?

Classification: Classify the architectural prototype; what type is it and why?

Characteristics: For the case, for each of the following characteristics, declare whether you "fully agree", "agree", "do not know", "disagree", or "fully disagree" with the following statements and explain why:

- "Architectural prototypes are constructed for exploration and learning of the architectural design space"
- "Architectural prototyping addresses issues regarding architectural quality attributes in the target system"
- "Architectural prototypes do not provide functionality per se"
- "Architectural prototypes typically address architectural risks"
- "Architectural prototypes address the problem of knowledge transfer and architectural conformance"

Are there additional characteristics of architectural prototypes that better characterize the case?

2.1 Lars

2.1.1 Prototype description

I forbindelse med udvikling af en ny softwareplatform til vores produkter ønskede vi at vurdere muligheden for at lave realtidsprocesser i stedet for at køre alt som en kerneproces. Hidtil havde vores tråde kommunikeret via et publish/subscribe baseret besked-system, der fungerede ved hjælp af delt hukommelse og message queues.

For at udnytte muligheden for at adskille vores tråde i realtidsprocesser der kan sikre at vores moduler ikke overskriver hinandens hukommelse skulle vores beskedsystem ændres til at kunne fungere uden brug af delt hukommelse.

Et client-server-baseret beskedsystem blev brugt til at lave en prototype der kunne vise om der kunne opnås god nok performance dels med en client-server baseret løsning og dels når der skulle kommunikeres mellem processer i separat memory-område. Den client-server-baserede løsning blev valgt da den på en simpel måde løser problemet med at vi ikke kan benytte delt hukommelse, frem for en løsning med et distribueret register over subscribers. Samtidig gav den nem mulighed for at koble sig på systemet udefra fx i forbindelse med test.

Prototypen blev lavet meget simpelt, og bestod i et antal processer der kommunikerede intensivt via det valgte beskedsystem. Resultatet viste at det ikke var muligt at opnå tilstrækkelig performance med den valgte løsning, og der blev i stedet for arbejdet videre med en løsning hvor realtidsprocesserne kommunikerer direkte med hinanden via message queues og et distribueret register over subscribers.

2.1.2 Classification

Ovenstående prototype er en eksperimentel prototype der blev brugt til at evaluere en arkitektur for om den kunne levere op til vores performance-krav.

2.1.3 Characteristics

“Architectural prototypes are constructed for exploration and learning of the architectural design space”

Delvist enig. I ovenstående tilfælde blev kun en enkelt arkitektur undersøgt. Efterfølgende blev der dog lavet simple forsøg for at afdække om direkte kommunikation mellem processer ville kunne opfylde vores performance-krav. Der blev ikke eksperimenteret med et distribueret register over subscribers.

“Architectural prototyping addresses issues regarding architectural quality attributes in the target system”

Fuldstændig enig. Prototypen blev netop brugt til at vurdere om den testede arkitektur kunne give os tilstrækkelig performance.

“Architectural prototypes do not provide functionality per se”

Fuldstændig enig. Der var ingen funktionalitet i vores prototype. Der blev sendt data uden nogen form for information, for at generere den load på systemet der skulle testes op i mod.

“Architectural prototypes typically address architectural risks”

Fuldstændig enig. Det var en stor risiko for os at skifte til både inter-proces kommunikation samtidig med at gå over til en client-server-baseret løsning. Eftersom udfaldet af undersøgelsen var negativ var det helt sikkert værd at bruge ressourcerne på en prototype.

“Architectural prototypes address the problem of knowledge transfer and architectural conformance”

Uenig. Der blev eksperimenteret med arkitekturen og et konkret beskedsystem. Efter undersøgelsen blev det hele kasseret (undtagen resultaterne), da det ikke opfyldte vores behov.

2.2 Marjus

Der skulle laves et system til modtagelse af dokumenter hvor madvareindustrien rapporterer ind for at dokumentere madvarernes oprindelse og kvalitet.

Hoved krav til systemet var at dokumenterne skulle være sikre og at uvedkommende ikke fik lov til at se dem. Desuden er der mange lovkrav, og andre udenfrakommende faktorer, som gør at krav til systemet er i konstant udvikling. Derfor var en høj grad af modifiserbarhed også et ønske uden at dette krav blev yderligere konkretiseret. Branchen havde i forvejen en række proprietære systemer, som i størst mulig grad skulle kunne interagere med det nye system.

Projektlederen havde ikke prøvet at lave lignende systemer før, og kunden var selv i tvivl om betydningen og omfanget af sine egne krav. Derfor blev det valgt at lave en vertikal prototype, som indeholdt forsimplet kernefunktionalitet. Kunderne kunne så verificere med deres it-afdelinger om deres proprietære systemer kunne bringes til at snakke sammen med det nye system.

Prototypen blev lavet i en lagdelt arkitektur, hvor der var et persistenslag, et lag til business logik og endelig blev alle metoder udstillet i et webservice lag. Efter at dele af den grundlæggende funktionalitet var implementeret, lavede kunderne udvidelser til deres systemer så de kunne kommunikere med kernesystemets webservices. Her blev det hurtigt klart at de fleste systemer naturligvis godt var i stand til at kommunikere via de udstillede webservices, men at det ikke var hensigtsmæssigt at udstille alle metoder som services. Performance blev alt for dårlig. Så vi lærte at det er nødvendig at lave servicerne mere coarse-grained. Modifiserbarheden er ikke let at teste i en prototype, men prototypen viste dog at selv om vi havde valgt en arkitektur som gavner modifiserbarhed, gav det ikke performanceproblemer.

2.2.1 Klassifikation

Denne prototype er exploratory. Der var ikke tilknyttet nogen egentlig arkitekt til udviklingen, så udviklingen byggede på projektlederens tidligere erfaring. Men da systemet på grundlæggende punkter afveg fra hvad projektlederen var vant til og fordi køberen af systemet var usikker på flere af signe egne krav, blev der valgt at lave et proof of concept. Store dele af den oprindelige prototype blev godkendt og derudfra blev det besluttet at fortsætte i samme retning.

2.2.2 Characteristics

“Architectural prototypes are constructed for exploration and learning of the architectural design space”

Enig. Prototypen blev lavet for at lære projektdeltagerne, ikke mindst kunden, om muligheder og begrænsninger ved arkitekturen.

“Architectural prototyping addresses issues regarding architectural quality attributes in the target system”

Enig. Den valgte prototype addreserede direkte flere af de vigtigste kvalitets attributter for systemet.

“Architectural prototypes do not provide functionality per se”

Uenig. Vores prototype indeholdt funktionalitet til at modtage dokumenter, og tillod brugeren at hente dokumenterne igen, søge i dem osv. Dette var helt enkelt nødvendigt for at demonstrere funktionaliteten før kunden tog den endelige afgørelse om at vælge at implementere systemet. Dette var også delvist en forudsætning for at

kundernes it-afdeliger kunne teste hvorvidt det var muligt at bruge kernefunktionalitet fra det nye system i deres egne systemer.

“Architectural prototypes typically address architectural risks”

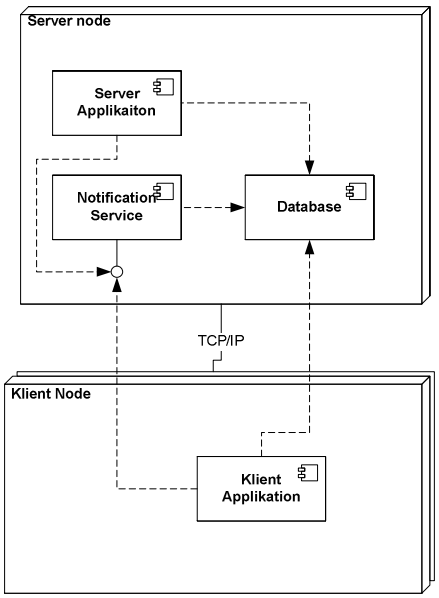
Delvist enig. Hovedformålet er nok typisk at afdække arkitektoniske risikoer, men ligesom med alle andre prototyper kan arkitektoniske prototyper også afdække alle mulige andre typer risikoer.

“Architectural prototypes address the problem of knowledge transfer and architectural conformance”
Enig. Prototypen blev brugt til at sætte kunden ind i den foreslåede løsning og at tillade kundens it-afdeling at afprøve om deres applikationer kunne bringes til at arbejde sammen med kernesystemet.

2.3 Morten

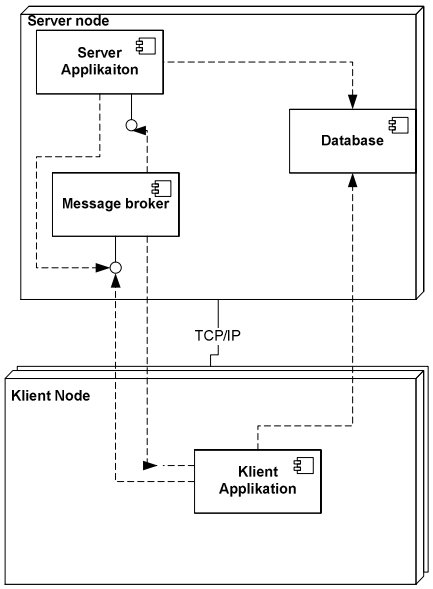
2.3.1 Prototype description

I forbindelse med et projekt hvor der var et antal systemer, som indsatte data i den samme database var der udviklet en infrastruktur, som kan ses af allocation viewpoint jvf. Figur 2-1.



Figur 2-1 Original arkitektur

Infrastrukturen var baseret på databasetriggers, som kopierer indsat data fra alle tabeller over i en notifikationstabel, i et propraitært tvær-applikationsspecifikt format. En notifications service poller med jævne mellemrum notifikationstabellen og distribuerer nye data til både klient applikationer samt server applikationer. Arkitekturen var valgt for at sikre understøttelse af flere DBMS'er på en genrisk måde, samt at kunne understøtte udvidelse af systemet med yderligere applikationer som skulle integreres på en enkel vis. Under produktets levetid ændredes kravene fra en server apllikation og en klient applikation til at der kunne være 25 klient applikationer. Da data indsættes i transaktioner, bliver notifikationstabellen en flaskehals, hvor det at triggers skal fyres giver et stort performance overhead samt at der opstod deadlocks når flere applikationer skrev data samtidigt. Alt ialt resulterede det i at produktet ikke kunne overholde kravene. For at imødekomme de nye krav lavedes en prototype hvor apllikationerne sendte data i samme format som tidligere, men til en microsoft message queue, og på serveren var der så en applikation som kunne distribuere data til andre apllikationer, ved hjælp af et distribueret observer pattern. Se Allocation viewpoint for ny arkitektur,



Figur 2-2 Ny arkitektur

Ved at fjerne databaseoverhead, kunne det igennem test sandsynliggøres at systemet ville kunne overholde 25 klients kravet, og samtidigt bibeholdes den garanterede leverance af data igennem brugen af microsoft message queue transaktioner. Herefter blev der lavet et fuldt design af Message broker komponenten, samt de interfaces der skulle benyttes til at abonnere på events fra message brokeren. Yderligere udvikledes et ny xml baseret dataformat til udveksling af data imellem applikationer.

2.3.2 Classification

Den arkitektoniske prototype er en eksperimental prototype. Den blev udviklet I forbindelse med tidligere analyser, samt en anbefaling fra en integrator, som I forvejen brugte Microsoft message queueing til deres infrastruktur. Prototypen havde til formål at benytte en kvanititativ tilgang til kvalificering af arkitekturen, hvor målinger dannede grundlag for at identificere hvor vidt prototypen kunne sandsynliggøre at den nye arkitektur kunne løse de tidligere identificerede problemer.

2.3.3 Characteristics

“Architectural prototypes are constructed for exploration and learning of the architectural design space”

Den arkitektoniske prototype er I fuldstændig overensstemmelse med ovenstående udsagn. Årsagen er at protoypen havde til formål at udforske muligheden for øget performance, og samtidig afdække muligheden for at benytte microsoft message queueing som teknologi til at udveksle information pålideligt.

“Architectural prototyping addresses issues regarding architectural quality attributes in the target system”

Den arkitektoniske prototype er I fuldstændig overensstemmelse med ovenstående udsagn. Årsagen er at protoypen havde til formål at udforske muligheden for øget performance igennem en ny arkitektur.

“Architectural prototypes do not provide functionality per se”

Udsagnet er som udgangspunkt I overenstemmelse med den udviklede prototype. Prototypen sandsynliggjorde at teknologi og arkitektur kunne facilitere en ønsket ændring, men prototypen kunne ikke benyttes som udviklet, da den funktionalitetsmæssigt ikke var fuldstændigt dækkende for produktet, samt ikke var udviklet i en kodekvalitet som levede op til målene for det påvirkede produkt.

“Architectural prototypes typically address architectural risks”

I forbindelse med den udviklede prototype har formålet ud over øget performance været at kunne sikre at der ikke ville blive spildt en stor mængde tid på at udvikle den forkerte arkitektur og infrastruktur. Efterfølgende er der forbrugt mere end 1000 timer på at udvikle infrastrukturen, integrere flere produkter, samt udvikle og dokumentere diverse API’er som integratorer skal udnytte til at integrere tredjepartsprodukter i løsningen.

“Architectural prototypes address the problem of knowledge transfer and architectural conformance”

Den arkitektoniske prototype addresserede I høj grad emnet omkring videndeling, og videnudvikling indenfor bestemte teknologier. Yderligere addresseredes forskellige problemstillinger undervejs, som gav input til hvordan der kunne blive en højere grad af overensstemmelse imellem de involverede produkter og den nye infrastruktur.

2.4 Peter

2.4.1 Prototype description

I forbindelse med udvikling af software til en ubemandet kiosk (info stander) løsning, var der i udviklings-organisationen opstået en idé om at bygge et persistenslag oven på systemets database. Ideen var opstået som et middel til at sikre at domænemodellen altid var afspejlet i databasen (konsistens var vigtig da systemet bla. håndterede køb og salg). Det var desuden defineret at modificérbarhed var vigtig, selv om udviklingsorganistation kun rådede over få personer med databaseviden.

Rent praktisk var konceptet at når et domæneobjekt blev instantieret, ville nye objekter replicere sig selv til databasen og eksisterende selv hente "sine" data fra databasen og lave konsistenscheck (sikre at to objekter med samme id ikke fandtes mv.).

Denne form for abstraktion oven på en database var ikke udført før i organistationen, hvorfor der var bekymring om det lod sig gøre (buildability) og om det ville performe tilstrækkeligt godt, i det en databasetransaktion umiddelbart kun kunne bestå af et objekts data.

Arkitekturen var således valgt ud fra en formodning at den ville give den ønskede konsistens og modificerbarhed, men med bekymringer omkring buildability og performance.

Det blev besluttet at lave en arkitekturprototype med de hovedformål at vise at det var muligt at lave et persistenslag og til vurdering af performance.

Prototypen blev bygget relativt hurtigt, hvilket viste at idéen var bygbar. Prototypen viste dog også at bekymringerne omkring performance var reelle, i det visse sekvenser af gentagen oprettelse af nye objekter ikke overholdte de stillede performance krav. Prototypen blev derfor brugt til performancemålinger, og problemet blev isoleret til den allerede identificerede bekymring om mange små databasetransaktioner.

Arkitekturen blev modificeret således det blev muligt at lave et sæt af domæneobjekter (som persistenslaget opretter i databasen som én transaktion) og der blev instrueret i at flere ens objekter skulle oprettes på denne måde. Med ændringen gav arkitekturen tilfredsstillende performance. Hvis ikke denne ændring var indført på dette tidlige tidspunkt, ville det helt sikkert have været dyrt at rette op på senere.

Formodningen om modificerbarheden er senere blevet bekræftet, i det udviklere uden databaseviden uden videre er i stand til at udvikle til produktet.

2.4.2 Klassifikation

Den beskrevne prototype er en eksperimentel prototype, i det den blev brugt til at undersøge egnetheden af en bestemt arkitektur, herunder at undersøge en bestemt arkitekturkvalitet, performance.

2.4.3 Caracteristics

“Architectural prototypes are constructed for exploration and learning of the architectural design space”

Delvist enig. I min erfaring bliver prototyper oftest brugt til at undersøge en bestemt arkitektur og der bliver kun undersøgt flere arkitekturer hvis den første fejler. Der stoppes mao. lige så snart et tilfredsstillende resultat er nået.

“Architectural prototyping addresses issues regarding architectural quality attributes in the target system”

Helt enig. Dette var netop formålet med den beskrevne prototype og den måde arktekturprototyper oftests benyttes i min erfaring.

“Architectural prototypes do not provide functionality per se”

Jeg er enig i at der ikke nødvendigvis følger funktionalitet med en arktitekturprototype. Jeg har dog erfaringer for at de tit vokser til at blive en del af applikationen. I det beskrevne scenario blev prototypen brugt som skeletsystem hvor i resten af den del af applikationen blev bygget.

“Architectural prototypes typically address architectural risks”

Enig. I min organisation bruges de oftest til at undersøge performance. Så vidt jeg kan vurdere fordi det er den kvalitet som udviklere er mest fortrolig med.

“Architectural prototypes address the problem of knowledge transfer and architectural conformance”

Enig. I det beskrevne scenario blev prototypen netop brugt som en skeletapplikation som andre udviklere kunne starte ud fra.

2.5 Diskussion

"Discuss the advantages and disadvantages of architectural prototypes in general and in relation to the cases you introduced":

Som det ses af ovenstående så har arkitektoniske prototyper været et godt værktøj for gruppens medlemmer til at afdække om visse arkitektoniske valg var farbare eller ej. En stor fordel ved arkitektoniske prototyper er at det med et minimum af indsats afdækkes om arkitekturen kan levere de ønskede kvalitetsatributter. Dette ses også af ovenstående cases, hvor prototyperne typisk har været benyttet til at validere om en tænkt arkitektur tilbyder tilstrækkelig performance.

Dette er måske også en ulempe ved arkitektoniske prototyper, i det de synes at fokusere på "let fattelige" kvaliteter som *performance* og *buildability* og måske mindre på andre kvaliteter som fx. *security* og *modifiability*. Ingen i gruppen havde fx. prøvet at bygge en prototype med det formål at se hvor hurtigt en udvikler kunne udvikle nyt på basis af denne.

Denne fokus på fx. performance kan lede en til at tro at en arkitektur som opfylder performance er det korrekte arkitekturvalg, selv om dette ikke er tilfældet. De kvaliteter som prototypen undersøger kan mao. komme til at overskygge andre (vigtige) kvaliteter.

De metoder som er beskrevet i dette dokument (ATAM, CBAM, aSQA) kan hjælpe med at sikre at fokus holdes på alle relevante aspekter af designet.

Et andet problem med arkitekturprototyper er at de (i visse organisationer) har en tendens til at blive fundament for det færdige produkt, selv om dette ikke var tiltænkt. Dette resulterer tit i et produkt hvor fokus for arkitekturprototypen (en protokol, et bestemt pattern mv.) er lavet "pænt" og resten af arkitekturen fremstår mindre gennemtænkt og gennemført.

Det kan være svært at opnå gehør for at udført arbejde (prototypen) skal smides væk bagefter, især hos ledelse med mindre god forståelse for software design processen.

Det er derfor vigtigt at formålet med prototypen er veldefineret før arbejdet sættes i gang.

3 References

[Bass et al. 2007] Bass, L., Clements, P., and Kazman, R. (2007). *Software Architecture in Practice*, Addison-Wesley, second edition.