SOAPIF

ETT RAMVERK FÖR UTVECKLING AV TJÄNSTEORIENTERADE API:ER

Magisteruppsats i Informatik

Marcus Blomberg
Per Davidsson

VT 2009:MI03

HÖGSKOLAN I BORÅS
INSTITUTIONEN FÖR DATA- OCH AFFÄRSVETENSKAP

Svensk titel: SOAPIF – Ett ramverk för utveckling av tjänsteorienterade API:er

Engelsk titel: SOAPIF – A Framework For Development of Service-Oriented API's

Utgivningsår: 2009

Författare: Marcus Blomberg & Per Davidsson

Handledare: Anders Hjalmarsson

Abstract
This report is written in Swedish.

This study aimed at developing a framework for development of service-oriented API's, and is based on a case study of the development of a service-oriented integration platform (e-Me) which purpose is to simplify students' electronic errands, and a benchmark of best-of-the-class API's. This study has 5 major phases: The theoretical study, a benchmark of best-of-the-class API's, two qualitative interview studies, and the development of the framework itself. The theoretical study discusses topics such as development methodologies, system architectures (most notably SOA), and method engineering. This theoretical study was the foundation for the first qualitative interview study, which in turn was the foundation for the benchmark. Both these studies are the foundation for the first version of SOAPIF, the Service-Oriented Application Programming Interface Framework. SOAPIF contained four phases: *Conceptualization* (conceptual modeling of the API, based on co-design activities), *Definition* (define the API contract), *Testing & Implementation* (implement the API using Test-Driven Development), and *Delivery* (deliver the final contract for the current version of the API, along with extensive documentation). In all of these phases, documentation and co-design are essential for the successful delivery of the API. After a qualitative validation of the framework, phases where renamed steps. These steps match the phases above, with one addition: A fifth step called *API Evaluation*. This fifth step was added to ascertain that API's developed using SOAPIF is coherent with consumer expectations. Based on the results of the different studies, the following conclusions have been reached: It's utterly important to develop flexible API's, which are responsive to shifts in consumer expectations. It's also extremely important that the API's are extensively documented, to ensure that consumers can easily use them in their own applications. We also concluded that the main reason for developing integration platforms is the need of flexibility.

Keywords: API, SOA, interface, co-design, tests, agile, framework, documentation, outside in

Sammanfattning

Denna studie syftade till att ta fram ett ramverk för utveckling av API:er för tjänsteorienterade arkitekturer. Studien tar sin utgångspunkt i forskningsprojektet e-Me (utvecklingen av en integrationsplattform för att underlätta studenters vardag) samt en benchmarking av de i nuläget mest använda API:erna på webben. Studien har fem faser: En litteraturstudie, en benchmarking, en kvalitativ genererande intervjustudie, en teoribildande fas (framtagning av ramverket), samt en kvalitativ validerande studie. Litteraturstudien tar upp sådana fenomen som utvecklingsmetoder, systemarkitekturer (bland annat SOA) samt metautveckling. Denna studie låg till grund för den första intervjustudien, som i sin tur låg till grund för benchmarkingen. Resultatet av dessa två studier bildade tillsammans en grund för ramverket SOAPIF (Service-Oriented Application Programming Interface Framework), som till en början innehöll fyra olika faser: *Conceptualization*, där man försöker bilda sig en uppfattning om vad API:et skall exponera; *Definition*, där man försöker definiera ett kontrakt för API:et; *Testing & Implementation* som tar sin utgångspunkt i tester för att implementera API:et; samt *Delivery*, som är den fas där API:et skall levereras. I alla dessa faser ingick kontinuerligt dokumentation samt co-design. Efter en validering av ramverket byttes de fyra faserna ut mot fem arbetsmoment, med samma namn som faserna ovan, med ett tillägg: Ett arbetsmoment som heter *API Evaluation*, och som syftar till att efter leveransen tillsammans med konsumenterna av API:et utvärdera det. Utifrån de resultat som denna studie genererat, har följande slutsatser kunnat dras: Det är oerhört viktigt med flexibla API:er som utvecklas i samarbete med den tänkta slutkonsumenten, och att dessa API:er testas mot de förväntningar som konsumenterna har på dem. Det är också oerhört viktigt att API:er är väldokumenterade, för att underlätta för konsumenter. När det gäller integrationsplattformar är anledningen till att man utvecklar dessa främst ett behov av flexibilitet.

Nyckelord: API, SOA, gränssnitt, co-design, tester, agile, ramverk, dokumentation, outside in

FÖRORD

Vi vill börja med att citera en av de största poeterna genom tiderna, Bob Dylan (1964):

Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'

Vi vill främst tacka vår handledare Anders Hjalmarsson, för hans goda råd och entusiasm i arbetet med denna studie. Vi vill även tacka Daniel Rudmark, för att han ledde oss på rätt väg i början av denna långa resa. Vi vill även tacka våra respondenter, Linus Andrén, Göran Golcher samt ovan nämnda Daniel, för att de tog sig tid att svara på våra frågor.

Slutligen vill vi även tacka musiken som har hjälpt oss att få den extra kraft som gjort denna studie möjlig: Tack Bob, Dexter, Michael, Bruce, Johnny, Roy, John, Don, och alla andra, för ert aldrig sviktande stöd.

Marcus Blomberg Per Davidsson

INNEHÅLLSFÖRTECKNING

Figurförteckning

Tabellförteckning

1 INLEDNING

Detta kapitel behandlar studiens bakgrund, problemformulering, syfte, frågeställning och avgränsning. Detta kapitel syftar till att ge läsaren en inledande uppfattning om vad studien och denna uppsats behandlar, för att underlätta fortsatt läsning av uppsatsen.

A journey of a thousand miles begins with a single step.
\- Lao-tzu

1.1 Bakgrund

Informatik definieras vid Högskolan i Borås (2002) som

> en samhällsvetenskap som studerar användning och utveckling av informationsteknik i verksamheter.

Enligt Beynon-Davies (2002, sid. 3) är informatik den vetenskap som studerar information, informationssystem och informationsteknik applicerad på olika fenomen. Enligt honom är informationsteknik de teknologier som används för att understödja informationsinsamling, -behandling, -distribution och –användning. Informationsteknik innehåller verktyg för att konstruera informationssystem, men är inte samma sak. Modern informationsteknik består av hårdvara, programvara, data- och kommunikationsteknologi. Ett informationssystem är enligt Beynon-Davies (2002, sid. 4) ett system för kommunikation mellan människor. Informationssystem är inblandade i insamling, bearbetning, distribution och användning av data, men skall inte förväxlas med informationsteknik, som har ett bredare spektrum.

Det finns ett stort utbud av informationssystem idag både på Internet och lokalt, och detta utbud växer ständigt. Det är därför viktigt att kunna kommunicera tjänster emellan. Ett sätt att kommunicera mellan tjänster och olika informationssystem på Internet är att tillämpa systemarktitekturstrategin *Service-Oriented Architecture*, SOA. Enligt Nordqvist (2006) är SOA ett sätt att integrera och organisera en verksamhets IT-stödda affärsprocesser.

För att framgångsrikt implementera SOA är det viktigt att man använder sig av *Service-Oriented Modeling* (SOM). De två mest populära verktygen inom SOM är *Service-Oriented Modeling and Architecture* (SOMA), en metod som har tagits fram av IBM, och *Service-Oriented Modeling Framework* (SOMF). (Wikipedia 2009) SOMA är en utökning av traditionella objektorienterade och komponentbaserade systemutvecklingsmetoder, som är utökad med komponenter för att utveckla för SOA. (Zimmermann, Krogdahl & Gee 2004) SOMF är ett förslag till ett ramverk som innehåller ett universellt språk och olika discipliner för att tillhandahålla taktiska och strategiska lösningar till verksamhetsproblem. (Wikipedia 2009)

Ett sätt att använda SOA är att tillhandahålla en integrationsplattform, som leverantörer kan integrera tjänster i. En integrationsplattform bör enligt Wipcore vara en central del av en organisations IT-miljö. En integrationsplattform är en mjukvara som bland annat sköter och övervakar kommunikation, mappning och meddelanden mellan olika system. Att införa en integrationsplattform är inte alltid lätt, men man löser oftast många kommunikationsproblem. (Wipcore 2008) För att externa leverantörer skall kunna utveckla tjänster för en integrationsplattform tillhandahålls oftast ett gränssnitt (kallat API, *Application Programming Interface*) för leverantören. Detta gränssnitt definierar hur tjänsten skall kommunicera med integrationsplattformen, samt med andra tjänster byggda på samma integrationsplattform. (Two Crows 2009)

På InnovationLab[1] vid Högskolan i Borås utvecklas just nu e-Me, en integrationsplattform för förenkling av en vanlig students vardag. Tanken med e-Me är att varje student skall få en dedikerad, personlig medhjälpare på Internet, som håller koll på till exempel studentens ekonomi och studier. Tanken med e-Me som integrationsplattform är att många olika leverantörer skall kunna implementera tjänster specifikt för e-Me, som studenten sedan kan ta del av. Dessa tjänster skall kommunicera med e-Me via ett API. (e-Me 2009)

1.2 Problemformulering

I dagens samhälle överger tillverkare alltmer den traditionella utvecklingen av produkter. I den traditionella utvecklingen undersöker först tillverkaren kundens behov, för att sedan utveckla produkter som fyller dessa behov. Detta angreppssätt är dock varken billigt eller tidseffektivt, dessutom förändras kundernas behov i en allt högre hastighet. Därför överger fler och fler tillverkare detta angreppssätt, och tillverkarna överför istället de behovsrelaterade aspekterna av en produkt till själva kunden: Kunden skapar själv den produkt den vill ha, med hjälp av så kallade *toolkits for user innovation* (verktygslådor för användarinnovation). (von Hippel & Katz 2002, s. 821)

Detta är sant även för informationssystem, och speciellt de sociala informationssystem som används flitigt på Internet, till exempel *Facebook*. På Facebook skapar användaren sitt eget informationssystem, och bestämmer själv vilka applikationer och funktioner som den vill ha. (Facebook 2009a) Ett API tillhandahålls av Facebook, så att utvecklare kan utöka funktionaliteten hos informationssystemet med fler applikationer, som användaren sedan kan infoga i sin egen del av informationssystemet. Facebook använder sig av en tjänsteorienterad arkitektur (en SOA), och är en integrationsplattform. (Facebook 2009b)

SOA har i flera år varit ett uppmärksammat ämne inom IT-branschen, och många organisationer har tagit det som en frälsning från ovan, utan att tänka på att även en SOA kan vara problematisk. En SOA kan lätt bli väldigt komplex och svår att överblicka, och innehåller oftast bara lösa kopplingar, vilket kan vara en nackdel vid buggar, då orsaken till dessa kan bli svårare att upptäcka. Att ha lösa kopplingar är dock ur ett annat perspektiv bra, då man lätt kan förändra en del av ett informationssystem eller arkitekturen utan att

[1] InnovationLab är ett systemutvecklingscenter som arbetar med Academic Computing, alltså IT-stöd för forskning, utbildning och administration inom den akademiska världen. (InnovationLab 2009)

behöva distribuera hela systemet eller arkitekturen på nytt. (Mackie 2007) För att undvika de nackdelar som finns med en SOA, samt att fullt ut ta vara på dess möjligheter, krävs det dock enligt Christer Ahlstedt[2] att man planerar, modellerar och designar sin SOA på ett bra sätt.

De accepterade metoder och ramverk som finns för att skapa en bra SOA (eller en bra integrationsplattform eller ett bra API) är dock relativt få och obeprövade. Den mest beprövade metoden är SOMA (Service-Oriented Modeling and Architecture) som har utvecklats av IBM, och den har genomgått flera cykler av förändringar, utifrån den feedback som IBM har fått från de organisationer som har använt den. Metoden har använts i några hundra projekt, dock är den endast fem år gammal, och även den relativt obeprövad. (Arsanjani et al. 2008, s. 395) Det finns också ett ramverk för utveckling av tjänstelösningar, detta är ännu mer obeprövat. Detta ramverk heter *Service-Oriented Modeling Framework* (SOMF) och har föreslagits av Bell (2008), och är alltså ännu nyare än SOMA.

1.3 Syfte

Syftet med denna studie är att ge råd om hur man modellerar, utformar och implementerar API:er, med fokus på API:er för SOA. Studien syftar till att fylla ett tomrum som finns inom området, då det inte finns någon metod eller något ramverk som beskriver hur man utvecklar just API:er för SOA.

1.4 Forskningsfrågor

Utifrån bakgrunden, problemformuleringen samt rapportens syfte har vi formulerat följande övergripande forskningsfrågor:

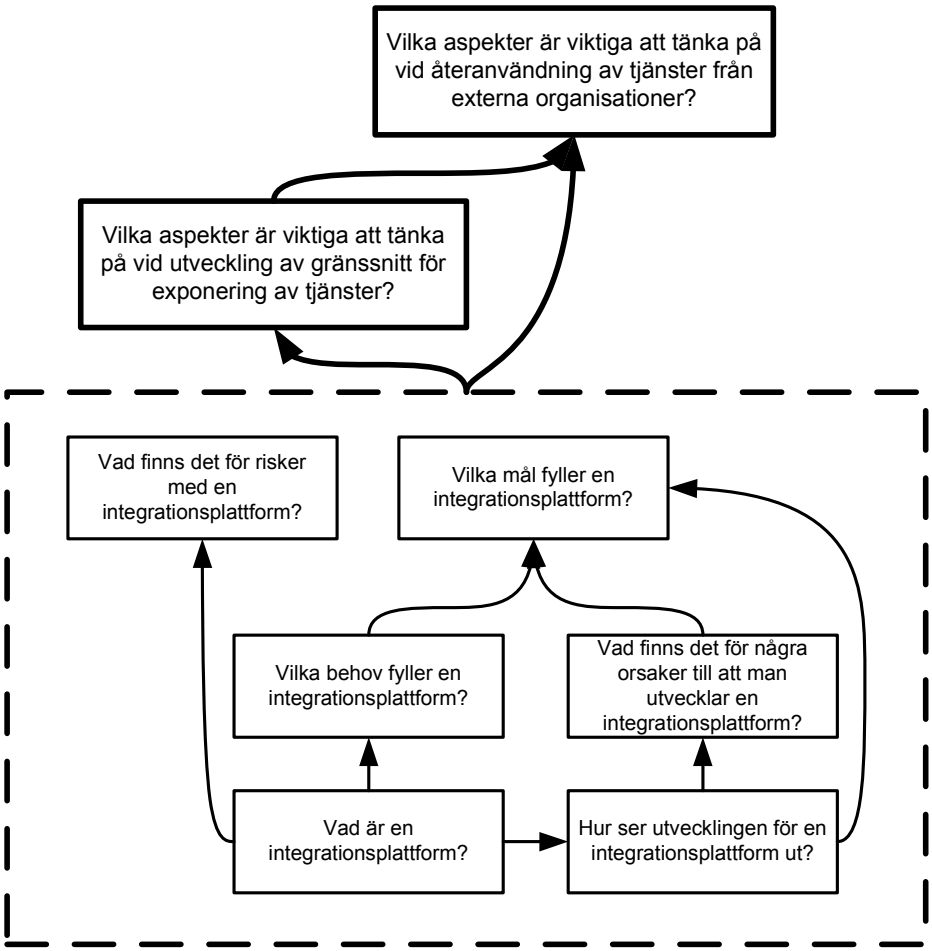
- *Vilka aspekter är viktiga att tänka på vid utveckling av gränssnitt för exponering av tjänster?*
- *Vilka aspekter är viktiga att tänka på vid återanvändning av tjänster från externa organisationer?*

För att svara på dessa frågor har vi formulerat några delfrågor, som ligger till grund för de två övergripande frågornas resultat:

- *Vad finns det för risker med en integrationsplattform?*
- *Vilka mål fyller en integrationsplattform?*
- *Vilka behov fyller en integrationsplattform?*
- *Vad finns det för några orsaker till att man utvecklar en integrationsplattform?*
- *Vad är en integrationsplattform?*
- *Hur ser utvecklingen för en integrationsplattform ut?*

Frågornas inbördes relationer till varandra visas i *Figur 1: Studiens forskningsfrågor*.

[2] Christer Ahlstedt, Sälj- och marknadschef, IBS Konsult AB. Föreläsning den 18 november 2008.

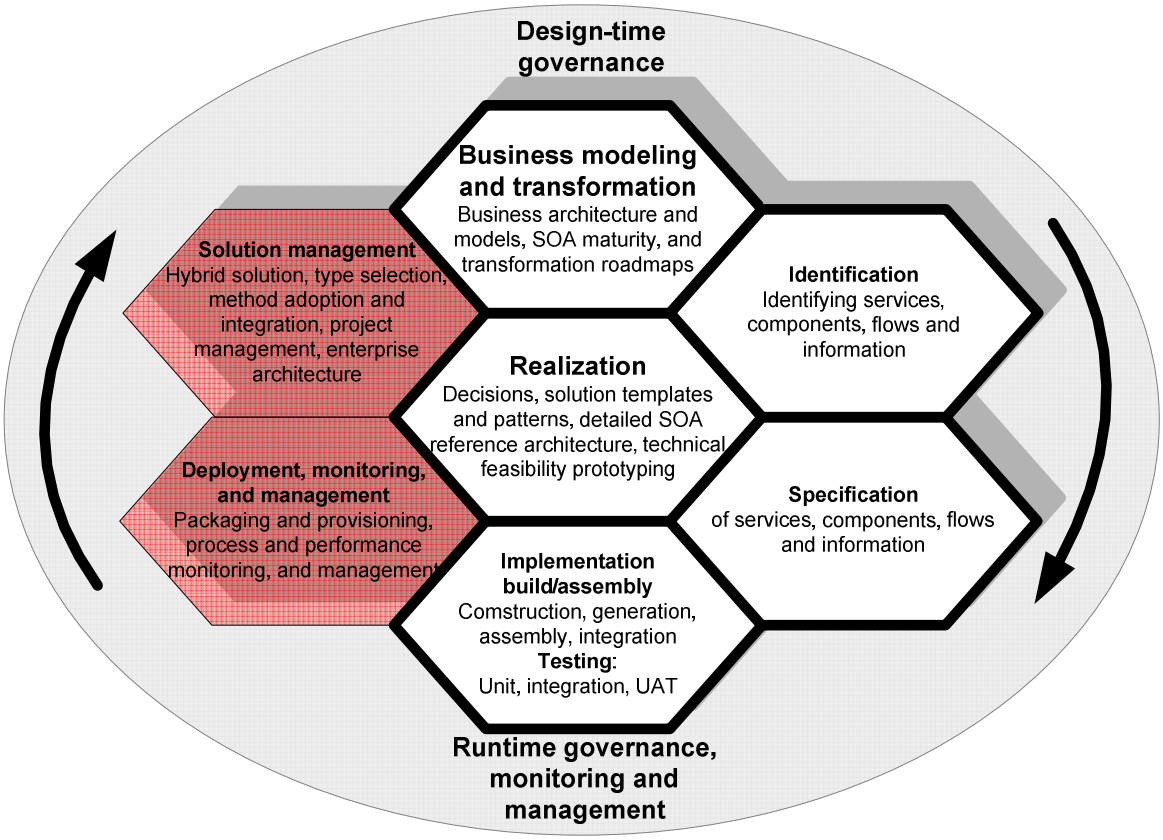


Figur 1: Studiens forskningsfrågor

Som synes i *Figur 1: Studiens forskningsfrågor* ligger de sex delfrågorna till grund för de övergripande, huvudsakliga forskningsfrågorna.

1.5 Avgränsning

Denna studie kommer att behandla delar av faserna *Business modeling and transformation, Identification, Specification, Realization* och *Implementation Build/Assembly & Testing* i SOMA (SOMF ingår i denna studie som en del av *Business modeling and transformation*), vilket syns i *Figur 2: Studiens avgränsning av faserna i SOMA*, samt endast inom projektet e-Me. Denna avgränsning är nödvändig, då projektet e-Me ej kommer att vara avslutat vid tiden för denna studies examination. Vi riktar därför in oss på de tidiga faserna i projektet. Vi avgränsar oss med hjälp av denna metod, då den syftar till att ta fram tjänsteorienterade lösningar. För en beskrivning av metoden, se *3.3.4 SOA*.



Figur 2: Studiens avgränsning av faserna i SOMA

1.6 Intressenter

Intressenter för denna studie bör vara systemutvecklare som arbetar med API:er och integrationsplattformar, men även organisationer som använder SOA. Även forskare som forskar om fenomenet SOA kan vara intresserade av denna studie. Studien bör vara speciellt intressant för de utvecklare som arbetar med projektet e-Me, då denna studie berör dem, och projektet.

2 VETENSKAPLIGT FÖRHÅLLNINGSSÄTT OCH METOD

I detta kapitel beskrivs det vetenskapliga förhållningssätt samt de vetenskapliga metoder och den strategi som har använts under studien.

I will prepare and someday my chance will come.
- Abraham Lincoln

2.1 Kunskapskaraktärisering

Enligt Goldkuhl (1998) innebär kunskapskaraktärisering att man anger den typ av kunskap som skall utvecklas i ett kunskapsarbete. Enligt honom behöver man karaktärisera den kunskap som genom studien skall genereras, för att kunna värdera denna nya kunskap. Denna karaktärisering är också viktigt för att kunna skapa en strategi för själva studien. I alla kunskapskaraktärer ingår kategoriell kunskap, det vill säga att man begreppsliggör världen. I kapitlet *1.4 Forskningsfrågor* identifierade vi åtta olika kunskapsbehov. Den kunskap som de förväntas generera karaktäriseras i *Tabell 1: Kunskapskaraktärisering av studiens forskningsfrågor*. Alla dessa frågor är i grund och botten kategoriserande.

Kunskapsbehov	**Karaktärisering**
Vilka aspekter är viktiga att tänka på vid återanvändning av tjänster från externa organisationer?	Normativ kunskap Karaktäriserande kunskap
Vilka aspekter är viktiga att tänka på vid utveckling av gränssnitt för exponering av tjänster?	Normativ kunskap Karaktäriserande kunskap
Vad finns det för risker med en integrationsplattform?	Deskriptiv kunskap Förklarande kunskap Karaktäriserande kunskap
Vilka mål fyller en integrationsplattform?	Deskriptiv kunskap Förklarande kunskap Karaktäriserande kunskap
Vilka behov fyller en integrationsplattform?	Deskriptiv kunskap Förklarande kunskap Karaktäriserande kunskap
Vad är en integrationsplattform?	Deskriptiv kunskap Karaktäriserande kunskap
Vad finns det för några orsaker till att man utvecklar en integrationsplattform?	Deskriptiv kunskap Förklarande kunskap Karaktäriserande kunskap

Hur ser utvecklingen för en integrations-plattform ut?	Förutsägande kunskap Förklarande kunskap

Tabell 1: Kunskapskaraktärisering av studiens forskningsfrågor

2.1.1 Normativ kunskap

Normativ kunskap är enligt Goldkuhl (1998) kunskap som skall vara vägledande, och som talar om hur man *bör* handla i olika situationer. Exempel på normativ kunskap är metoder och modeller såsom SIMM[3]. De två huvudsakliga forskningsfrågorna i denna studie syftar till att ta fram just sådan normativ kunskap: Vad bör man tänka på när man utvecklar ett gränssnitt för integration av externa tjänster, och vad bör man tänka på vid återanvändning av tjänster från externa organisationer?

2.1.2 Förklarande kunskap

Förklarande kunskap innebär att man förklarar *varför* något är på ett visst sätt, och man försöker ofta hitta orsaker till varför något är som det är. (Goldkuhl 1998) I denna studie vill vi bland annat få kunskap om varför man utvecklar en integrationsplattform (vilka behov och mål den fyller), vad det finns för några orsaker till att man utvecklar den, samt vilka risker det finns med att utveckla en integrationsplattform. Den kunskap som förväntas genereras från dessa frågeställningar är förklarande.

2.1.3 Deskriptiv kunskap

Deskriptiv kunskap beskriver egenskaper hos en studerad företeelse. (Goldkuhl 1998) I denna studie är den företeelse som studeras en integrationsplattform, och genom att beskriva vad en integrationsplattform egentligen är byggs en grund på vilken studien kan byggas vidare. Vi vill också beskriva de risker, mål och behov som en integrationsplattform uppfyller, samt även de orsaker som finns till att man utvecklar en sådan.

2.1.4 Förutsägande kunskap

Förutsägande kunskap är en typ av kunskap som syftar till att förutsäga framtiden. Denna typ av kunskap applicerar oftast andra typer av kunskap på en specifik situation, för att därigenom försöka göra förutsägelser. (Goldkuhl 1998) Denna typ av kunskap förväntas genereras genom frågeställningen *Hur ser utvecklingen för en integrationsplattform ut?*, det vill säga att vi vill skapa förutsägande kunskap om hur en sådan utveckling kan se ut i framtiden.

2.1.5 Karaktäriserande kunskap (förståelseinriktad kunskap)

Karaktäriserande (förståelseinriktad) kunskap försöker beskriva vad ett fenomen är, och tilldela innebörder till ett fenomen. Denna typ av kunskap är en förutsättning för förklarande kunskap: För att kunna förklara varför något är som det är, behöver man först veta vad något är. (Goldkuhl 1998) Vi vill ge en förståelse för vad en integrationsplattform är, samt varför man utvecklar den, och vad det finns för några risker, mål, behov och orsaker till den. Detta anser vi vara förståelseinriktad kunskap, då den skapar förståelse för vad en integrationsplattform är. Vi ämnar också skapa förståelse för vad man bör tänka på när

[3] Metod för grafnotation

man utvecklar API:er för integration av tjänster, samt vad man bör tänka på vid återanvändning av tjänster. Också detta är förståelseinriktad kunskap.

2.2 Vetenskapligt förhållningssätt

För att förhålla sig till vetenskap finns det i huvudsak två synsätt: *Hermeneutik* samt *positivism*. Hermeneutiken är inriktad på att förstå och tolka, medan positivismen i större utsträckning endast beskriver det man studerar. Positivismen är naturvetenskapligt inriktad, medan hermeneutiken är mer samhällsvetenskapligt inriktad. Positivismen handlar också om att studera forskningsobjektet i mindre delar, och studera var del för sig, medan hermeneutiken fokuserar på att tolka helheten. (Bryman 2002, s. 24-25)

I vår studie inriktar vi oss mest på karaktäriserande (förståelseinriktad) kunskap, och anlägger därmed ett hermeneutiskt synsätt: Vi vill studera och förstå hur man tar fram en integrationsplattform, vilka orsaker, mål och behov som ligger bakom. Därefter vill vi ta fram ett ramverk, som utifrån denna studie och förståelse ger råd om hur man skall utveckla ett gränssnitt för integration av externa tjänster.

2.3 Forskningsansats

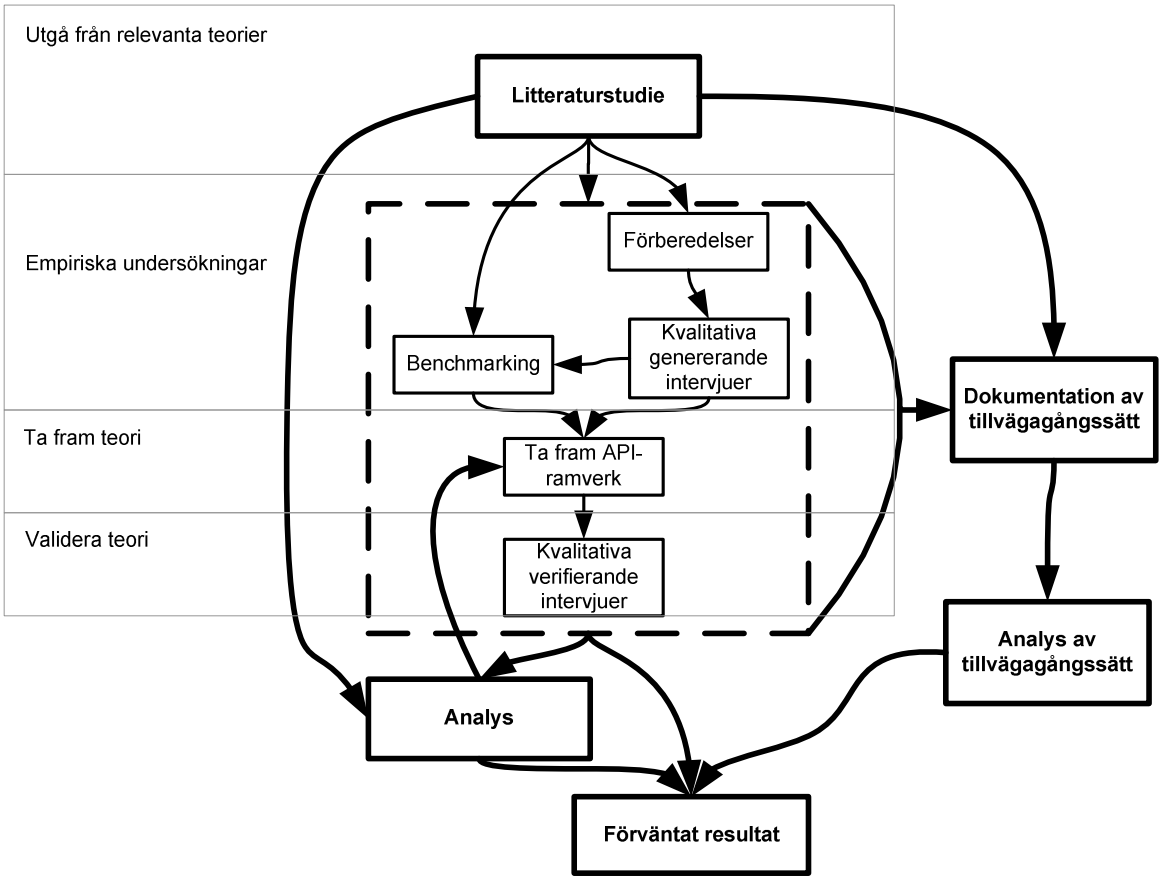
Det finns huvudsakligen två ansatser vid skapandet av ny kunskap: Induktion och deduktion. En kombination av de båda kallas för abduktion. Deduktion representerar den vanligaste samhällsvetenskapliga tolkningen om relationen mellan teori och praktik. Inom deduktionen utgår man som forskare från teorin, och tar fram en hypotes, för att sedan empiriskt undersöka denna hypotes. Induktion jobbar från andra hållet: Genom empiriska undersökningar tar forskaren fram en teori. (Bryman 2002, s. 21-23) En kombination av dessa två ansatser kallas som sagt för abduktion. (Le Duc 2007) I denna studie kommer en abduktiv metodansats att användas, då denna ansats täcker in de metoder som krävs för studiens färdigställande.

Vi har använt en abduktiv strategi (som framgår i *Figur 3: Studiens abduktiva forskningsstrategi*), då vi genom deduktion tog fram en hypotes genom en litteraturstudie. Denna låg sedan till grund för en induktiv ansats, där vi genomförde två empiriska studier för att ta fram en teori. Denna teori validerades sedan genom ytterligare en empirisk studie, vilket även den bidrog till att stärka den abduktiva ansatsen. Vi har jobbat från båda tolkningarna, och grundat vår teori genom både induktion och deduktion. Vi har därigenom använt oss av en abduktiv ansats.

2.4 Strategi

Studien tar sin utgångspunkt i en litteraturstudie, som ligger till grund för kvalitativa genererande intervjuer. Resultatet av dessa intervjuer ligger dels till grund för det ramverk studien förväntas utmynna i, dels för den benchmarking som även den kommer att hjälpa till med att ta fram ramverket. Denna benchmarking har också den input från litteraturstudien. Efter dessa två aktiviteter tas ett förslag på ett API-ramverk fram, och därefter valideras förslaget genom fler kvalitativa intervjuer, som återkopplar till de tidigare intervjuerna. Efter detta ses API-ramverket över, för att sedan analyseras. Under hela denna studie dokumenteras och analyseras också författarnas arbetssätt, så att detta sedan kan

utvärderas. Alla dessa olika resultat sammanställs sedan för att kunna presenteras. Därefter utvärderas arbetssättet.



Figur 3: Studiens abduktiva forskningsstrategi

Som synes i *Figur 3: Studiens abduktiva forskningsstrategi*, har denna studie en abduktiv forskningsansats. Genom att studera relevant teori inom området, och senare använda denna teori som grund för en empirisk studie, förväntas en grund ges för en ny teori. Denna teori undersöks sedan empiriskt, för att validera den.

2.5 Insamlingsmetod

För att skapa en grundförståelse för ämnesområdet och de fenomen vi vill studera, genomförs i början av studien en litteraturstudie. Denna litteraturstudie fokuserar främst på det relativt breda området *Service Oriented Architecture*, SOA, samt ramverket *Service-Oriented Modeling Framework*, SOMF, och metoden *Service-Oriented Modeling and Architecture*.

Efter litteraturstudien kommer en kvalivativ intervjustudie att genomföras, där systemutvecklarna inom e-Meprojektet intervjuas om e-Me som API och integrationsplattform. Genom denna studie hoppas vi få fram material som kan stödja oss i nästa fas i vår studie,

att konstruera ett ramverk för utveckling av API:er. Denna intervjustudie kommer också att ge input till den benchmarking av framgångsrika API:er som kommer att genomföras.

I en benchmarking undersöker man de ledande och mest framgångsrika fenomenen inom ett område, för att applicera samma metodik och tillvägagångssätt som har använts på sin egen lösning, för att därigenom skapa en så god produkt som möjligt. Man lär helt enkelt av *the best of the class*, och applicerar detta på sin egen verksamhet. I detta fall ämnar vi undersöka olika API:er och integrationsplattformar, till exempel Google Maps.

Efter intervjustudien och benchmarkingen ämnar vi konstruera ett ramverk för tjänsteorienterade API:er, med intervjustudien och benchmarkingen som input. Efter att detta ramverk har konstruerats valideras det genom ytterligare en intervjustudie med utvecklarna i e-Meprojektet.

2.5.1 Litteraturstudie

I litteraturstudien kommer vi att försöka skapa oss en överblick av det som tidigare publicerats genom att undersöka litteratur, artiklar rapporter och fackskrifter om SOA, SOMF, och SOMA. Vi kommer även att använda oss av sökmotorn Google (främst Google Scholar) för att hitta relevant information om aktuell litteratur. För att få tag på den fysiska litteraturen använder vi oss av Högskolan i Borås bibliotekskatalog. Vi kommer även att noga värdera informationen med hjälp av kriterier för källkritik.

Källkritik

För källkritik finns det enligt Leth och Thurén (2000) fyra olika kriterier:

- **Äkthet**. Det är viktigt att försöka avgöra om informationen är äkta eller ej, eller om den på något sätt är förfalskad. Detta kan man göra genom att till exempel jämföra med andra källor om liknande fenomen, och se om det finns några diskrepanser.
- **Tid**. Tidsaspekten innebär att den som söker information måste beakta hur lång tid det tog från det att informationen skapades tills den faktiskt dokumenterades.
- **Beroende**. Det är viktigt att den person som har dokumenterat informationen själv har tagit fram den, eller iakttagit det som ledde fram till den, och inte endast hört det från någon annan källa.
- **Tendens**. Det är viktigt att försöka uppfatta tendensen i information. Med tendens menas att författaren kanske inte är helt objektiv, och har egenintresse av att en sak framställs på ett visst sätt.

Det är även viktigt att skilja mellan primärkällor, sekundärkällor och tertiärkällor, det vill säga att informationen inte skall ha passerat för många källor. För Internetkällor tillkommer dessutom enligt Leth och Thurén (2000) tre kriterier:

- **Världsbild och kunskapssyn som tendens**. Detta kriterium knyter tillbaka till tendenskriteriet, men tar också in en lokaliserad aspekt: Hur ser världsbilden och kunskapssynen ut i olika delar av världen, och inom olika discipliner? Detta är viktigt att beakta.

- **Trovärdighet**. Det är viktigt att värdera trovärdigheten hos en webbplats: Vem står bakom den? Har de några egenintressen? Verkar det vara en seriös forskare eller organisation?
- **Källans förutsättningar och egenskaper**. Det är viktigt att tänka på de förutsättningar och egenskaper som källan har: Är det rimligt att just denna källa ska ha kunnat få tillgång till eller generera denna information?

Dessa kriterier kommer vi att beakta i vår litteraturstudie, och under vårt sökande efter information. Vi söker främst efter vetenskapliga artiklar och vetenskaplig litteratur, och värderar dessa främst efter trovärdighet, äkthet och tendens.

2.5.2 Kvalitativa intervjuer

När vi intervjuar våra respondenter kommer vi använda oss av ett kvalitativt tillvägagångssätt. Vi kommer att använda oss av semistrukturerade intervjuer för att skapa en bra stämning och få respondenten att känna sig bekväm i situationen. Semi-strukturerade intervjuer innebär att forskaren har en lista över olika teman som ska tas upp under intervjutiden och respondenten har stor frihet i hur han/hon levererar svaren. Frågorna behöver inte alltid komma i samma ordning som det var tänkt tidigare. Intervjuprocessen ska vara flexibel och tonvikten ska ligga på hur respondenten uppfattar frågorna och vad den tycker är viktigt i dem. (Bryman 2002, s. 301)

För att förbereda intervjuerna kommer vi att skapa en intervjuguide. En intervjuguide är en typ av minneslista över vilka områden som ska tas upp på intervjun. Det som är viktigt är att frågorna tillåter forskaren att få den information som han/hon är ute efter men även hur respondenten upplever sin värld och sitt liv, vilket gör intervjuerna mer flexibla. (Bryman 2002, s. 304-305) Följande kriterier, som Bryman (2002, s.305) beskriver skall vi använda oss av vid skapandet av vår intervjuguide:

- Ha en ordning över vilka teman som skall tas upp och hur dem följer varandra, men även ha i åtanke att ordningen kan ändras.
- Formulera frågorna så det blir enklare att svara på forskningsfrågorna, men inte ha dem allt för specifika.
- Använd ett enkelt och begripligt språk som passar respondenterna.
- Ställ inte några ledande frågor till respondenterna.
- Fråga om bakgrundsfakta så som kön, ålder och namn och vad personen jobbar med. Detta gör att man kan sätta in svaren från respondenten i ett sammanhang.

Före intervjun finns det enligt Bryman (2002, s. 305-306) några viktiga saker att tänka på för att få ut det bästa av intervjun och respondenten:

- Undersök miljön där intervjun ska äga rum, för att underlätta tolkningen av svaren som respondenten ger.
- Ha en bandspelare tillhands som fungerar, en sådan är bra för en enklare analys.
- Utför intervjun på en lugn och behaglig plats där respondenten känner sig trygg och vet att ingen förutom forskarna kan höra honom/henne.

Vi kommer i vår studie att genomföra intervjuer i två omgångar: En omgång med kvalitativa genererande intervjuer, vars resultat skall hjälpa oss att ta fram ett ramverk, och en omgång med kvalitativa verifierande intervjuer, som skall hjälpa oss verifiera detta ramverk.

Under interjuerna kommer vi att följa de kriterier för intervjuer som tidigare satts upp. Vi kommer att börja med att göra en intervjuguide, som strukturerar upp de fenomen vi vill ta upp under intervjuerna. Denna är dock ej huggen i sten, utan kan ändras. Vi kommer också att tänka på det språk vi använder under intervjuerna, och anpassa oss efter respondenterna, samt att inte själv vara för långdragna. Vi skall också försöka att inte ställa några ledande frågor, detta för att få ut respondentens egen syn på saker.

I början av intervjun kommer vi att ställa frågor som rör respondenten själv, frågor som är lätta att svara på och som gradvis leder in på rätt område. Vi kommer också att genomföra intervjuerna på respondenternas egen arbetsplats, så att de känner sig bekväma. Intervjuerna genomförs med bandspelare, så att vi kan fokusera på att få ut så mycket relevant information från respondenten som möjligt.

Kvalitativa genererande intervjuer

De kvalitativa genererande intervjuerna syftar tillsammans med benchmarkingen till att ge ett underlag för att ta fram ett API-ramverk. Under intervjun kommer vi att ta upp vissa teoretiska begrepp och modeller, därför anser vi att det finns ett behov av att förbereda respondenterna innan intervjuerna. Detta planerar vi göra genom att skapa ett startpaket, som e-postas till respondenterna före intervjun. Detta paket innehåller information om de teoretiska begrepp och modeller vi planerar att ta upp, så att respondenterna kan vara förberedda inför intervjun. Detta gör att vi kan fokusera på att prata om respondenternas egna erfarenheter och synpunkter under intervjuerna, i stället för att fokusera på att förklara dessa begrepp. I *Tabell 2: De kvalitativa genererande intervjufrågornas koppling till studiens forskningsfrågor* visas de övergripande frågor vi planerar att ställa under denna första intervju, samt dessa frågors koppling till våra forskningsfrågor.

Forskningsfråga	**Intervjufrågor**
Vilka aspekter är viktiga att tänka på vid återanvändning av tjänster från externa organisationer?	- Hur hanteras tjänsterna i e-Me?
Vilka aspekter är viktiga att tänka på vid utveckling av gränssnitt för exponering av tjänster?	- Av de erfarenheter du har fått genom arbetet med e-Me + SOA, vilka du lyfta fram?
Vad finns det för risker med en integrationsplattform?	- Skulle du kunna ge en kort beskrivning av hur du uppfattar projektet e-Me, och vad din roll i projektet är?
Vilka mål fyller en integrationsplattform?	- Skulle du kunna ge en kort beskrivning av hur du uppfattar projektet e-Me, och vad din roll i projektet är?

Vilka behov fyller en integrationsplattform?	- Skulle du kunna ge en kort beskrivning av hur du uppfattar projektet e-Me, och vad din roll i projektet är?
Vad finns det för några orsaker till att man utvecklar en integrationsplattform?	- Skulle du kunna ge en kort beskrivning av hur du uppfattar projektet e-Me, och vad din roll i projektet är?
Hur ser utvecklingen för en integrations-plattform ut?	- Kan du ge en kort beskrivning av hur SOA-miljön i e-Me ser ut? - Hur hanteras tjänsterna i e-Me? - Hur ser gränssnittet för tjänsterna ut? - Är du bekant med SOMA? - Är du bekant med SOMF? - Av de erfarenheter du har fått genom arbetet med e-Me + SOA, vilka du lyfta fram?

Tabell 2: De kvalitativa genererande intervjufrågornas koppling till studiens forskningsfrågor

Kvalitativa verifierande intervjuer

De kvalitativa verifierande intervjuerna syftar till att validera det ramverk som vi planerar ta fram under denna studie. Vi anser även här att det finns ett behov av att förbereda respondenterna innan intervjuerna, genom att i förväg dela med oss av det ramverk vi planerar ta fram. Precis som före de genererande intervjuerna kommer respondenterna att få tillgång till detta ramverk före själva intervjuerna, för att på så sätt kunna ge oss bättre och mer genomtänka synpunkter och kommentarer.

2.5.3 Benchmarking

Vi kommer som en del av vår studie att utföra en så kallad benchmarking på ett antal olika integrationsplattformar och dess API:er. Litteraturstudien samt delvis de kvalitativa genererande intervjuerna fungerar som input till denna benchmarking. Benchmarkingen genomförs för att bilda oss en uppfattning om hur det bästa API:erna ser ut, för att lära oss av de bästa inom ämnet. Betydelsen av ordet benchmarking definieras av Andersen och Pettersen (1997, s. 11) som

> Ett benchmark är en prestationsnivå som är känd som den bästa för en verksamhetsprocess (bäst-i-klassen) och som kan användas som referens vid en jämförelse.

Med hjälp av benchmarking kan man hitta inkrementella ändringar och förbättringsidéer som kan ha ett ursprung i det egna arbetet. Man hittar även orsaker till olika förbättringar utanför sin egen organisation och man kan ta fram innovativa och helt nya metoder som kan användas för att göra stora framsteg i företagets/projektets utveckling. Att kunna skapa mål som ligger på samma nivå som de bästa inom ämnet leder organisationen/projektet till att förbättra och tydliggöra sina mål, vilket är att förbättra och förändra sig så att man hamnar på en högre eller likvärdig nivå. (Andersen & Pettersen 1997, s. 13-14)

Det finns olika typer av benchmarking, som beskrivs av Andersen och Pettersen (1997, s. 16-19):

- **Konkurrentbenchmarking** vilket innebär att man gör jämförelser av egna processer eller resultat mot de bästa konkurrenterna inom samma område som tillhandahåller samma tjänster eller produkter.
- **Funktionell benchmarking** innebär att man gör jämförelser av olika processer och funktioner med företag som inte konkurerar med det egna företaget/projektet men ändå är inom samma teknikområde och bransch.
- **Processbenchmarking** innebär att man jämför allmänt bruk och metoder när man utför verksamhetsprocesser. Syftet är att själv bli bättre och lära sig av de bästa inom området.

Den typ av benchmarking som vi ämnar utföra är en så kallad processbenchmarking, då denna bäst passar in på vår studie. Vi kommer att (med litteraturstudien och de kvalitativa genererande intervjuerna som grund) ta fram ett antal kriterier för den benchmarking vi skall genomföra.

2.6 Analysmetod

Eftersom vi kommer att studera projektet e-Me så tycker vi att grundad teori är ett bra val för oss som analysmetod, då grundad teori används när forskarna är engagerade i småskaliga projekt och man använder kvalitativ data för att studera den mänskliga interaktionen eller om forskningen är inriktad på speciella miljöer eller utforskande. (Denscombe 2009, s. 125) Det man enligt Denscombe (2009, s. 126-128) bör tänka på vid användning av grundad teori är dessa 5 kriterier:

- **Teorier ska vara ”grundade” i empirisk forskning.** Det räcker inte med att göra en studie och sen bara hänga på teorin. Det är bättre att ta utgångspunkt i den empiriska studien och därefter bygga upp allmänna teorier ur dessa data.
- **Teorier ska genereras genom systematisk analys av data.** Teorier och begrepp som kommer att utvecklas av studiens empiriska data är en ständig process där man jämför befintlig data med idéer. De teorier och begrepp som träder fram ska hela tiden förbättras genom att de testas mot den data som samlats in.
- **Valet av enheter som ska ingå i undersökningen återspeglar teorins utvecklingskaraktär och går inte att förutsäga vid starten.** Inom grundad teori väljer man inte i startskedet vilka undersökningsobjekt som ingår i själva studien, då detta inte kan förutsägas vid starten. Detta leder till att man som forskare hela tiden undersöker nya infallsvinklar och nya vägar.
- **Forskare ska börja med ett öppet sinne.** När man använder sig av grundad teori är det viktigt att forskarna vågar pröva en teori och närma sig ämnet utan att ha för många idéer och tankar sedan tidigare som kan styra forskarna till att ha en för snäv bild av ämnet. Forskarna ska våga upptäcka nya saker.
- **Teorier ska vara användbara på en praktisk nivå och meningsfulla för dem ”på golvet”.** Det är viktigt att lägga stor vikt på det praktiska istället för det abstrakta när det gäller frågor om kunskap och sanning. Grundad teori utgår ifrån att

en teoris värde bara kan mätas i hur mycket nytta den kan ha vid hantering av verkliga praktiska behov.

Det finns två olika varianter av grundad teori, som tagits fram av de två skaparna: Strauss och Glaser. Strauss variant, som han har skapat tillsammans med Corbin, innehåller validering som en oerhört viktig del av den grundade teorin. Dem menar också att om en annan forskare gör om samma studie som vi, så skall dem komma fram till samma resultat. Genom att vi har validerat den benchmarking och de första intervjuer som vi genomförde, anser vi att denna typ av grundad teori stämmer väl överens med vår studie. Vi har som sagt också studerat ett småskaligt projekt genom kvalitativa intervjuer, och också detta stämmer väl överens med grundtanken för grundad teori. (Funcke 2005)

En benchmarking har även genomförts, denna har också haft en validerande effekt. Vi anser också att de kriterier som Denscombe satt upp stämmer väl överens med genomförandet av vår studie: Vi har grundat teorin i empirisk forskning, och genom en systematisk analys av den empiriska datan; vi har inte haft några förutbestämda enheter som har ingått i vår forskning, utan dessa enheter har kommit fram under studiens gång; vi började med ett öppet sinne, och den teori vi slutligen tog fram kan enligt utvecklarna användas på praktisk nivå, och var meningsfull för dem.

Enligt Goldkuhl (1998) anser Corbin och Strauss att det är viktigt med förståelseinriktad kunskap, och att tilldela innebörder till olika fenomen. Även detta har vi tagit hänsyn till i vår analys. Sammanfattningsvis anser vi alltså att vi har tillämpat grundad teori i vår analys, då alla dessa kriterier och grundidéer bakom grundad teori har tillämpats.

2.6.1 Framtagning av ramverk

Under framtagningen av vårt ramverk kommer vi att gå igenom tre faser av kodning (inom ramen för grundad teori):

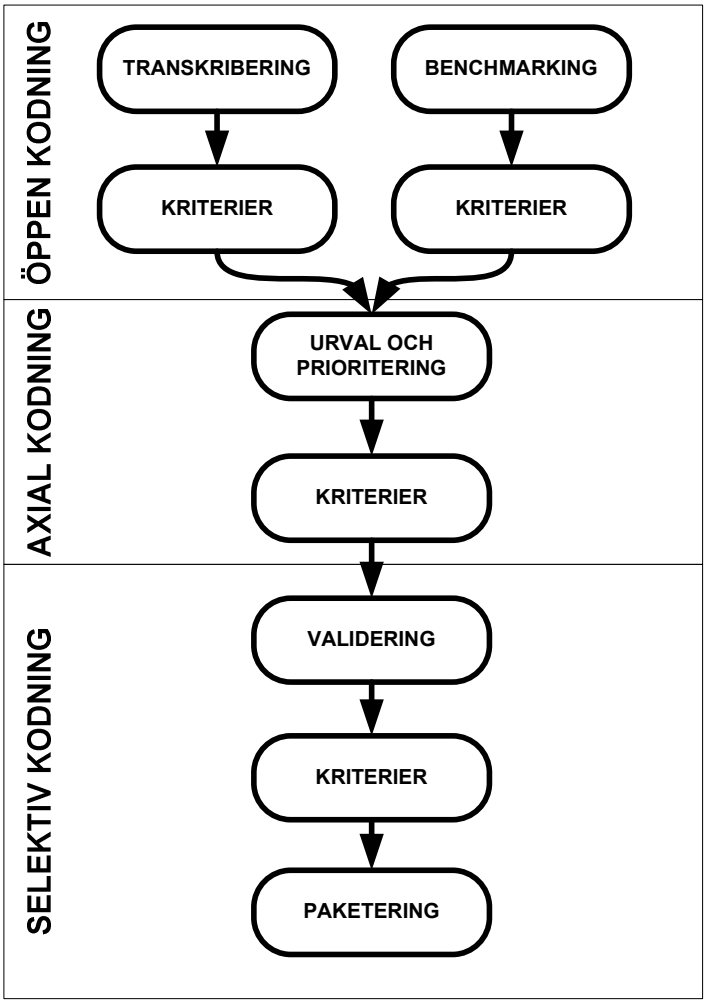
- **Öppen kodning**. I denna fas går vi från de transkriptioner vi genomfört av de kvalitativa genererande intervjuerna till en grund för själva ramverket.
- **Axial kodning**. Under denna fas kommer vi att jämföra teorin och benchmarkingen med den grund vi tagit fram, och göra justeringar av ramverket utifrån denna jämförelse.
- **Selektiv kodning**. I denna fas kommer vi att paketera ramverket, i form av en generell modell. Denna paketering kommer också att vara beroende av resultatet av våra kvalitativa verifierande intervjuer.

Genom att följa dessa tre faser kommer vår modell att vara väl grundad i teorin, benchmarkingen och respondenternas svar.

Beskrivning av tillvägagångssätt

Vi kommer i detta kapitel att diskutera det tillvägagångssätt vi har tillämpat för att ta fram ramverket. Denna beskrivning tar sin utgångspunkt i de tre olika faser inom grundad teori som vi har använt för att ta fram ramverket: Öppen kodning, axial kodning samt selektiv kodning, och går därefter igenom den arbetsprocess som vi tillämpat i varje kod-

ningsfas. Den strategi vi har analyserat det empiriska materialet utefter kan ses i *Figur 4: Studiens analysstrategi*.



Figur 4: Studiens analysstrategi

Som synes i analysstrategin består vår öppna kodning av att transkribera intervjuer, samt genomföra en benchmarking, och därigenom beskrev vi de begrepp som var centrala för framtagningen av vårt ramverk. Corbin och Strauss (2008, s. 52) beskriven den öppna kodningen som den analytiska processen, där man just skall ta fram de begrepp som är centrala för studien, samt beskriva dessa. Detta gjorde vi i form av två listor med kriterier, där vi även beskrev kriteriernas egenskaper. Även en del av den axiala kodningen genomfördes samtidigt.

Den axiala kodningen består enligt Corbin och Strauss (2008, s.198) av att forskarna sorterar och kategoriserar begreppen som framkommit under den öppna kodningen. Vi sorterade de kriterier vi kom fram till utefter prioritet, och valde även ut de kriterier vi ansåg viktigast, det vill säga de kriterier som starkt genomsyrade den öppna kodningen.

Under den sista fasen i grundad teori, selektiv kodning, förfinade vi den teori som framkommit (den första versionen av ramverket). Detta ligger också i linje med vad Corbin och Strauss säger att denna kodningstyp går ut på, nämligen att välja ut de viktigaste begreppen, samt att relatera dessa till ett kärnbegrepp. Vårt kärnbegrepp var API:er, och de andra begreppen syftade till att hjälpa till att ta fram just API:er. På detta sätt är dessa begrepp och kategorier kopplade till kärnbegreppet. Denna sista fas resulterade i den andra versionen av vårt ramverk. I denna sista kodning bidrog även den kvalitativa studien till kriterier för själva paketeringen av ramverket.

2.7 Presentationsmetod

Vår studie presenteras i form av denna rapport, som är utformad enligt de riktlinjer som finns för magisteruppsatser vid Högskolan i Borås. Vi redovisar i denna rapport hur vi har genomfört studien, samt de resultat studien genererat. Studien kommer även att presenteras muntligt vid ett seminarium, där det finns möjlighet att ställa frågor och framföra kommentarer till författarna.

2.8 Utvärderingsmetod

Det finns enligt Goldkuhl (1998) tolv generella utvärderingskriterier, som skall stödja utvärdering av kvalitativt material. Dessa kan ses som allmänna lämpliga utvärderingskriterier, och vi kommer att använda dessa kriterier för att under och efter studiens genomförande utvärdera just detta genomförande. De kriterier som främst kommer att användas under studiens genomförande, men också under eftervärderingen, är:

- **Nyfikenhet.** De personer som genomför studien måste ha ett engagemang och en vilja att verkligen studera fenomenet som studien undersöker.
- **Tydlighet.** Antaganden och värderingar som författarna har gjort måste klart och tydligt redogöras för. Det måste också tydligt visas hur studien har genomförts.
- **Ärlighet.** Det är viktigt att man inte favoriserar eller förskönar teorier eller antaganden som man själv gynnas av, utan att man är ärlig, även när det gäller resultat.
- **Öppenhet.** Det är viktigt att de som genomför studien är öppna för nya hypoteser och teorier, och inte är låsta i gamla värderingar.
- **Noggrannhet.** Studien skall genomföras på ett noggrant sätt.
- **Nyskapande.** De personer som genomför studien skall ej vara rädda för att ge sig in på outforskade områden, de skall vara kreativa i sitt tillvägagångssätt.
- **Ansvarsfullhet.** Det är viktigt att de personer som genomför studien tar ansvar för den producerade studien, då andra forskare senare kan tänkas referera till verket.
- **Rationalitet.** De beslut som tas inom studiens ramar måste vara välgrundade, och väldokumenterade.
- **Relevans.** Den kunskap som genereras skall vara relevant, det vill säga att den skall vara meningsfull inom ämnesområdet.
- **Kontext.** Det är viktigt att kunna ändra synsätt på det som studeras. Ibland måste studieobjektet ses i ett sammanhang, medan det ibland kan ses från ett fristående perspektiv.

De kriterier som i störst utsträckning används i efterv ärderingen är:

- **Tillgängliggörande.** Det är viktigt att på ett bra och lättillgängligt sätt göra studien och dess resultat tillgängliga för andra forskare.
- **Reflektion.** Det är viktigt att ha ett kritiskt och granskade förhållningssätt till den egna studien, samt andra forskares bidrag inom samma område.

För att säkra en god kvalitet på vår studie och dess presentation kommer författarna att ha dessa kriterier i åtanke, och som sagt kommer de även att användas för att utvärdera författarnas arbets- och förhållningssätt under genomförandet av studien. Denna utvärdering görs efter studiens färdigställande.

Vi kommer även att ha de uppsatskrav som gäller för kandidat- och magisteruppsatser vid Institutionen för data- och affärsvetenskap vid Högskolan i Borås i åtanke.

3 TEORI

I detta kapitel tar vi upp den teori som är relevant för studien. Denna teori är grundad på litteratur som beskriver de metoder och koncept som har tillämpats inom projektet e-Me, samt den typ av systemarkitektur som är relevant för studien. Vi tar även upp konceptet metautveckling, då det är detta vi ämnar bedriva genom framtagandet av vårt ramverk.

Get your facts first, and then you can distort them as much as you please.
\- Mark Twain

3.1 Systemutvecklingsmetoder

En systemutvecklingsmetod kan enligt Avison och Fitzgerald (2003, s. 20) definieras som

> A collection of procedures, techniques, tools and documentation aids which will help the systems developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of subphases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control, and evaluate information systems projects.

En metod är dock mer än bara en samling av de saker som nämns ovan: Den är vanligtvis baserad på något sorts filosofiskt perspektiv, som till exempel agila metoder. Vanliga mål som en systemutvecklingsmetod försöker uppfylla är ofta till exempel att dokumentera de krav som finns på systemet som skall byggas, att ta fram ett system inom en rimlig tid till en rimlig kostnad, eller att producera ett system som är väldokumenterat samt lätt att underhålla. (Avison & Fitzgerald 2003, s. 21-22) I projektet e-Me används den **agila** systemutvecklingsmetoden **Scrum**.

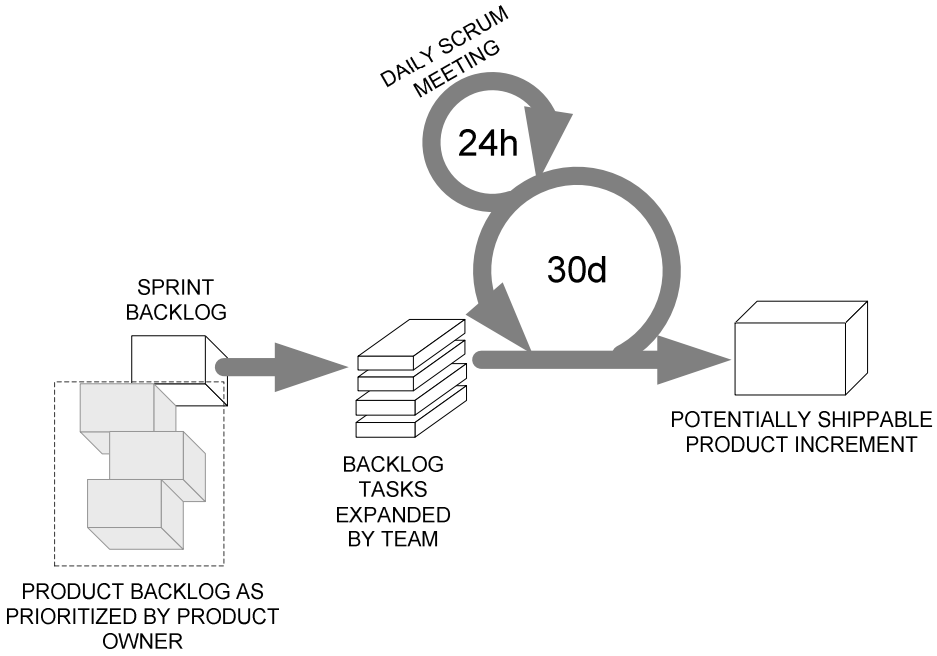
3.1.1 Agile

Agile är en systemutvecklingsansats som är byggd på tanken att olika projekt behöver olika utvecklingsprocesser och utvecklingsmetoder. Ansatsen fokuserar på färdigheter, kommunikation och utvecklingscommunityn, i stället för att fokusera på utvecklingsprocessen. Detta gör enligt Cockburn (2000) att projekt där agilt tänkande tillämpas blir mer effektiva och reagerar bättre på förändringar än de hade gjort med traditionella systemutvecklingsansatser.

Enligt Highsmith och Cockburn (2001) förutsätter många systemutvecklingsorganisationer att variationer i en process eller i ett arbetssätt beror på fel, antingen i kommunikation eller planering. Inom agile är förmågan att hantera variationer, och även att skapa dessa (genom att anpassa sig till situationen så bra som möjligt) den stora hörnstenen. Agilitet innebär att utvecklarna väljer de verktyg och de resurser som på bästa och mest effektiva sätt realiserar produkten så som beställaren vill ha den. Flexibilitet är ett nyckelord.

Scrum

Scrum är en systemutvecklingsmetod som behandlar ett systemutvecklingsprojekt som en black box: Det är inte i början helt klart vad projektet skall uppnå, och vad som skall implementeras. Scrum har en agil ansats, och metoden är därigenom flexibel i förhållande till förändringar som sker i projektet under utvecklingens gång. Utveckling i en Scrumprocess sker iterativt och inkrementellt, vilket visas i *Figur 5: Scrum Development Process (Mountain Goat Software 2008)*. En iteration i Scrum kallas för *sprint*, och varar normalt i två till fyra veckor. Scrum består av tre huvudkomponenter: Roller, artefakter och möten. (Schwaber 1997)



Figur 5: Scrum Development Process (Mountain Goat Software 2008)

Roller

Det finns i Scrum fyra olika roller: *Product Owner*, *ScrumMaster*, *Team* samt *Stakeholder*. Rollen *Product Owner* har ansvaret att presentera för alla vad projektet handlar om och vad slutprodukten ska bli. Personen har även hand om projektet och dess egenskaper och krav som ska implementeras. Det är även *Product Owner* som ansvaret ligger på om projektet lyckas eller inte. Personen som innehar rollen *ScrumMaster* ansvarar över Scrumprocessen och för att lära ut denna till alla inblandade i projektet samt se till att alla följer de regler som gäller för projektet. *Team* är en grupp på runt sju personer som utför själva analysen, designen, implementeringen och testningen i Scrumprocessen. (Schwaber 2007) En *Stakeholder* är någon som har intresse av projektet, men som inte är direkt inblandad i själva utförandet. En stakeholder observerar och ger råd till deltagarna i projektet. (Wake 2004)

Artefakter

Det finns fyra typer av artefakter i Scrum: *Product Backlog, Sprint Backlog, Sprint Goal* samt *Increment*. En *Product Backlog* är en lista med krav på vad som ska finnas med i den slutliga produkten. Personen med rollen Product Owner är ansvarig för att de mest värdefulla kraven implementeras först och kan byggas vidare på. Detta görs genom att frekvent prioritera de krav som finns i Product Backlog på ett sådant sätt att de mest nödvändiga kraven ligger först inför nästa iteration. *Sprint Backlog* används för att beskriva arbetet eller de uppgifter som laget måste tar fram för att göra om den Product Backlog som tagits fram för att användas i den här sprinten till funktionalitet som skall kunna fungera som en del av den slutgiltiga produkten. *Sprint Goal* är de mål som laget ska uppnå i en sprint. Det färdiga resultatet skapar en Increment vilket betyder en fungerande produkt som har de egenskaper som den senaste sprintens backlog genererat. (Schwaber 2007)

Möten

Det finns fyra typer av möten i Scrum. *Sprint Planning Meeting, Daily Scrum, Sprint Review Meeting* samt *Sprint Retrospective.* På ett Sprint Planning Meeting bestäms det vilka krav som ska finnas med i Product Backlog till nästa iteration. På dessa möten presenterar Product Owner de nya saker som den vill ha implementerade i nästkommande sprint. Därefter kommer laget sen överens om de kommer att hinna med att implementera dessa krav under den tiden. (Schwaber 2007)

Varje dag håller laget ett 15 minuters möte kallat Daily Scrum och det går ut på att varje lagmedlem ska svara på tre frågor: *Vad har du gjort sedan senaste Daily Scrum mötet?*, *Vad planerar du att göra tills nästa Daily Scrum möte?* samt *Finns det några svårigheter som hindrar dig att bli klar med det du ska göra den i den här sprinten?* Syftet med det här mötet är att organisera och synkronisera arbetet för alla medlemmar i gruppen dagligen så att arbetet skall kunna fungera på ett bra sätt och undersöka om det behövs sättats in några extra möten för att uppnå målet. (Schwaber 2007)

I slutet av varje sprint hålls ett sprint review meeting vilket är ett möte som är ungefär fyra timmar långt där gruppen diskuterar vad som har utvecklats under sprinten och framför detta till Project Owner och andra som vill deltaga. Detta är ett informellt möte där det främst gäller att ta reda på vad som ska göras härnäst baserat på vad som hittills har tagits fram. Efter ett sprint review meeting och innan nästa Sprint planning meeting håller ScrumMaster ett Sprint retrospective meeting där ScrumMaster vill att laget ska utvärdera processen och få förslag på vad som varit bra och vad som kan göras annorlunda i nästa sprint. (Schwaber 2007)

Test-Driven Development

Test-Driven Development (TDD) är en agil systemutvecklingsmetod som innebär att utvecklingen av en applikation tar sin utgångspunkt i tester. Enligt Ron Jeffries (se Beck 2003, s. 9) är målet med TDD att producera

> clean code that works.

Enligt Beck (2003, s. 9) är detta mål värdefullt, då det är ett förutsägbart sätt att utveckla applikationer. När utvecklaren börjar med att skriva testet, och därefter fortsätter med att skriva koden, vet utvecklaren precis när koden är klar: När den går igenom testet. Detta leder också till att utvecklaren slipper ett långt spår av buggar att följa tillbaka genom koden.

TDD fungerar också som en sorts lärare för utvecklaren: Koden lär utvecklaren hur den är mest effektiv, och vilken kod som är bäst. Enligt Beck (2003, s. 9) gör det också livet lättare för användaren av applikationen, då den fungerar på ett mer tillfredsställande sätt. Det leder också till beroenden mellan utvecklarna: Utvecklarna räknar med att koden som de andra skriver fungerar, och om den passerar testerna, så fungerar den. TDD är beroende av att utvecklarna skriver sina egna tester, och gör detta före de skriver koden som skall testas. Det är också oerhört viktigt att vara agil, då testerna ger feedback på kod och funktioner som kan behöva ändras snarast.

3.2 Co-design

Co-design är en filosofi som säger att alla intressenter skall vara med i framtagandet av en ny produkt, och därigenom ta fram en produkt som passar alla intressenternas behov. Företag och verksamheter försöker konstant att fånga kunskap om ideala situationer för sina kunder och klienter, vilken de sedan försöker mappa mot de resurser och tillgångar som finns inom verksamheten, eller som de kan skapa som ett resultat av denna kunskap. (Lindell, Lind & Forsgren 2006)

Kunder och klienter försöker å andra sidan konstant att fantisera om och tänka på hur en ideal situation skulle se ut för dem, och vilka resurser de kan använda för att nå dessa ideala situationer. Ur detta perspektiv kan forskare hjälpa både företag och kunder att nå sina behov, och upptäcka den kunskap om de ideala situationerna som saknas. Denna process kallas för den kunskapsskapande processen inom co-design. (Lindell, Lind & Forsgren 2006) Genom hela denna process krävs det ett ständigt kommunikationsflöde, för att säkerställa att alla intressenters ideala situationer beaktas. (Lind, Albinsson, Forsgren & Hedman 2007)

Det är viktigt att ha någon som leder arbetet med co-design, en så kallad *Maestro*. Maestrons uppgift är att bestämma vilka perspektiv som är är nödvändiga för arbetet, samt att ta in dessa perspektiv i själva arbetet med co-design. Det är också maestrons ansvar att hitta ett designspråk, som gör att alla intressenter kan kommunicera sina önskemål och idéer på ett för alla förståeligt och effektivt sätt. (Albinsson, Forsgren & Lind 2008)

För att ta tillvara på intressenternas önskemål och idéer kan flera tekniker användas, till exempel workshops (där de får berätta om vad de egentligen vill ha) eller prototyper (ofta utifrån det som kommit fram under en workshop) som de direkt kan ge feedback på. (Oosterholt, Kusano & de Vries 1996)

3.3 Systemarkitekturstrategier

En systemarkitekturstrategi beskriver hur en verksamhet planerar och strukturerar sina informationssystem, samt kommunikationen mellan dessa. Detta är viktigt för att få en

god datakvalitet och ha kontroll över de olika verksamhetsdelarnas autonoma informationssystem. Dessa strategier kan vara mer eller mindre detaljerade. Exempel på sådana strategier är IRM, VBS, PAKS och SOA. (Eriksson, Goldkuhl, Axelsson & Hultgren 1997) Nedan presenteras dessa fyra strategier: IRM, VBS och PAKS relativt överskådligt, och sedan dyker vi in djupare på SOA då denna strategi har störst betydelse för vår studie.

3.3.1 IRM

Information Resource Management (IRM) är en datadriven ansats. Inom många områden ökar ständigt kraven på samverkan mellan informationssystem både inom verksamheter och mellan verksamheter. Inom IRM-strategin löser man detta genom att se informationen som en resurs på samma sätt som maskiner och arbetskraft. När man jämställer informationen på det här sättet innebär det att man måste styra och administrera resursen på ett bra sätt, liknande det sätt man styr andra resurser i verksamheten. Det betyder till exempel att informationen skall kunna

- planeras med hjälp av datamodellering
- anskaffas, bara en gång och då vid källan
- lagras, och på ett sådant sätt att alla kan nå informationen vid behov

Detta är grunden för den datadrivna ansats som kallas för IRM. (Axelsson & Goldkuhl 1998, s. 35) I IRM-strategin utbyts data genom registersamverkan, och möjliggörs av relationsdatabaser. Syftet med IRM är att undvika redundant lagring av information, och att undvika att information arkiveras längre än nödvändigt, och detta sker alltså genom registersamverkan, i stället för meddelandesamverkan. (Axelsson 2007)

3.3.2 VBS

Verksamhetsbaserad systemstrukturering (VBS) innebär att man använder sig av decentraliserat ansvarstagande, detta för att avgränsa verksamhetsbaserade informationssystem. Inom VBS är denna samverkan mellan informationssystem definierad på förhand. VBS, är som namnet antyder, helt och hållet verksamhetsbaserad. Verksamhetsfunktioner är inte alltid uppdelade som de organisatoriska gränserna visar, och kan delas mellan olika avdelningar. Uppdelningen av verksamhetsfunktioner grundas istället på uppdelningen av ansvar inom verksamheten. De olika informationssystemen kan, där det finns behov, kommunicera med varandra. (Axelsson & Goldkuhl, s. 45)

Inom VBS menar man att gemensamma resurser alltid skapar beroendeförhållanden, och att ansvarsgränserna som en effekt av detta blir otydliga. Handlingsfriheten för en verksamhetsfunktion anses öka om funktionen själv förfogar över samtliga sina resurser. VBS förespråkare anser därför att det är av stor betydelse att avgränsningen av verksamhetsfunktionernas autonoma informationssystem matchar de ansvarsgränser som finns i verksamheten. (Axelsson & Goldkuhl 1998, s. 45)

VBS bygger till stor del på meddelandesamverkan. Denna samverkan är den kommunikation som sker mellan de olika verksamhetsfunktionernas autonoma informationssystem. Detta kallas även för transaktionssamverkan. Meddelandesamverkan innebär att informa-

tionssystemen utbyter meddelanden direkt med varandra, i ett fördefinierat och väl beskrivet format. Detta kan vara till exempel vanlig text, eller filer. (Axelsson & Goldkuhl 1998, s. 47)

3.3.3 PAKS

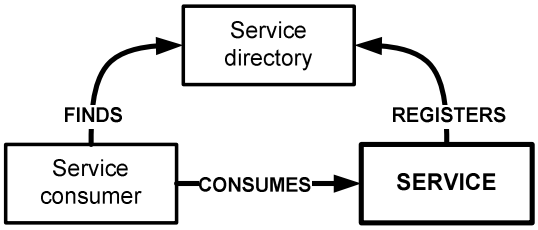
Process-, Aktivitets- och Komponentbaserad Systemstrukturering förkortas PAKS och är en systemarkitekturstrategi. Vid utvecklingen av PAKS försökte skaparna av strategin ta det bästa från IRM och VBS, och kombinera detta till en ny strategi. PAKS är en så kallad syntes av tesen IRM och antitesen VBS, och kan ses som en kompromiss mellan IRM och VBS. PAKS överskrider dock också dessa ursprungliga strategier, och bygger även på med mer innehåll. PAKS bygger på tre hörnpelare: Process, aktivitet, och komponent. Processen är själva affärsprocesstänkandet, aktiviteten är det kommunikativa handlandet mellan olika processer, och komponenten är den objektorienterade delen av strategin. (Axelsson & Goldkuhl 1998, s. 181-182)

PAKS försöker ta tillvara på de fördelar som finns med IRM och VBS, och försöker undvika dessa strategiers nackdelar. Systemstruktureringen bör enligt denna strategi (precis som i VBS) ta sin utgångspunkt i de processer (funktioner) som finns inom verksamheten. Med PAKS försöker skaparna eftersträva en god helhetslösning gällande informationsbehandling, och anlägger ett processägareperspektiv när verksamhetsprocesserna utformas. PAKS syftar också till att underlätta förändringar inom verksamheten. (Axelsson & Goldkuhl 1998, s. 194-201)

3.3.4 SOA

I en ideal värld är enligt Erl (2005, s. 3) resurser separerade och konsekventa, och tjänsteorienterade. När detta tänkande appliceras på systemarkitekturer, skapar det en universell modell i vilken automationslogik och även affärslogik blir tjänsteorienterade. Denna modell kan appliceras både på en uppgift, en lösning, en verksamhet, en community och även bortom dessa. Detta tjänsteorienteringsideal har positionerat SOA (den tjänsteorienterade arkitekturstrategin) som den nästa fasen i evolutionen av distribuerade informationssystem. Webbtjänstbaserad SOA ersätter traditionellt distribuerade arkitekturer på en global nivå.

SOA betyder oftast att ett stort, relativt komplext problem, delas upp i mindre problem, som löses separat med hjälp av väl avgränsade tjänster. Varje tjänst representerar en specifik del av ett problem. Det som separerar SOA från andra ansatser som delar upp ett problem i mindre delar är det sätt som SOA gör detta på. Problem delas upp på tjänster, som sedan kan återanvändas på ett standardiserat sätt för att återigen lösa samma typ av problem. (Erl 2005, s. 32) I *Figur 6: SOA Overview (O'Reilly Media 2005)* visas den mest traditionella bilden av SOA.



Figur 6: SOA Overview (O'Reilly Media 2005)

Som synes i figuren ovan skapar någon en tjänst, registrerar den i en tjänstekatalog, där en konsument sedan hittar tjänsten, och konsumerar den. (O'Reilly Media 2005) För att bibehålla sitt oberoende innehåller tjänsterna sin egen logik, och fungerar som en black box. När en konsument anropar en tjänst vet de ej vad som händer bakom själva anropet: De vet endast hur anropet skall se ut, och vad de kan vänta sig för något svar. Detta är den mest traditionella användningen av SOA, även om den inte innehåller allt som SOA står för. En tjänst är som sagt oftast specifik för ett visst sammanhang: Till exempel en verksamhetsprocess eller –entitet. (Erl 2005, s. 33)

Principer för en SOA

Det finns några huvudsakliga principer för en SOA. Dessa presenteras kortfattat nedan.

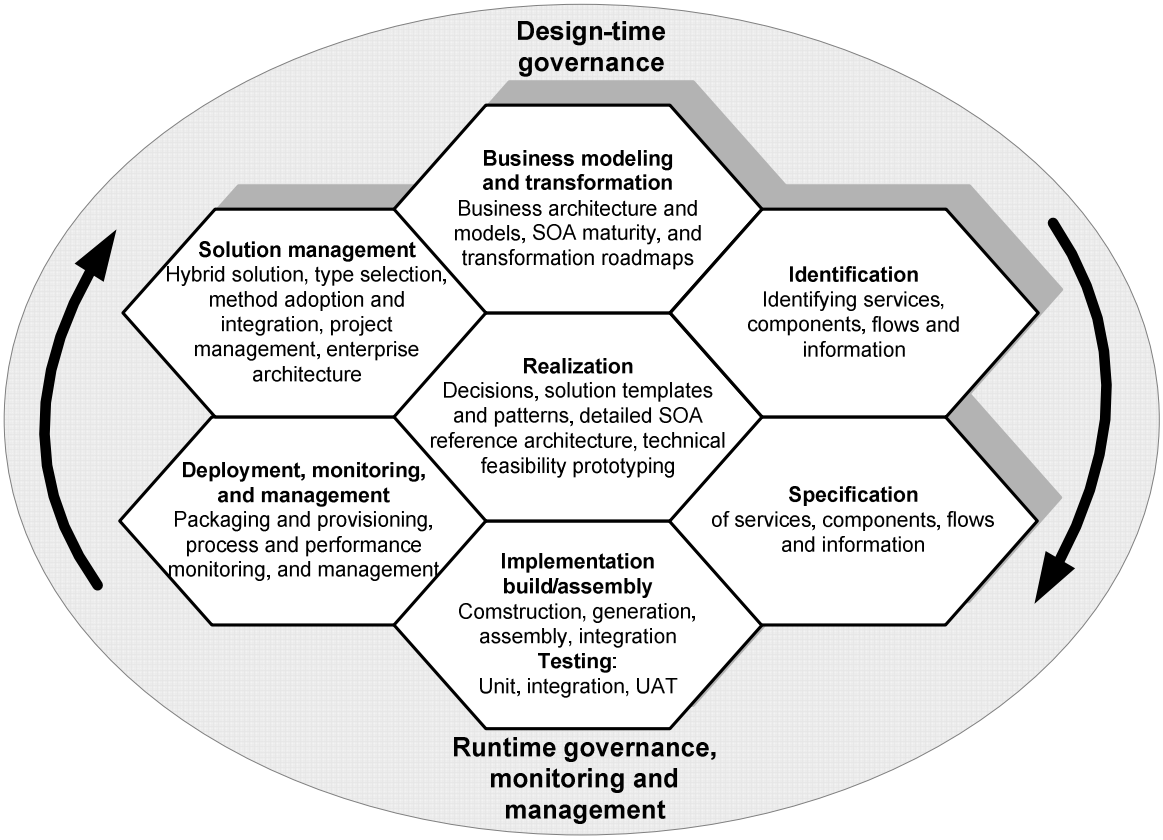
- *Loose coupling*. Det är viktigt att tjänster har så lösa kopplingar till varandra som möjligt, för att minimera beroenden och att de har så lite kännedom som möjligt.
- *Service contract*. Tjänster bör ha ett kontrakt: Konsumenten skall få veta hur tjänsten anropas, och vad den kan vänta sig för svar. Detta kontrakt bör vara samma för alla konsumenter till en tjänst.
- *Autonomy*. Tjänster bör ha kontroll över den logik som de kapslar in. Tjänster skall vara black boxes: Konsumenten skall ej veta vad som pågår bakom gränssnittet mot tjänsten.
- *Abstraction*. Denna punkt hör ihop med den föregående: Konsumenten skall ej veta vad som händer bakom tjänstens gränssnitt. Detta bör dock beskrivas i tjänstekontraktet.
- *Reusability*. Tjänster bör inte vara för komplexa: Komplex logik bör delas upp i mindre delar, för att främja återanvändning av tjänsterna.
- *Composability*. Tjänster bör kunna kombineras för att göra nya tjänster.
- *Statelessness*. Tjänster bör minimera tillstånd där information för en specifik aktivitet sparas.
- *Discoverability*. Tjänster skall vara utformade på ett sådant sätt att dem lätt kan upptäckas genom tillgängliga upptäcktsmekanismer.

Med dessa principer i åtanke har verksamheter och utvecklare av SOA en god grund för att strukturera och implementera en SOA. Det enda som saknas är en implementationsplattform för att sy ihop alla dessa principer och delar. Enligt Erl (2005, s. 37) är *Web services* (webbtjänster) teknologin som erbjuder en sådan plattform. För att modellera och

designa SOA, finns det främst två verktyg: Metoden SOMA (*Service-Oriented Modeling and Analysis*) och ramverket SOMF (*Service-Oriented Modeling Framework*).

SOMA

SOMA (Service Oriented Modeling and Analysis) är en livscykelmetod skapad av IBM för att utveckla SOA-baserade mjukvaruprojekt eller andra typer av tjänsteorienterade lösningar. SOMA definierar några viktiga tekniker och roller som ingår i ett SOA-projekt. För att åstadkomma detta har man tagit fram ett sätt att bryta ner arbetsstrukturen i mindre delar. Detta kallas för WBS (Work Breakdown Structure) och innefattar uppgifter, input- och outputarbete för uppgifter och en guide för detaljerad analys, design, implementation och färdigställande av tjänster, komponenter och flöden som behövs för att bygga en robust och stabil och återanvändbar SOA. (Arsanjani et al 2008, s. 379-381) I SOMA ingår sju grundläggande faser som framgår av *Figur 7: SOMA Phases*.



Figur 7: SOMA Phases

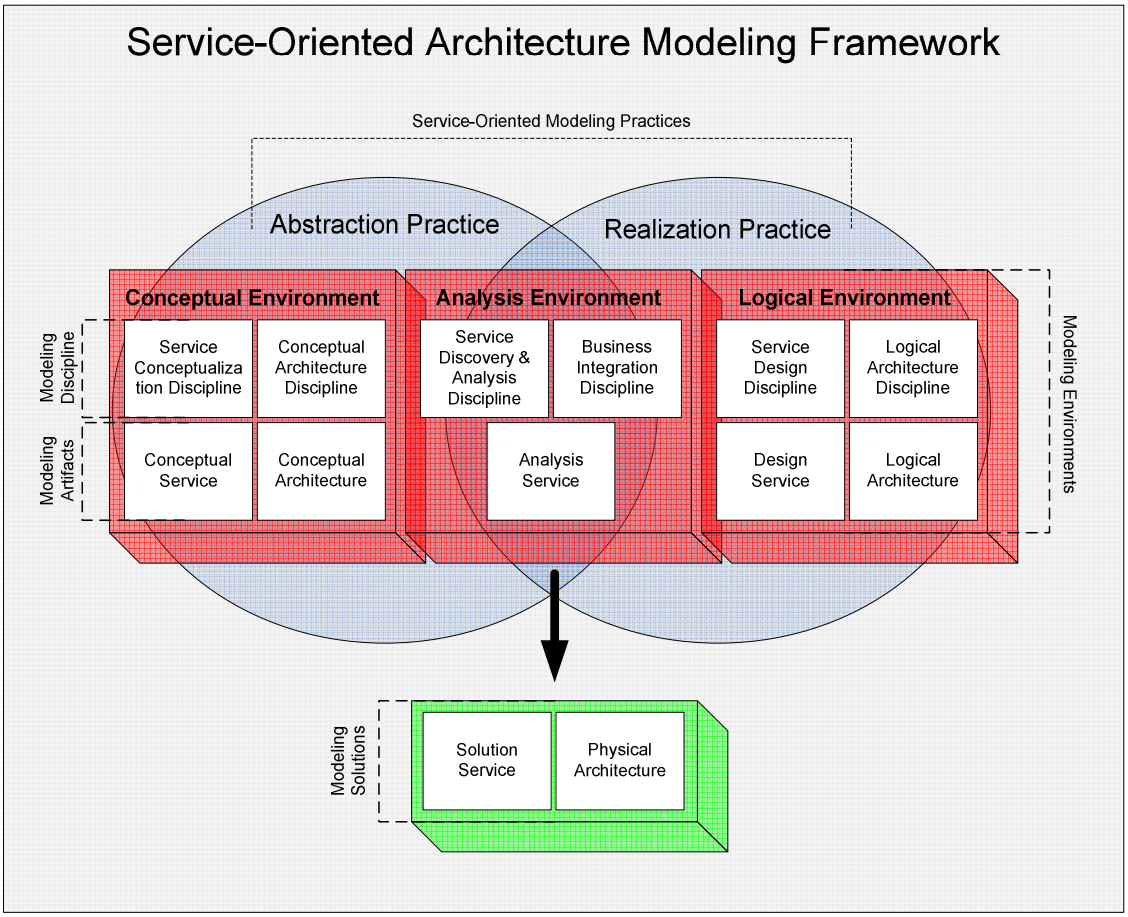
Dessa faser innehåller egenskaper som kan förändras så de passar in i de olika sekvenser de ska användas i under ett projekt. SOMA-metoden använder sig av metodkomponenter rekursivt genom små iterationer där fokuset ligger på att rätta till tekniska risker och leverera system som verksamheten kan uppskatta. Till exempel finns realisation med i alla iterationsfaser. Modellen bygger på att det inte finns någon färdigplanerad väg att följa utan alla faser kan sammankopplas med varandra. (Arsanjani et al 2008, s. 381)

I fasen *Business Modeling And Transformation* så modelleras, simuleras och optimeras verksamheten. Denna fas är inte alltid nödvändig men rekommenderas starkt. I fasen *Solution Management* skapas en extern metodmall kallad lösningsmall för projektet. Denna mall innehåller uppgifter, roller och hjälp specifikt anpassat för varje del i den slutgiltiga lösningen. Fasen *Identification Phase* används för att identifiera de tre grunderna som SOA är byggt på: Tjänster, komponenter och flöden. Det som rekommenderas är att börja med att passa in tjänster med verksamhetsmål för att underlätta fortsatt verksamhetsmodellering och tillgångsanalys.

I fasen *Specification Phase* designar man SOA. Både högnivådesign och viktiga delar av detaljdesignen skapas i denna fas. I fasen ska även de tjänster man har, vidareutvecklas för att hjälpa till att nå implementationsmålen. I *Realization Phase* fasen valideras de val som har gjorts gällande realisering och möjligheten att utföra del olika delarna i projektet rent tekniskt. I faserna *Implementation* och *Deployment, Monitoring And Management* implementeras och konstrueras tjänsterna och deras funktion. Även tekniska komponenter och flöden tas fram i denna fas. (Arsanjani et al 2008, s. 383-394)

SOMF

SOMF (*Service Oriented Modeling Framework*) är ett ramverk framtaget av författaren Michael Bell. Det är utvecklat som ett tjänsteorienterat modelleringsspråk för systemutveckling. Det är till för att ge taktiska och strategiska lösningar på verksamhetsproblem. SOMF följer en tjänsteorienterad livscykel. Den bidrar till att projekt ska få en lyckad tjänsteorienterad styrning och modellering vilket framgår av figur *Figur 8: Service-Oriented Architecture Modeling Framework*. (Wikipedia 2009)



Figur 8: Service-Oriented Architecture Modeling Framework

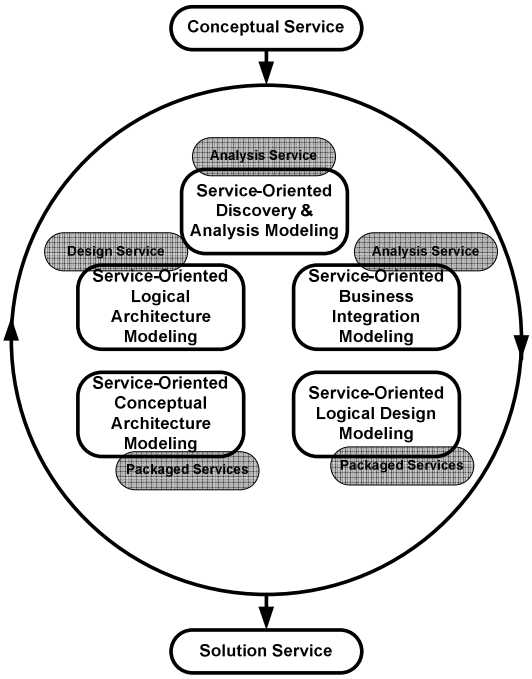
Som synes i figuren så fokuserar den på vad som är viktigt att göra i tjänsteorienterad utveckling. Detta är grundpelare för modellering som kommer att underlätta och få de inblandade att skapa en effektiv projektplan och identifiera milstolpar, detta gäller för både små och stora verksamheter. (Wikipedia 2009)

SOMF innehåller fem övergripande livscykelaktiviteter som driver en utveckling av en tjänst genom design- och driftbeslut. Vid designbeslut används den konceptuella modellen över tjänsten som input (conceptual service), därefter analyseras tjänsten sett ur ett tjänsteorienterat perspektiv (analysis service). I nästa fas skapas ett kontrakt och en logisk modell över tjänsten (design service), och därefter skapas tjänsten konkret, och blir en produkt (solution service). Följande fem livscykelaktiviteter definieras i SOMF:

- **Service-oriented discovery & analysis modeling**. Upptäck och analysera tjänster för kornighet, återanvändning, interoperabilitet, loose coupling och konsolideringsmöjligheter.
- **Service-oriented business integration modeling**. Identifiera tjänsteintegration och positioneringsmöjligheter med affärsprocesserna i verksamheten.

- **Service-oriented logical design modeling**. Etablera relationer mellan tjänster och definiera kommunikationsvägarna och meddelandeformerna för dessa tjänster.
- **Service-oriented conceptual architecture modeling**. Etablera en riktning för verksamhetens tjänsteorienterade arkitektur, och beskriv en teknisk miljö för denna. Identifera också vem som är ägare till verksamheten.
- **Service-oriented logical architecture modeling**. Integrera mjukvarutillgångar, och etablera beroenden för den tjänsteorienterade arkitekturens logiska miljö. Ta vara på återanvänding, loose coupling och konsolidering.

Dessa livscykelaktiviteter, deras relationer och deras kronologiska ordning beskrivs i *Figur 9: SOMF Life Cycle Activities*. (Wikipedia 2009)



Figur 9: SOMF Life Cycle Activities

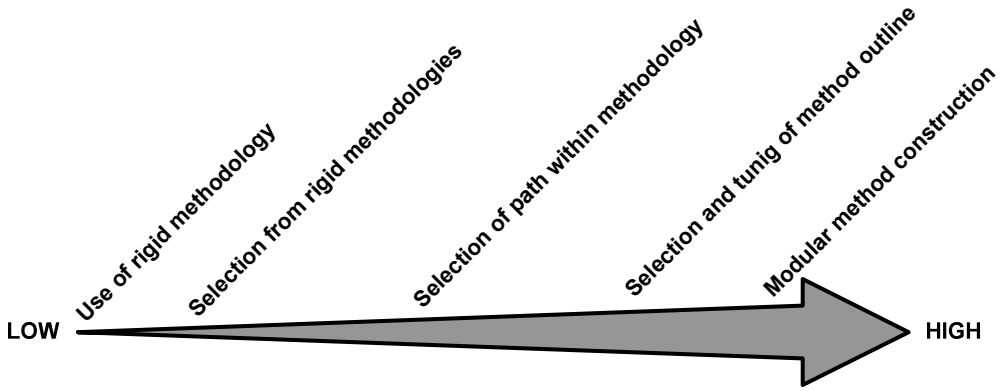
Som synes i figuren är livscykelprocessen iterativ, och tar som en input en konceptuell modell över en tjänst, och levererar slutligen en lösning för tjänsten.

3.4 Metautveckling

Om man har flexibilitet utan kontroll kan det knappast kallas för en metodik, eftersom det inte finns det minsta systematiska eller koordinerade upplägg för att framställa några metoder i det här sättet att jobba. För att en sådan process ska bli mer systematiskt hanterbar måste man använda sig av metautveckling. Med hjälp av metautveckling skapar man metodik och metoder. För ett informationssystem är en metodik en mängd metoder framtagna för att utveckla automatiserade system. (Brinkkemper, Lyytinen & Welke 1996, s. 2)

En metod i det här fallet definierar de steg man måste ta för att automatisera ett system tillsammans med de verktyg och tekniker som behövs för att utveckla ett system. Om man anpassar en metodik för att fylla behoven för ett visst projekt kallas det ibland för situationsmetautveckling. För ett informationssystem betyder det att en situationsmetautveckling både designar, konstruerar och använder sig av olika informationssystemsmetoder. (Brinkkemper, Lyytinen & Welke 1996, s. 2)

Om man använder sig av en för rigid metodik innebär det väldigt lite flexibilitet. Sådana metodiker är byggda på bara en typ av utvecklingsfilosofier och använder sig av fixerade standarder, principer och tekniker. Det är oftast inte tillåtet för Projektledaren att ändra i metodiken. Man kan istället välja att för varje projekt ta och använda olika rigida metodiker för att hitta en som bättre passar in på det specifika projektet. (Brinkkemper, Lyytinen & Welke 1996, s. 2) Detta visas i *Figur 10: Graden av flexibilitet i metautveckling för ett informationssystem*.



Figur 10: Graden av flexibilitet i metautveckling för ett informationssystem

Detta kan vara som att köpa grisen i säcken: Du kanske kan göra det bästa av situationen men metodiken kanske ändå inte passar in fullt ut och andra metodiker kan ha fördelar som du går miste om. Det är även så att för varje ny metodik som används tar det tid för alla inblandade att sätta sig in i och lära sig den för att den ska kunna användas på ett bra sätt. (Brinkkemper, Lyytinen & Welke 1996, s. 2)

Inom många metodiker skapas mer flexibilitet genom att låta användaren välja olika fördefinierade vägar. Typiska vägar är traditionellt och snabb systemutveckling. En del metodiker innehåller nu även vägval som hanterar utvecklingsaspekter så som objektorientering. Nackdelen med dessa metodiker är att det inte alltid går att kombinera alla typer av vägval. Man kan även välja och finjustera en metod. Denna typ bidrar till att varje projekt kan välja metoder från olika ansatser och finjustera dem för att passa så bra in på projektet som möjligt. (Brinkkemper, Lyytinen & Welke 1996, s. 2)

Modulmetodtuveckling är ett av de mest flexibla sätten att att generera en metodik på. Varje byggsten är ett metodfragment sparat i en basmetod. Genom att använda olika regler kan dessa byggstenar byggas ihop baserat på hur ett projekts profil ser ut. Resultatet är

en väldigt effektiv, konsistent och färdig metodik för projektet. (Brinkkemper, Lyytinen & Welke 1996, s. 3)

3.5 Sammanfattning

En **systemutvecklingsmetod** innehåller procedurer, tekniker, verktyg och dokumentationshjälpmedel, och är oftast baserad på någon sorts filosofiskt perspektiv. I projektet e-Me används den agila systemutvecklingsmetoden Scrum. **Agile** är en systemutvecklingsansats som är byggd på tanken att olika projekt behöver olika processer och metoder.

Scrum är en metod som tar till sig denna ansats, och behandlar ett systemutvecklingsprojekt som en black box: Det är i början av projektet oklart vad projektet skall uppnå, samt vad som skall implementeras. Denna metod är med andra ord väldigt flexibel. En annan agil utvecklingsmetod är **Test-Driven Development**, som innebär att utvecklingen av en applikation börjar genom att skriva tester.

Co-design är en metod för att ta reda på vad som egentligen skall byggas, och hur det skall byggas för att uppnå de mål och krav som har tagits fram. Målen och kraven tas fram av alla intressenter i projektet tillsammans, under ledningen av en så kallad Maestro. Co-design har tillämpats, och tillämpas, inom projektet e-Me.

e-Me återanvänder sig av tjänster, och kan därför ses som en del av en tjänsteorienterad arkitektur, en så kallad SOA. SOA är en **systemarkitekturstrategi**. En systemarkitekturstrategi beskriver hur en verksamhet planerar och strukturerar sina informationssystem, samt kommunikationen mellan dessa. Andra exempel på sådana strategier är IRM (Information Resource Management), VBS (Verksamhetsbaserad systemstrukturering) och PAKS (Process-, aktivitets- och komponponentbaserad systemstrukturering).

SOA bygger som sagt på tjänster, och återanvändning av dessa. I en ideal SOA-värld är alla resurser separerade och konsekventa samt tjänsteorienterade. Det finns några huvudsakliga principer för en SOA, till exempel loose coupling, reusability, composability och discoverability. För att utforma, designa och implementera en SOA finns det en relativt välanvänd metod, SOMA, samt ett relativt ickebeprövat ramverk, SOMF.

SOMA (Service-Oriented Modeling and Analysis) är en livscykelmetod skapad av IBM, och beskriver hur man skapar SOA-baserade lösningar. SOMA definierar några viktiga roller och tekniker som ingår i ett SOA-projekt. **SOMF** (Service-Oriented Modeling Framework) är ett ramverk som är utvecklat som ett tjänsteorienterat modelleringsspråk för systemutveckling, och följer en tjänsteorienterad livscykel.

Metautveckling är en metodik för att skapa metodik och metoder. För ett informationssystem är en metodik en mängd metoder framtagna för att utveckla automatiserade system.

4 E-ME

I detta kapitel beskrivs integrationsplattformen e-Me, utifrån den kvalitativa genererande intervjustudie vi har genomfört. Detta kapitel mynnar ut i kriterier för genereringen av vårt ramverk.

The computer was born to solve problems that did not exist before.
- Bill Gates

---

e-Me är som sagt ett verktyg som är tänkt att hjälpa studenter med deras vardagliga IT-sysslor. Vi har, för att få en bild över e-Me som integrationsplattform, genomfört intervjuer med systemutvecklarna på InnovationLab, som utvecklar e-Me. För de intervjufrågor vi ställde till utvecklarna, se *Bilaga 1: Intervjuguide 1*. Intervjuerna genomfördes på respondenternas arbetsplats, med hjälp av inspelningsutrustning. Samtliga respondenter godkände att denna användes, och ingen av respondenterna ville vara anonym. Vi har ändå valt att anonymisera respondenternas åsikter och uppfattningar, därför benämns respondenterna A, B och C. Denna anonymisering har vi valt som en konsekvens av vissa av de åsikter och uppfattningar som framfördes under intervjuerna.

4.1 Respondenter

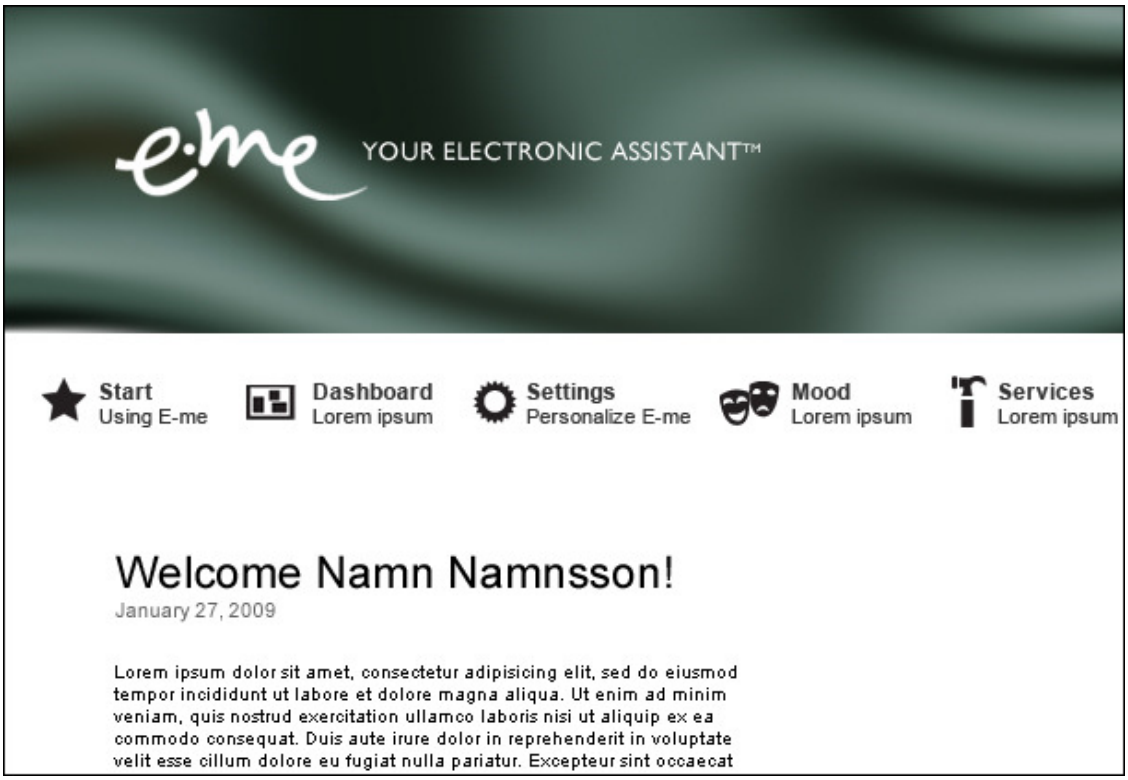
Respondenterna är Göran Golcher, Linus Andrén och Daniel Rudmark. Göran Golcher jobbar som systemutvecklare på InnovationLab, och har arbetat med systemutveckling professionellt sedan 1991. Han har arbetat med SOA i cirka fem år. Linus Andrén är systemutvecklare på InnovationLab, och har jobbat där i tre år. Daniel Rudmark har jobbat med systemutveckling sedan 1999, nu som systemutvecklare och professionsdoktorand på InnovationLab.

Respondenterna har lite blandade åsikter om just SOA: Vissa tycker att det är ett ”buzz word” och en hype, andra ser det som ett sätt att möjliggöra integration på ett enkelt sätt, medan andra anser att det mest är till för stora organisationer. Det finns också en åsikt om att det kan gå väldigt fel när man använder SOA och tjänster: Om en tjänsteleverantör ändrar sin webbtjänst utan att notifiera konsumenterna av den kan det leda till stora konsekvenser i konsumentens applikation.

4.2 Projektet e-Me

e-Me handlar om att integrera flera olika system och tjänster, och därigenom förenkla studenters vardag. Dessa tjänster kan till exempel vara scheman och studentrabatter. e-Me skall också kunna fungera som en plattform för andra typer av applikationer, som inte är knutna till just studenter. Ett exempel kan till exempel vara för beslutsstöd för beslutsfattare, där e-Me som plattform samlar in data som behövs för ett beslut, och sedan presenterar detta som beslutsunderlag för beslutsfattaren.

Det nuvarande projektet syftar till att ta fram e-Me 2.0, det har alltså funnits en tidigare version, som fungerade som en slags prototyp. Av våra respondenter är det endast Linus som har arbetat med den tidigare versionen. Till en början diskuterades att göra e-Me som en sorts mash-upapplikation, som själv samlar in information från andra informationssystem och applikationer och presenterade detta för användaren. Det beslutades dock att istället göra e-Me till en integrationsplattform, som möjliggör för externa utvecklare att skriva plugins till plattformen, genom att implementera ett API. I *Figur 11: Grafisk skiss av e-Me 2.0* visas e-Me som applikationen ser ut enligt ett av de första designförslagen.



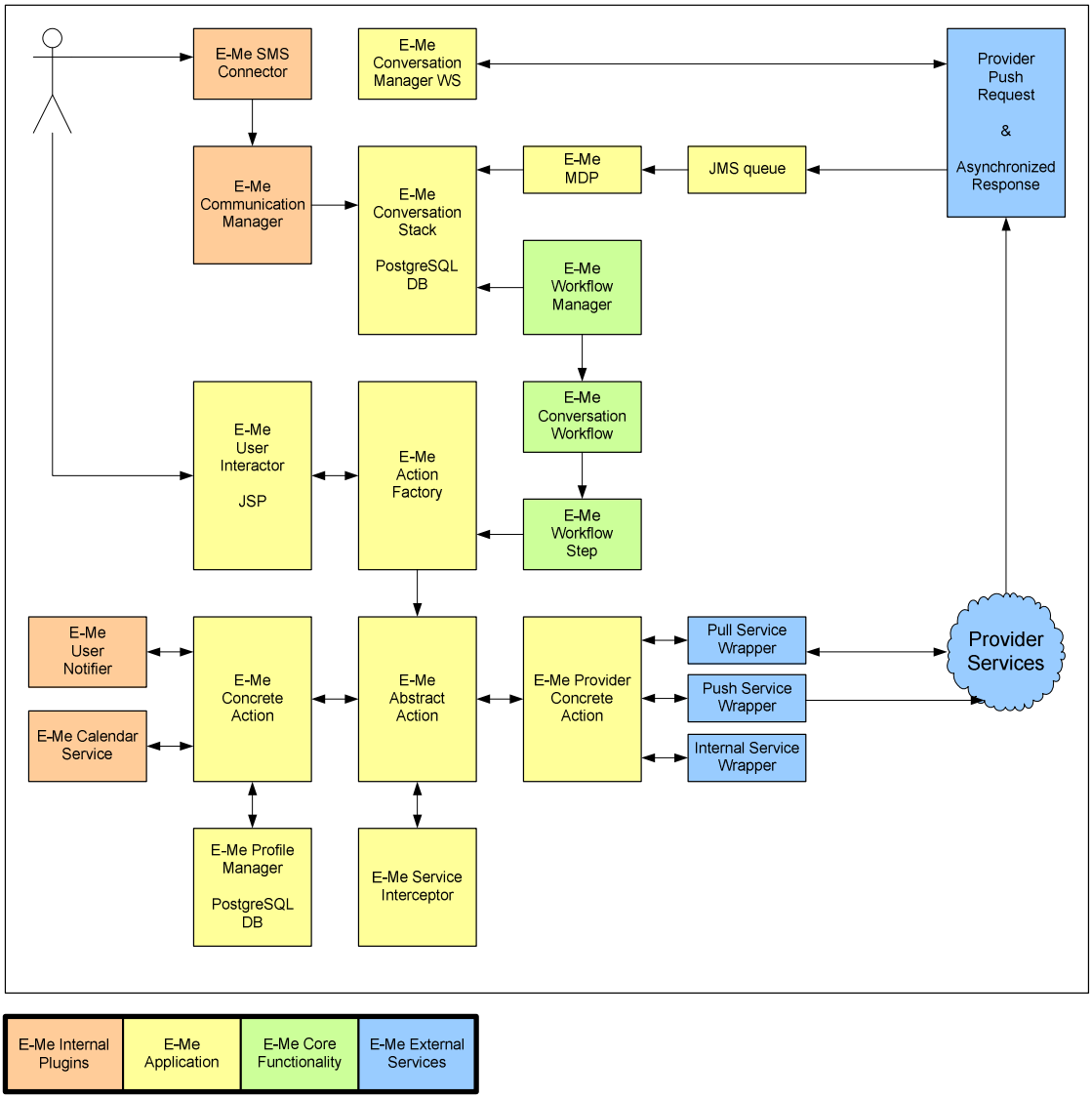
Figur 11: Grafisk skiss av e-Me 2.0

4.3 Integrationsplattformen e-Me

Det blev beslutat att e-Me skulle vara en integrationsplattform under en workshop i Stockholm, där respondent A träffade Lars Albinsson, en av upphovsmännen till e-Me. Det diskuterades först att göra en sorts mashuplösning, där e-Me själv läste aggregerade data och visade för användaren. Sedan kom man dock in på spåret med en sorts pluginarkitektur, där externa leverantörer själva kan bygga tjänster för e-Me genom att implementera ett specifikt API. I *Figur 12: Översiktlig arkitektur e-Me 2.0* visas en översiktlig bild av den tänkta arkitekturen för e-Mes integrationsplattform.

Denna grundläggande modellering har varit någon sorts ledstjärna för det fortsatta arbetet med plattformen. Respondenterna har också försökt identifiera den typ av tjänster som de och e-Mes styrgrupp förväntar sig skall finnas i e-Me. De anser också att plattformen

måste evolvera fram: Det går inte att i början spika exakt hur den skall se ut. Det gjordes i början av projektet också en riskanalys, vilket utvecklarna tycker att det är viktigt att tänka på.



Figur 12: Översiktlig arkitektur e-Me 2.0

Denna figur visar hur e-Me kommunicerar med tjänster, samt hur de interna tjänsterna och kärntjänsterna samarbetar.

4.3.1 Tekniker och ramverk

När integrationsplattformen skulle börja utvecklas tittade respondenterna först på ett agentramverk som heter JADE (Java Agent Development Framework)[4], som har möjlighet att exponera tjänster för tjänstearkitekturer. JADE hade enligt respondent C varit jättebra om man endast hade viljat ha agenter, och bara ville tillämpa det inom det ramver-

[4] http://jade.cselt.it/

ket. Det hade dock varit svårt att kommunicera med många av de andra delar som utvecklarna planerades bygga, då JADE är byggd som en färdig black box och inte har några kontaktytor utåt.

I stället såg respondent A och C ett behov av att bygga (eller använda sig av) en work flowhanterare. Det första respondenterna A och C tittade närmare på var JBPM (Java Business Process Management)[5]. Ingen av dem hade dock någon tidigare erfarenhet av denna work flowhanterare, och eftersom det redan hade beslutats att Spring[6] (också en produkt som ingen av utvecklarna hade någon erfarenhet av) skulle agera kommunikationsserver tillsammans med JBoss[7] (en applikationsserver, och ytterligare en produkt som ingen av utvecklarna hade någon erfarenhet av) så beslutades det att utvecklarna skulle bygga en egen work flowhanterare. De ansåg helt enkelt att det skulle bli alldeles för många nya produkter att komma på banan med.

Också Hibernate har använts i integrationsplattformen. Hibernate[8] är ett öppet källkodsprojekt, och ett persistence framework, som underlättar dataaccess. Enligt utvecklarna A och C har dem lagt cirka hundra till hundrafemtio timmar bara på att stångas med dessa produkter och ramverk, timmar som om de hade valt produkter de tidigare var bekanta med hade kunnat läggas på utveckling. Ett val hade istället kunnat vara att använda EJB[9] (Enterprise Java Beans) som utvecklarna var mer bekanta med. Då hade de hundra till hundrafemtio timmarna kunnat fokuseras mer på utveckling av den faktiska produkten. Utvecklarna A och C tycker dock att det har varit utmanande och spännande att lära sig nya tekniker och ramverk. De tycker också att det är viktigt att de utvecklar plattformen så att tjänsteleverantörerna och användarna skall behöva göra så lite som möjligt.

4.3.2 Hantering av tjänster

Det finns två sorters tjänster i integrationsplattformen: Interna samt externa. Enligt utvecklarna A och C är dock båda typerna av tjänster slutanvändartjänster eller konsumtionstjänster, tjänster som på något sätt påverkar och har en inverkan på slutprodukten. Tjänsterna är även tänkta att jobba mot applikationer som studenten redan idag använder, så att studenten sedan skall kunna utföra så mycket som möjligt av sina dagliga IT-sysslor i just integrationsplattformen e-Me. Tjänsterna kan vara väldigt enkla eller väldigt komplicerade: Det är upp till leverantören av tjänsten. Tanken är att vem som helst skall kunna skriva tjänster för e-Me, som sedan exponeras ut mot studenterna, som kan skapa prenumerationer på dessa.

I en prenumeration ställer man in vilken data man vill att tjänsten skall ha åtkomst till, samt även tjänstespecifika inställningar. I fallet med KronoX (en schematjänst) kan man till exempel ställa in vilka kurser och program man vill visa i sitt schema. Tjänsterna kan även kommunicera med varandra genom integrationsplattformen, dock kan de aldrig direkt läsa användarspecifik data. Detta sker genom e-Me, och plattformen fungerar därför

[5] http://www.jboss.org/jbossjbpm/
[6] http://www.springsource.org/
[7] http://www.jboss.org/
[8] https://www.hibernate.org/
[9] http://java.sun.com/products/ejb/

som en sorts integritetslås, så att användaren bara delar med sig av den data som den har godkänt att tjänster skall få tillgång till.

Tjänsterna kan fungera på två olika sätt: Antingen pushas data till tjänsten, som sedan hanterar den på något sätt, eller så pollar tjänsten regelbundet den datakälla den ansluter till. I det första fallet notifierar datakällan själv tjänsten om att det har skett uppdateringar. I det senare fallet är det alltså upp till tjänsten själv att kontrollera om det finns någon ny data som kan påverka prenumerationer av tjänsten. Ett exempel på det senare fallet är just KronoX-schematjänsten, som själv ligger och pollar schemadatakällan och kontrollerar om det har skett några schemaändringar. Om det har gjort det sedan den senaste kontrollen lägger den helt enkelt in den nya datan i användarens kalender.

För att pusha information till e-Me så exponerar integrationsplattformen en JMS-kö (Java Message Services[10]) för att ta emot arbetsflödesmeddelanden. En tjänsteleverantör kan alltså pusha in information i denna JMS-kö genom integrationsplattformen. Om de använder sig av det andra sättet kan dem som en av utvecklarna säger ”skita” i kön och lägga jobben direkt på stacken. Utvecklarna A och C har även diskuterat ett tredje sätt in: Nämligen att exponera en vanlig webbtjänst, och att därigenom ge utvecklarna möjligthet att säga ”Hej, här, ta emot!” till e-Me.

Det finns också ett antal kärntjänster i e-Me, som använder sig av det interna API:et. Dessa tjänster är tjänster som bör vara intressanta för många av de leverantörer som vill skriva tjänster för e-Me. Dessa tjänster kan vara till exempel notifiering eller kalender. I dagsläget har endast kalendern implementerats, och då endast Google Calendar[11]. Tjänsterna kan däremot aldrig prata direkt med varandra, utan all kommunikation måste gå genom själva integrationsplattformen e-Me.

API:er

För att skriva en tjänst för e-Me måste en tjänsteleverantör ladda ner ett paket som innehåller de filer som krävs, bland annat en abstrakt klass och gränssnitt som måste implementeras. Det finns två API:er: Ett internt och ett externt. Utvecklaren väljer det API som krävs för den tjänst som skall utvecklas och pluggas in i e-Me. De båda API:erna innehåller ett antal fördefinierade metoder som krävs för att tjänsten skall kunna fungera när den pluggas in i integrationsplattformen.

För att skapa en tjänst för integrationsplattformen måste tjänsteleverantören få tillgång till plattformens egen kod, och utvecklarna A och C har i största möjliga omfattning försökt göra koden så säker och med så hög integritet som möjligt, bland annat genom att i största möjliga utsträckning använda gränssnitt och liknande. Säkerheten och integriteten är en punkt som en av utvecklarna (respondent A) trycker extra hårt på: Studenten skall få behålla sin integritet i plattformen, och tjänsteleverantörer skall ej kunna gå in och läsa vilken data dem vill, eller göra vad de vill i plattformen. Det måste också undvikas att känsliga klasser exponeras utåt mot tjänsteleverantörerna, så att en leverantör ej skall komma åt till exempel databaskopplingen.

[10] http://java.sun.com/products/jms/
[11] http://www.google.com/calendar

En annan av utvecklarna (respondent C) hade viljat ha in en säkerhetsexpert som kommer in i projektet och täpper igen alla hål som finns, som utvecklaren uttryckte det. Han jämförde e-Me (i dagsläget) med en schweizerost: Naggande god men full av hål. Säkerheten är därför något som är extra viktigt att tänka på vid utveckling av ett gränssnitt eller en tjänst, och något som utvecklarna upprepade gånger lyfte upp som något oerhört viktigt.

För att skapa en tjänst för e-Mes integrationsplattform laddar alltså tjänsteleverantören ner ett Javapaket som innehåller de klasser och gränssnitt som krävs, och programmerar sedan mot detta. När leverantören sedan anser att tjänsten är klar laddar den upp tjänsten till e-Me, i form av ett Javapaket (JAR).

Ett sätt att hantera denna säkerhetsaspekt på är enligt en av utvecklarna (respondent C) att endast tillåta ett skriptspråk mot själva integrationsplattformen, och i detta skriptspråk endast exponera de klasser som ej är känsliga sett ur en säkerhetssynpunkt. Utvecklaren säger dock att detta kan vara väldigt svårt och komplicerat att göra, för att få det på banan. Ett annat sätt att kontrollera att en tjänst ej använder sig av känsliga klasser kan till exempel vara att söka igenom koden just när Javapaketet som innehåller tjänsten laddas upp.

Varje tjänst skall implementeras som en fristående klass, och skall implementera två gränssnitt, ett av dem definierar hur tjänsten skall anropas när den körs i ett arbetsflöde. När tjänsten körs i ett arbetsflöde anropas en metod `Call()` och så måste tjänsten specificera vilka attribut som skall gälla för att själva tjänsten skall dra igång, alltså vilka förutsättningar som måste finnas i systemet. Det kan till exempel vara att en viss annan tjänst skall ha gjort något annat före, och då bildar detta en förutsättning för att den aktuella tjänsten skall dra igång.

Här har utvecklarna A och C använt två designmönster som kallas för Split och Merge: De går ut på att en aktivitet delas upp i mindre delar (Split) som sedan sys ihop när de har slutförts (Merge). Detta leder till att alla tjänster som är förutsättningslösa exekveras på en gång, medan de som har förutsättningar väntar på att dessa förutsättningar skall bli uppfyllda innan de drar igång. Integrationsplattformen, som respondent C uttryckte det, skiter fullständigt i vilken ordning som tjänster körs i om de inte har några förutsättningar. I ett anropet `Call()` anropas rätt action på rätt tjänst. Ett action är helt enkelt den metod som skall anropas på tjänsten, när den skall exekvera.

De interna tjänsterna i e-Me (som implementerar det interna API:et) har tillgång till i stort sett allt i kärnan, medan de externa tjänsterna som sagt endast får läsa och skriva värden som de själva äger. Om flera tjänster delar på en prenumeration (som till exempel KronoX som är beroende av en kalendertjänst) så har tjänsterna endast tillgång till sin egen information, och de måste anropa den andra tjänsten om den manipulera någon information i den, och då är det endast den anropade tjänsten som vet vad som görs och vilken data som ändras.

4.3.3 Utvärdering av utvecklingsfasen

Utvecklarna (A, B, och C) anser att själva utvecklingen av plattformen har gått relativt bra, även om det var lite svårt att utveckla i början, då det var så mycket som var tvunget att komma på plats innan det gick att testa själva plattformen. Ett av de stora problemen som utvecklarna stötte på var hur tjänsterna skulle kommunicera med varandra, utan att få åtkomst till den andra tjänstens data. Det har också varit svårt att få en uppfattning om själva flödet när en prenumeration innehåller fler än en tjänst, vad kommer att hända härnäst i flödet? Detta gör också att det blir väldigt svårt att logga, eller i alla fall att följa loggen och se vad som egentligen har gått fel. Det blir ganska fort komplext, enligt en utvecklare. Enligt samma utvecklare är detta dock ej något problem gränssnittsmässigt.

Enligt utvecklarna A och C har de lyckats väldigt bra med att sandboxa in de externa leverantörernas tjänster, så att dessa ej skall kunna få åtkomst till information de inte bör ha åtkomst till. Integritetsfrågan har varit väldigt central för utvecklarna, och den är på något sätt sammankopplad med säkerhetsfrågan: Att inte tillåta externa leverantörer att få åtkomst till personlig data som användaren inte har godkänt att de har tillgång till. Respondenten C tycker också att det är spännande att e-Me eventuellt kommer att sitta på väldigt mycket profildata, som kan berätta mycket om studenters vanor, till exempel hur många studenter i en viss ålder och ett visst kön som är intresserad av en specifik sak just nu. Han anser att det kan vara en stor affärsmöjlighet och intäktskälla att sälja denna data, för att kunna hålla plattformen vid liv, utan att sälja ut integriteten.

En av de stora fördelarna med integrationsplattformen är enligt utvecklarna (A, B och C) att den är väldigt flexibel: Det är lätt att prenumerera och konfigurera en tjänst för själva slutanvändaren, även om själva mekaniken bakom är komplex. Det kan dock vara väldigt svårt att få en överblick över plattformen när det väl finns många tjänster, då själva integrationsplattformen både exponerar och konsumerar tjänster. Detta leder till en hög komplexitet, och låg överblickbarhet.

Respondent B tror ej att det kommer att komma in speciellt många externa aktörer, utan att det främst kommer att vara universiteten själva som integrerar sina befintliga tjänster med integrationsplattformen, för att underlätta för studenterna. Han tror att det främst kommer att handla om att det är universitetens tjänster som vill kommunicera med externa tjänster, såsom Google Calendar eller Gmail, och att studenterna till exempel skall få sina mejl eller schemaposter skickade till dessa, istället för att behöva gå in på flertalet ställen och leta efter detta. En av de stora fördelarna är enligt denna utvecklare alltså att integrera befintliga tjänster från universitetens håll, för att förenkla studentens vardag. I dagsläget finns det oerhört många system på en del universitet, som studenten själv måste hålla reda på. e-Me skulle då kunna sy ihop dessa system och verkligen för studenten.

4.3.4 Återanvändning av tjänster i e-Me

Utvecklarna (A, B och C) hade i stort sett samma syn på SOA-miljön när det gäller e-Me: e-Me återanvänder egentligen inte själv tjänster, men fungerar som en aggregator för tjänster för slutanvändaren, det vill säga i det är fallet studenten. En del tjänster återanvänds dock av e-Me, men då är de implementerade som just tjänster i e-Me. Dessa tjänster är kärntjänster, såsom kalendern (just nu Google Calendar). Det är också tänkt att oli-

ka mejltjänster skall implementeras som kärntjänster, till exempel Gmail, och även MSN Live Messenger för notifiering.

De två tjänster som utvecklarna A och C själva har implementerat som tjänster i e-Me är KronoX och Google Calendar. De anser att dokumentationen för dessa tjänster har varit bristfällig, och i flera fall några versioner för gammal. Denna kommentar om dokumentationen gäller även de produkter som utvecklarna har använt, till exempel JBoss, Spring och Hibernate. En utvecklare (C) tycker att det är det största problemet med den öppna källkodsvärlden: Dokumentationen underhålls ej i fas med implementationen. Det har därför funnits vissa problem i implementationen av dessa tjänster och produkter, och utvecklarna tryckte hårt på att det är viktigt med god och utförlig dokumentation kring olika API:er och produkter.

4.3.5 SOMA och SOMF i e-Me

Utvecklarna (A, B och C) ansåg närmast unisont att både SOMA och SOMF var alldeles för generella, och i stort sett hade kunnat passa in på vilket systemutvecklingsprojekt som helst, om man bara bytte ut några ord i benämningarna. De hade dock en mer positiv inställning till SOMA än till SOMF. Utvecklarna ansåg också att en just SOMA som modell (graf) sett inte sade alltför mycket, utan hade välkomnat en utförligare modell. En av utvecklarna hade svårt att se själva SOA-delen i SOMA, och varför just den modellen skulle vara så bra för SOA när den ser ut som i stort sett samma generella systemutvecklingsmodell som helst.

De anser dock att den på ett bra sätt fångar hur sammanflätade de olika aktiviteterna är, och de beroenden aktiviteterna har sinsemellan. Det kunde dock vara bra att ha en klar startpunkt i modellen. En av utvecklarna (respondent A) tycker att SOMA fångar det alla redan vet och redan utvecklar efter, och att den alltså är rätt så meningslös. Han anser också att modellen främst utgår från att man vill hitta tjänster och utnyttja dessa, men för projektet e-Me vill man istället bygga tjänster och möjliggöra byggandet av tjänster. Han säger också att projektet e-Me inte främst arbetar med tjänster, utan själva gränssnittet som möjliggör integration av tjänster.

En av utvecklarna (respondent C) anser att det skulle vara intressant att tillämpa SOMA i något projekt på Högskolan i Borås, och testa metoden och se hur väl den fungerar i verkligheten. Han tyckte dock att metoden inte kändes särskilt agil, och ville ha in testning som en större del av själva metoden. Han anser att det är oerhört viktigt att jobba outside in, och jobba med Test-Driven Development. Det är också oerhört viktigt att veta vilken målgrupp man jobbar mot. Han anser också att det är oerhört viktigt att inte fastna i modellen, bara för att följa den, då det negativt påverka budgeten, och man kan fastna i modellen så att verkligheten springer ifrån. Då kanske man till slut står med endast en förstudie men ingen budget kvar. Utvecklaren anser att det är oerhört viktigt att få med sig agile, och speciellt TDD, när man arbetar med en SOA.

När det gäller SOMF anser utvecklarna (A, B och C) att den är väldigt otydlig och inte har något klart flöde. Det är svårt att förstå sig på den, och vad författaren egentligen menar. Precis som innan tycker de också att den känns som en beskrivning av all utveckling

man gör, och inte specifikt utveckling för SOA. Respondent A utrycker det som att det känns som att författaren har tagit en generell systemutvecklingsmodell, och bytt ut development och istället skrivit service.

Respondent B tycker också att det verkar som om man endast skall ta fram en enda tjänst i SOMF, när en SOA i själva verket oftast är uppbyggd av många fler tjänster. Han tycker inte att modellen var så bra, och svår att förstå sig på. Han undrar också vad som menas med discipline i sammanhanget. Respondent A tycker att SOMF känns väldigt mycket vattenfall, vilket även styrks av en av de andra utvecklarna, som tycker likadant. Utvecklarna tycker också att testning borde ha en tydligare roll i modellen. Respondent A tycker också att det saknas ett par saker i modellen, att den ej är fullständig. Han tycker även att det är lite märkligt att författaren har listat best practices.

Utvecklarna A och C anser också att SOMF är en ren akademisk modell, utan större koppling till verkligheten, och att de första två stegen ej i verkligheten kan göras i någon fördefinierad, strukturerad form. De tycker också att det tar alldeles för lång tid med för mycket dokument i denna modell, innan man faktiskt börjar implementera och kan kontrollera om det faktiskt fungerar som det ska för och mot kunden.

4.3.6 Utvecklarnas viktigaste erfarenheter

Utvecklarna A och C tycker att projektet har gett dem nya insikter om hur man egentligen bygger gränssnitt, och hur omfattande det egentligen är. De anser också att de har fått ut mycket av projektet, i form av nya erfarenheter och tekniker. Respondent C tycker att det är oerhört viktigt att köra outside in, även fast det inte har skett i e-Mes fall. Utvecklaren fick denna insikt på en konferens, och när han fick den kände han att han ville gå hem och skriva om all kod i e-Me. Efter närmare reflektion, och efter att han tänkt igenom det, kom han dock fram till att koden var relativt bra.

Han anser också att det är väldigt viktigt med Co-design, och att kommunicera tydligt med kunden. Han anser också att det inte är utvecklarens ”fetischer” som skall avgöra vad som byggs, utan det är kunden som skall få vad den vill ha. Där det har gått dåligt i e-Me, har det gått dåligt i stort sett för att tester inte har genomförts på ett tillräcklig bra sätt, enligt utvecklaren. Han anser dock att det inte hade gått att arbeta fullt outside in i just e-Me, utan föreslog ett tredje sätt, mittifrån och ut, och det är enligt honom vad e-Me har gjort.

Respondent B anser att det är väldigt viktigt att ha samspelet mellan kund och utvecklare, och att plattformen faktiskt skall hjälpa till med något. Han anser också att det är väldigt viktigt att jobba outside in, och att det inte är tekniken som skall bestämma hur produkten blir, utan användarnas (kundernas) förväntningar.

4.3.7 Sammanfattning

För att få en överblick över projektet e-Me har vi intervjuat tre av systemutvecklarna som jobbar med detta projekt, som syftar till att ta fram en plattform för integration av externa tjänster, med studenter som slutanvändare. Tanken är att projektet skall underlätta för studenters vardag.

Integrationsplattformen exponerar två API:er, ett externt och ett internt. Det interna tillhandahåller större access till e-Mes kärnkod, medan det externa är mer restriktivt. Tjänsterna har två val när det gäller hur dem skall ta emot ny data: Antingen att pusha denna data, eller polla den. Om den pushas läggs den i en JMS-kö, används en poller läggs den direkt på stacken. Utvecklarna har tänkt mycket på säkerhets- och integritetsaspekten vid utvecklingen av plattformen, samt även att den skall vara väldigt flexibel och kunna användas till många olika syften.

De två tjänster som hittills har implementerats för e-Me är KronoX (schema) och Google Calendar. Det största problemet vid implementationen av dessa har varit den bristfälliga dokumentationen.

När det gäller SOMA och SOMF så anser utvecklarna att dessa egentligen inte går att applicera på projektet e-Me. De anser också att dessa modeller är alltför generella, och kan passa in på alla typer av systemutveckling, även sådana som inte rör SOA. Modellerna beskrivs också som för övergripliga och för svåra att förstå, och speciellt SOMF anses vara orealistisk, och en mestadels akademisk modell.

De erfarenheter som utvecklarna lyfter fram ur projektet är vikten av agilt arbete, Test-Driven Development, att dokumentation är väldigt viktigt för gränssnitt, samt att det är viktigt att tänka på säkerhet och integritet.

4.3.8 Kriterier för ramverket

Från de intervjuer vi har genomför som en del av denna kvalitativa genererande intervjustudie har vi identifierat följande kriterier vi kommer beakta i framtagandet av vårt ramverk:

- **Flexibilitet**. API:et måste vara flexibelt, för att kunna reagera på förändringar och ändrade förutsättningar.
- **Outside in (tester)**. Det är oerhört viktigt att ta vara på konsumentens förväntningar på API:et, och att skriva tester utifrån dessa **innan** implementationen av själva API:et påbörjas.
- **Riskanalys**. Det är viktigt att tänka igenom vilka risker det finns med att skapa ett API, och att tänka igenom vad man egentligen skall tillhandahålla genom detta. Denna analys är speciellt viktig ur en säkerhets- och integritetssynpunkt: Ger man konsumenten tillgång till känsliga resurser, eller känslig information som ej bör exponeras?
- **Prototyper**. Det är oerhört viktigt att snabbt visa upp något för slutanvändaren av produkten, och att ej spendera för mycket tid på att analysera ”sönder” i början av utvecklingen. Det är viktigt att snabbt få feedback från slutanvändaren på något konkret.
- **Co-design**. Det är oerhört viktigt att ta tillvara på de tänkta konsumenternas tankar och idéer, samt att identifiera dessa tidigt under projektet.
- **Dokumentation**. Det är oerhört viktigt att tillhandahålla en utförlig, lättförståelig och framförallt aktuell dokumentation, som ligger i fas med det faktiska API:et.

- **Funktionalitet**. Gör så mycket som möjligt för, så att utvecklaren/konsumenten slipper göra mer än nödvändigt för att använda API:et.
- **Enkelhet och tydlighet**. Ramverket skall på ett enkelt och tydligt sätt beskriva sitt syfte, samt den arbetsgång som krävs.

Dessa kriterier kommer att ligga till grund för vårt ramverk, tillsammans med de kriterier vi ämnar ta fram genom vår benchmarking. Kriterierna för denna benchmarking baseras delvis på denna intervjustudies resultat.

5 BENCHMARKING

I detta kapitel beskrivs den benchmarking som vi genomfört. Denna benchmarking tittade närmare på fyra "best-of-class"-API:er, utifrån ett antal olika kriterier, som är baserade på den studie av projektet e-Me vi genomfört. Resultatet av denna benchmarking är kriterier som kan användas i genereringen av ramverket.

I don't know how, but she is cheating! Nobody can be that attractive and this good at a videogame.
- Sheldon, The Big Bang Theory

I vår benchmarking har vi utgått från de fyra mest populära och använda API:erna för *Web 2.0*.[12] Dessa är enligt ProgrammableWeb (2009) Google Maps, Flickr, YouTube och Amazon eCommerce (baserat på antal applikationer som har gjort mash-ups[13] av dessa tjänster). De kriterier vi kommer att bedöma dessa API:er utefter är följande:

- **Funktionalitet.** Hur mycket funktionalitet innehåller API:et, i förhållande till applikationen som exponerar det?
- **Plattform/Tekniker.** Vilka plattformar eller tekniker måste användas för att använda sig av API:et? I vilken utsträckning är dessa tekniker tillgängliga?
- **Kompabilitet.** Med vilka tekniker, plattformar och protokoll är API:et kompatibelt med?
- **Dokumentation.** Hur väldokumenterat är API:et? Finns det till exempel referenslitteratur eller exempel på användning?
- **Svårighetsgrad.** Hur svårt bedöms det vara, utefter ovanstående punkter, att implementera en tillämpning av API:et?

Dessa kriterier kommer att vara grundande i vår benchmarking av ovan nämnda API:er. Dessa kriterier har kommit fram som en delprodukt av den studie vi gjort av integrationsplattformen e-Me.

5.1 Google Maps

Google Maps är en tjänst som tillhandahåller kartor samt vägbeskrivningar. Kartorna är dynamiska och interaktiva. Google Maps tillhandahåller ett API för att utnyttja dessa kartor i webbapplikationer, mobilapplikationer samt övriga typer av applikationer. (Google 2009a)

[12] Web 2.0 är ett begrepp som definierar den nya webben som en kommunikationsplattform, där användarna är med och formar innehållet. (O'Reilly 2005)
[13] En mash-up är en applikation som sammanställer information och funktionalitet från av varandra oberoende tjänster. (Krohn 2008)

5.1.1 Funktionalitet

Google Maps innehåller funktionalitet uppdelat i fem olika paket: Basklasser, Eventklasser, Controlklasser, Overlayklasser och Serviceklasser. Dessa klasser reprenterar i stort sett all funktionalitet som finns i Googles egna Mapapplikation. I dessa klasser finns det stöd för att använda olika språk och olika lokaliserade format (till exempel vad gäller avstånd). Utifrån denna mängd funktionalitet, samt dess relation till den funktionalitet som Googles egna Mapapplikationer har, anser vi att funktionaliteten är mycket hög. (Google 2009c)

5.1.2 Plattform/Tekniker

Googles API:er för Maps bygger JavaScript[14], XHTML[15] och VML[16]. Dessa tekniker är beprövade, och de flesta normala användare har stöd för dessa i sina webbläsare. API:erna kommunicerar via anrop i JavaScript, till Googles webbtjänster. Utvecklaren och användaren behöver dock ej konstruera dessa anrop själv, då dessa utomatiskt genereras av Googles kod, vilket gör det förhållandevis lätt och snabbt att utveckla mot detta API. API:ets teckenkodning är UTF-8[17], som innehåller 1 048 576 tecken, vilket gör att i stort sett alla tecken i alla möjliga språk finns tillgängliga. (JSolutions 2007) Tekniken gör att det är väldigt lätt att implementera en lösning byggd på API:et, och teckenkodningen gör att användare och utvecklare från världens alla hörn kan använda sig av API:et.

5.1.3 Kompabilitet

När det gäller kompabiliteten för API:et, så är alla webbläsare som är kompatibla med Googles egna Mapapplikation kompatibla med API:et och de applikationer som är byggda kring API:et. De flesta webbläsare klarar av att förstå JavaScriptet. (Google 2009b) Kompabiliteten är alltså hög, vilket leder till att de flesta kan använda API:et.

5.1.4 Dokumentation

Dokumentationen kring Google Maps API är väldigt omfattande och håller god kvalitet. Det finns en komplett utvecklingsguide, en komplett referens till API:et (där alla klasser och metoder beskrivs), flertalet kodexempel, flertalet videolektioner, samt fallstudier och information om vanliga frågor och vanliga problem. (Google 2009b) Dokumentationen ger en väldigt god grund till att utveckla mot API:et.

5.1.5 Svårighetsgrad

Det förefaller vara förhållandevis lätt, snabbt och effektivt att utveckla mot Googles API, framförallt på grund av den goda dokumentationen och de välkända teknikerna.

---

[14] JavaScript är ett skriptspråk, främst avsett för webbplatser och webbläsare. (Mozilla Foundation 2009)

[15] XHTML (Extensible Hypertext Markup Language) är en XML-baserad dokumenttyp som används för att visa webbsidor. (W3C 2002b)

[16] VML (Vector Markup Language) är en sorts utökning av HTML som stödjer vektorbaserad grafik. (W3C 1998)

[17] UTF-8 (Unicode Transformation Format-8) är en teckenkodningsstandard för Unicodetecken. (UTF-8 2008) Unicode är en mer generell standard för teckenkodning. (Davis 1999)

5.1.6 Sammanfattning

Google Maps API förefaller som sagt vara enkelt att utveckla mot, samtidigt som det har potentialen att nå ut till en väldigt stor skara utvecklare och användare. Dokumentationen håller god klass, och kompabiliteten är hög.

5.2 Flickr

Flickr är en tjänst som gör det möjligt att dela med sig av sina fotografier videoklipp direkt på webben. Enligt Flickr själva är det den bästa fotohanterings- och delningsapplikationen i världen. Flickr har två huvudmål: Att låta användare dela med sig av sina foton till de personer som står dem närmast, samt att göra det möjligt att organisera foto och video. Flickr tillhandahåller ett API som gör det möjligt att bland annat ladda upp och hämta foton. (Flickr 2009a)

5.2.1 Funktionalitet

Den funktionalitet som Flickrs API tillhandahåller är i stort sett den samma som Flickrs egen webbapplikation. Det finns dock en del undantag, det går inte att skapa allt som går att skapa i deras egen applikation (till exempel grupper). Kärnfunktionaliteten (exempelvis autentisering, ladda upp innehåll, hämta vänner) och liknande finns dock. (Flickr 2009b) Sammfattningsvis är funktionaliteten god, ställt i relation till den funktionalitet som finns i webbapplikationen.

5.2.2 Plattform/Tekniker

Flickr använder sig av olika tekniker för request[18] och response[19]: För request REST[20], SOAP[21] och XML[22]/RPC[23], och för response REST, SOAP, XML/RPC, PHP[24] samt JSON[25]. Dessa protokoll är standardprotokoll, och kan användas i flertalet olika applikationer, ramverk och tekniker, vilket ger utvecklarna en stor frihet. Det är alltså lätt att implementera applikationer mot API:et i den teknik som passar utvecklaren bäst. Även i detta API är teckenkodningen UTF-8, vilket ger stora lokaliseringsmöjligheter.

5.2.3 Kompabilitet

API:et är kompatibelt med REST, SOAP och XML/RPC, vilket gör att det är kompatibelt med de allra flesta tekniker för att utveckla applikationer. Detta skapar stor frihet för utvecklare av applikationer mot API:et. (Flickr 2009b)

---

[18] En request är en förfrågan om att få använda en webbtjänst. (W3C 2002a)
[19] Ett response är ett svar på en webbtjänstförfrågan (typiskt ett resultat av en metodkörning). (W3C 2002a)
[20] REST (Representational State Transfer) är en tillståndslös metod för att göra requests och ta emot responses. (Tyagi 2006)
[21] SOAP (Simple Object Access Protocol) är ett XML-baserat protokoll som låter applikationer utbyta data over HTTP.
[22] XML (Extensible Markup Language) är ett metaspråk som används för att strukturera data. (W3C 2001)
[23] RPC (Remote Procedure Call) är ett protokoll som kan användas av en applikation för att begära ett proceduranrop i en annan applikation. (Berners-Lee 1999)
[24] PHP (Hypertext Preprocessor, alt. Personal Home Page) är ett skriptspråk speciellt lämpat för webbapplikationer. (The PHP Group 2009)
[25] JSON (JavaScript Object Notation) är ett lättviktsformat för dataöverföring. (JSON 2009)

5.2.4 Dokumentation

Dokumentationen kring API:et är omfattande och god, dock är den ej så omfattande som Google Maps dokumentation. Det finns tillräckligt mycket information för att utvecklare snabbt skall komma på banan med API:et. Dokumentationen innehåller en överblick över API:et, en komplett referens (metoder), information om teckenkodning, autentisering och informationshantering, samt flertalet exempelimplementationer. (Flickr 2009b)

5.2.5 Svårighetsgrad

Eftersom Flickr bygger på standardprotokoll för webbtjänster kan svårigheten att bygga implementationer av API:et sammanfattas som att klarar utvecklaren av att använda vanliga webbtjänster, så klarar utvecklaren av att använda Flickrs API.

5.2.6 Sammanfattning

Flickrs API har en god och omfattande dokumentation som klart och tydligt visar hur implementationer mot API:et utvecklas, och dess standardiserade överföringstekniker skapar stora friheter för utvecklare. Funktionaliteten är även den god.

5.3 YouTube

YouTube är den största videodelningswebbplatsen på Internet, där användarna lätt och enkelt kan dela med sig av sina videoklipp, och där andra användare sedan kan svara och kommentera på dessa. YouTube tillhandahåller fyra olika API:er, vi behandlar dock dessa som ett gemensamt, då det ändå är liknande funktionalitet i de olika API:erna. (YouTube 2009)

5.3.1 Funktionalitet

YouTubes API:er är byggda kring fyra olika huvudfunktioner (och behandlas av YouTube som fyra olika API:er, dock är dem snarlika): Åtkomst till data (användare, videoklipp, kommentarer), åtkomst till YouTubes egen mediaspelare, åtkomst till YouTubes anpassade mediaspelare (samma som föregående men med större valmöjligheter) samt widgets[26]. Detta är i stort sett den funktionalitet som YouTube tillhandahåller, funktionaliteten som exponeras i API:et är därför väldigt omfattande och god. (Google 2009d)

5.3.2 Plattform/Tekniker

De plattformar och tekniker som YouTubes API använder är HTTP[27], XML, Adobe Flash Player[28] samt XHTML/DOM[29]. Dessa tekniker är standardtekniker (Adobe Flash Player måste dock installeras separat, och ingår från början ej i någon webbläsare). De allra flesta användare och utvecklare kan därför utnyttja YouTubes API. (Google 2009d)

---

[26] En widget är en väldigt liten applikation som har ett väldigt specifikt användningsområde. (Princeton 2009)

[27] HTTP (Hypertext Transfer Protocol) är standardprotokollet för att överföra webbplatsdata. (Running 2007)

[28] Adobe Flash Player är en plattformsoberoende webbläsarmodul för webbmaterial. (Adobe 2009)

[29] DOM (Document Object Model) är ett plattforms- och språkoberoende gränssnitt för åtkomst och dynamisk uppdatering av ett dokuments innehåll, struktur och presentation. (Berners-Lee 1999)

5.3.3 Kompabilitet

Adobe Flash Player krävs som sagt för att kunna utnyttja API:et, både för utvecklare och användare. Detta är dock kostnadsfritt och kan relativt lätt installeras. De andra teknikerna är standardtekniker, som finns i alla moderna webbläsare, vilket ger en förhållandevis hög kompabilitet. (Google 2009d)

5.3.4 Dokumentation

Dokumentationen är precis som för Google Maps, det vill säga väldigt omfattande, utförlig och god. (Google 2009d)

5.3.5 Svårighetsgrad

Svårighetsgraden för att implementera en applikation som jobbar mot YouTubes API är ej hög, då API:et använder sig av standardtekniker och från början har gjort väldigt mycket för utvecklaren. Att utveckla en applikation mot detta API är effektivt, och tar ej lång tid.

5.3.6 Sammanfattning

YouTubes API är lätt, snabbt och effektivt att implementera mot, och använder sig till största delen av standardtekniker som moderna webbläsare har. Adobe Flash Player måste dock installeras separat, vilket drar ner kompabiliteten en aning. Dokumentationen är precis som för Googles andra API väldigt utförlig och god.

5.4 Amazon eCommerce

Amazon eCommerce (även kallat Amazon Product Advertising API) är ett API som tillhandahåller funktionalitet för att hämta produktdata från Amazons produkter. Det innehåller också funktionalitet för att hämta användardata och tjäna pengar (genom att länka vidare till Amazon). (Amazon 2009)

5.4.1 Funktionalitet

Amazon eCommerce-API:et ger tillgång till en begränsad del av Amazons funktionalitet. API:et exponerar metoder som ger tillgång till information om Amazons sortiment och sökfunktioner, samt även vissa delar av kunddatan och webbutiksfunktionaliteten. Det är även möjligt att tjäna pengar genom API:et, då utvecklaren kan få en procentsats av inköp som har länkats genom utvecklarens applikation. Totalt sett är andelen funktionalitet relativt låg i jämförelse med Amazons egen applikation. (Amazon 2009)

5.4.2 Plattform/tekniker

Amazons API använder sig precis som de andra API:erna av standardtekniker såsom HTTP (REST) och XML (SOAP). Detta ger stor frihet för utvecklarna, då dessa tekniker går att implementera i flertalet ramverk, plattformar och språk. (Amazon 2008)

5.4.3 Kompabilitet

Kompabilitetsfrågan ligger i detta API mest på utvecklarens applikation, då Amazon inte tillhandahåller något användargränssnitt, och API:ets överföringsteknik är standardiserad. Detta gör att kompabiliteten främst är utvecklarens ansvar. (Amazon 2008)

5.4.4 Dokumentation

Även Amazons API är förhållandevis bra dokumenterat, dock ligger mycket av dokumentationen på användarna själva, och på den community som finns på Amazon eCommerce. Det finns flertalet exempel, och en fullständig referenslitteratur om API:et, dock kan implementationsexemplen ses som ganska begränsade. Det är även viktigt att notera att Amazon har ett något annorlunda licensavtal än de andra API:erna, då utvecklaren även kan tjäna pengar genom detta API. (Amazon 2009)

5.4.5 Svårighetsgrad

Det är något svårare att implementera mot detta API än de föregående, då funktionaliteten och dokumentationen är mer begränsad. Utvecklaren får inte heller lika mycket gratis som i de andra API:erna, då Amazon inte tillhandahåller något användargränssnitt. Detta kan orsaka problem med kompabilitet gentemot slutanvändaren, då det är utvecklaren som är ansvarig för användargränssnittsfrågorna.

5.4.6 Sammanfattning

Amazons API är svårare att implementera mot gentemot de andra API:erna, och dokumentationen är något mer begränsad. Dock så kan utvecklaren få mer frihet när det gäller presentationen av den data som API:et erbjuder.

5.5 Sammanfattning

Alla API:er i vår benchmarking har gemensamma nämnare: God och utförlig dokumentation, de använder standardtekniker, de är väldigt kompatibla med moderna webbläsare, samt att de ger utvecklaren rätt så mycket funktionalitet och frihet. De kriterier vi tar med oss från benchmarkingen in i genereringen av vårt ramverk blir därför:

- Vikten av god och utförlig dokumentation kring API:et.
    - Exempelvis komplett referenslitteratur, goda exempel, guider, community
- Användning av standardiserade tekniker
    - Främst när det gäller kontraktet för API:et
- Säkerställ så hög kompabilitet som möjligt
    - Också detta genom användande av standardiserade tekniker i så hög utsträckning som möjligt
- Ge utvecklaren så mycket funktionalitet som möjligt, samtidigt som det inte får inkräkta för mycket på utvecklarens frihet att just utveckla.

Vi kommer som sagt att ha dessa kriterier i åtanke vid genereringen av vårt ramverk.

6 RAMVERK VERSION 1

I detta kapitel beskrivs den första versionen av ett nytt ramverk för framtagning av tjänsteorienterade API:er. Detta ramverk baseras på de föregående tre kapitlen.

I've got an idea - an idea so smart that my head would explode if I even began to know what I'm talking about.
- Peter Griffin

Utifrån de kriterier vi har tagit fram genom de två föregående delstudierna i denna studie, är det nu hög tid för oss att ta fram ett ramverk för modellering, design och utveckling av API:er, med fokus på tjänsteorienterade API:er.

6.1 Kriterier för ramverk

De kriterier vi har tagit fram från de två föregående delstudierna i denna studie, som ligger till grunden för vårt ramverk är:

- **Flexibilitet**. API:et måste vara flexibelt, för att kunna reagera på förändringar och ändrade förutsättningar.
- **Outside in (tester)**. Det är oerhört viktigt att ta vara på konsumentens förväntningar på API:et, och att skriva tester utifrån dessa **innan** implementationen av själva API:et påbörjas.
- **Riskanalys**. Det är viktigt att tänka igenom vilka risker det finns med att skapa ett API, och att tänka igenom vad man egentligen skall tillhandahålla genom detta. Denna analys är speciellt viktig ur en säkerhets- och integritetssynpunkt: Ger man konsumenten tillgång till känsliga resurser, eller känslig information som ej bör exponeras?
- **Prototyper**. Det är oerhört viktigt att snabbt visa upp något för slutanvändaren av produkten, och att ej spendera för mycket tid på att analysera ”sönder” i början av utvecklingen. Det är viktigt att snabbt få feedback från slutanvändaren på något konkret.
- **Co-design**. Det är oerhört viktigt att ta tillvara på de tänkta konsumenternas tankar och idéer, samt att identifiera dessa tidigt under projektet.
- **Dokumentation**. Det är oerhört viktigt att tillhandahålla en utförlig, lättförståelig och framförallt aktuell dokumentation, som ligger i fas med det faktiska API:et.
- **Enkelhet och tydlighet**. Ramverket skall på ett enkelt och tydligt sätt beskriva sitt syfte, samt den arbetsgång som krävs.
- **Funktionalitet**. Gör så mycket som möjligt, så att utvecklaren/konsumenten slipper göra mer än nödvändigt för att använda API:et.
- **Användning av standardiserade tekniker**. Detta gäller främst för API:ets kontrakt, som bör kommunicera på ett standardiserat sätt.
- **Kompabilitet**. Säkerställ så hög kompabilitet som möjligt, särskilt om API:et exponerar ett grafiskt gränssnitt för slutanvändare.

Utifrån det som utvecklarna har valt att lyfta fram, samt utifrån den benchmarking vi har genomfört har vi även prioriterat kriterierna, efter hur viktigt det är att de finns med i vår modell. Denna prioritering visas i *Tabell 3: Prioritering av kriterier för ramverket*.

Kriterium	Prioritet
Flexibilitet	HÖG
Outside in (tester)	HÖG
Riskanalys	HÖG
Prototyper	HÖG
Co-design	HÖG
Dokumentation	HÖG
Enkelhet och tydlighet	HÖG
Funktionalitet	MEDEL
Användning av standardiserade tekniker	MEDEL
Kompabilitet	MEDEL

Tabell 3: Prioritering av kriterier för ramverket

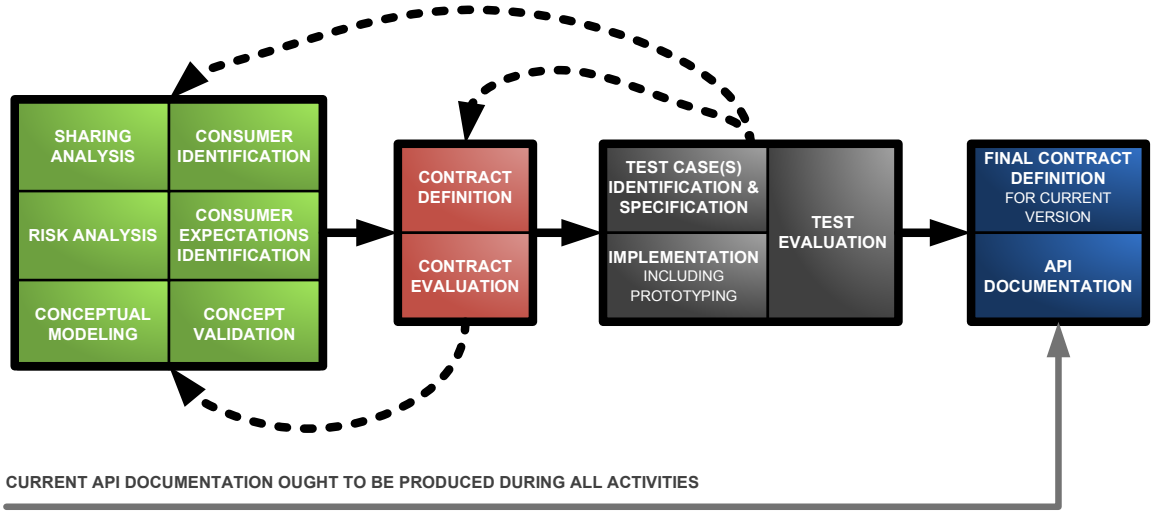
De kriterier ovan som har markerats med hög prioritet, är de punkter som utvecklarna starkast lyfte fram under våra intervjuer, samt de punkter som starkast identifierades under benchmarkingen som viktiga. De övriga tre kriterierna föreföll vara viktiga under själva benchmarkingen, men var inte lika framträdande under intervjuerna. Dessa är självklart ändå viktiga att ta hänsyn till.

6.2 SOAPIF – Service-Oriented Application Programming Interface Framework

Vi har valt att kalla vårt ramverk för Service-Oriented Application Programming Interface Framework, förkortat SOAPIF. Vi har valt detta namn eftersom det klart och tydligt beskriver vad ramverket är till för, samtidigt som vi anser att förkortningen är lätt att uttala, samtidigt som den även beskriver att det handlar om SOA (**SOA**PIF) och API:er (SO**API**F). Detta återknyter till kriteriet att ramverket skall vara enkelt och tydligt. SOAPIF innehåller fyra övergripande faser: *Conceptualization*, *Definition*, *Testing & Implementation* och *Delivery*. Dessa faser visas i *Figur 13: SOAPIF Phases*. Det är tänkt att dessa faser är tätt sammanknutna, och att de återbesöks i iterationer.

Figur 13: SOAPIF Phases

Vi har även valt att färgkoda de olika faserna i modellen, för att klart och tydligt kunna visa vilka aktiviteter som hör till vilken fas, också detta för att förenkla och förtydliga den grafiska representationen av modellen. De olika faserna är uppdelade i aktiviteter, vilket syns i *Figur 14: SOAPIF Activities*, och de är mappade med färgerna gentemot faserna.



Figur 14: SOAPIF Activities

Dessa faser går som sagt in i varandra, vilket vi försöker illustrera med de streckade looparna som är tänkt att vara återkoppling från de förekommande valideringarna av de tidigare faserna. Under hela processen bör dokumentation ske, vilket har poängterats både i våra intervjuer och under benchmarkingen. Det är viktigt att ta tillvara på konsumenternas önskemål och förväntningar, och att dokumentera dessa.

6.2.1 Conceptualization

Denna fas syftar till att identifiera vad man egentligen vill göra: Vilka är konsumenterna, vad vill dem ha, och finns det några risker med detta för applikationen som exponerar API:et, samt vad kan man exponera? Fasen består av sex olika aktiviteter: *Sharing Analysis*, *Consumer Identification*, *Risk Analysis*, *Consumer Expectations Identification*, *Conceptual Modeling* samt *Concept Validation*.

Sharing Analysis

I denna aktivitet analyserar man vilken information man egentligen kan exponera, och vilken information man är villig att exponera. Det är här viktigt att tänka över sådana frågor som säkerhet och integritet: Vilka konsekvenser kan det få att man delar med sig av känslig information, eller känslig funktionalitet? Och vilken information kan man egentligen dela med sig av?

Consumer Identification

Denna aktivitet syftar till att identifiera de tänkbara konsumenter som finns av tjänsten. Det är här viktigt att få en så representativ bild av konsumenterna som möjligt, och att

även försöka få med sig en del av dessa konsumenter, för att jobba med co-design i någon form. På detta sätt kan man få en bra input till aktiviteten *Consumer Expectations Identification*.

Risk Analysis

I denna aktivitet analyserar man de risker som finns genom exponeringen av funktionalitet och information. Aktiviteten är tätt sammanknuten med *Sharing Analysis*, men även med *Consumer Expectations Identification*. Det är i denna aktivitet viktigt att tänka på tänkbara scenarion gällande säkerhet och integritet, och vad som kan hända om det till exempel finns säkerhetshål eller om en tjänst till exempel delar med sig av användardata utan användarnas godkännande.

Consumer Expectations Identification

Denna aktivitet syftar till att identifiera de förväntningar, tankar och idéer som tänkbara, representativa konsumenter kan tänkas ha kring API:et. Den är tätt länkad till *Consumer Identification*, *Sharing Analysis* samt *Risk Analysis*. Inom aktiviteten kan till exempel ett co-designverktyg som workshop användas.

Conceptual Modeling

I denna aktivitet samlar man den information och erfarenheter som producerats i aktiviteterna *Sharing Analysis*, *Consumer Identification*, *Risk Analysis* och *Consumer Expectations Identification*. Här får utvecklarna väga dessa mot varandra, och försöka komma fram till en så bra konceptuell bild över API:et som möjligt. Detta beslut bör dokumenteras. Denna aktivitet bör producera en konceptuell bild över vad API:et kommer att exponera för någon funktionalitet, som sedan kan användas som input för en ytterligare iteration i denna fas (om *Conceptual Validation* på något sätt fallerar), eller som en grund för ett kontraktsförslag för API:et i fasen *Definition* och aktiviteten *Contract Definition* om aktiviteten *Conceptual Validation* resulterar i att den konceptuella modellen bedöms vara tillräckligt god. Det är dock viktigt att poängtera att dessa aktiviteter kan utföras parallellt, och att kanske endast vissa delar av den konceptuella bilden behöver arbetas om.

Conceptual Validation

Denna aktivitet syftar till att validera den konceptuella bild som tagits fram i aktiviteten *Conceptual Modeling*, och att stämma av den mot de risker, förväntningar samt den önskade funktionalitet som identifierats. Det är i denna aktivitet oerhört viktigt att inkludera alla intressenter, och nå en konsensus. Aktiviteten kan resultera i att hela den konceptuella bilden görs om, att delar av den görs om och andra delar börjar modelleras för ett kontrakt, eller att bilden bedöms vara tillräckligt god för att hela den konceptuella modellen skall omformas till ett kontrakt för API:et.

Koppling till kriterier

Denna fas handlar främst om co-design, och att i samarbete med de förväntade konsumenterna av API:et ta fram en konceptuell bild över hur API:et bör se ut. Denna fas kan därför kopplas till kriterierna:

- **Flexibilitet**. Genom att i samarbete med konsumenterna identifiera deras förväntningar på API:et är ramverket flexibelt för förändringar, vilket även gör API:et flexibelt. Det är också viktigt att poängtera att aktiviteterna med fördel genomförs samtidigt, och sammankopplat, vilket ytterligare ökar flexibiliteten.
- **Outside in (tester)**. Genom att hela tiden utgå från konsumentens förväntningar på API:et skapas goda förusättningar för tester, och arbetssättet är på detta sätt outside in.
- **Riskanalys**. Genom att identifiera risker tidigt i utvecklingen kan dessa hanteras, och förhoppningsvis undvikas. Att denna aktivitet sker tidigt kan också vara en fördel, då konsumenten klart och tydligt kan få en bild av varför något inte är exponerat i API:et.
- **Co-design**. Genom att blanda in alla intressenter i denna fas, och ta tillvara på deras förväntningar och synpunkter, skapas en god grund för co-design.
- **Dokumentation**. Att dokumentera de beslut görs, och de förväntningar som finns, är viktigt i alla faser, även denna. Det är viktigt att detta görs, så att det är lätt att göra till exempel ett överlämnande, och att skapa testfall senare i ramverket.

Vi anser att kriterierna ovan med god marginal uppfylls i denna första fas, genom att det i ramverket sker en relativt stor orientering mot konsumentens önskemål, samt en analys av de risker som finns med att exponera tjänster genom ett API.

6.2.2 Definition

Denna fas syftar till att definiera det kontrakt som gäller för själva API:et som utvecklas. Detta kontrakt kan förändras som en konsekvens av förändrade förutsättningar i faserna *Conceptualization* eller *Implementation*. Fasen innehåller två aktiviteter: *Contract Definition* och *Contract Validation*.

Contract Definition

I denna aktivitet försöker man utifrån den konceptuella bild som skapats i föregående fas att skapa ett kontrakt för själva API:et. Om utvecklarna direkt förstår att det inte går att genomföra, går man tillbaka till föregående fas. Om det verkar rimligt att kontraktet håller, går man vidare till aktiviteten *Contract Validation*. Kontraktet bör vara väldefinierat, utförligt och lättförståeligt dokumenterat.

Contract Validation

Denna aktivitet syftar till att validera det kontrakt för API:et som tagits fram i aktiviteten *Contract Definition*. Denna validering görs i samarbete med alla intressenter, och om valideringen fallerar går man tillbaka till föregående fas, *Conceptualization*, och undersöker på nytt förväntningar eller risker, beroende på vad som krävs. Därefter försöker man skapa ett nytt kontrakt.

Koppling till kriterier

Denna fas handlar främst om att definiera och validera det som har tagits fram i föregående fas. Denna fas handlar därför även den främst om co-design och dokumentation. De kriterier som vi anser att denna fas fyller är följande:

- **Flexibilitet**. Detta kriterium uppfylls då kontraktet i och med co-design är väldigt flexibelt för förändringar.
- **Co-design**. Detta kriterium uppfylls genom att alla intressenter är inblandade i valideringen av själva kontraktet.
- **Dokumentation**. Då kontraktet bör vara väldigt väldokumenterat och väldefinierat, anser vi att även detta kriterium uppfylls.

Vi anser att de kriterier som gäller co-design och användarinblandning, samt dokumentation, är väl uppfyllda i denna fas, då konsumentens förväntningar (vägt mot riskanalysen) är det som leder fram till själva kontraktet.

6.2.3 Implementation

Denna fas behandlar själva implementationen av API:et, även generering av prototyper. Denna implementation baseras på resultaten från de föregående faserna. Fasen innehåller de tre aktiviteterna *Test Case(s) Identification & Specification*, *Implementation, Including Prototyping* samt *Test Validation*, och kan fungera som input till de två tidigare faserna, genom prototyper, problem vid implementationen eller förändrade förutsättningar.

Test Case(s) Identification & Specification

I denna aktivitet skapas testfall, genom användning av den information om förutsättningar, risker, förväntningar och kontrakt, som har genererats i de tidigare faserna. Testfallen ligger sedan till grund för själva implementationen, denna implementation sker alltså outside in, enligt Test-Driven Development-prinicper. När testfallet är klart övergår fasen till aktiviteten *Implementation¸ Including Prototyping*. Om det ej går att skapa något testfall utefter de förutsättningar som givits, återgår arbetet till en tidigare fas.

Implementation, Including Prototyping

Denna aktivitet syftar till att faktiskt skapa något konkret, antingen en prototyp, ett färdigt API, eller en delmängd av ett API. Det är viktigt att fokusera på att skapa något som konsumenten och övriga delar av teamet faktiskt kan ge feedback på, och komma med förslag till förbättringar. Det är självklart också viktigt att det man implementerar passerar testfallen, samt att man dokumenterar väl. Aktiviteten kan återkoppla till tidigare faser och aktiviteter vid behov.

Test Validation

I denna aktivitet sker en validering, så att de testfall som har skapats kan appliceras på den implementation som tagits fram, så att det klart och tydligt går att se om denna implementation framgångsrikt kan passera testfallet och nå upp till konsumentens och de övriga intressenternas förväntningar, samtidigt som den minimerar de risker som identifierats. Om implementationen inte matchar testfallet, görs en återkoppling till tidigare faser, och antingen testfallet eller implementationen görs om enligt de förutsättningar, risker och förväntningar som identifierats.

Koppling till kriterier

Denna fas har kopplingar till både co-design, dokumentation samt prototyping. Det är viktigt att fortfarande involvera alla intressenter, även under implementationen och test-

ningen av implementationen, samt att dokumentera, och att skapa något som intressenterna kan ge feedback på.

- **Flexibilitet**. Detta kriterium uppfylls då implementationen kan förändras som ett resultat av ett fallerat test eller som en konsekvens av förändrade förutsättningar.
- **Co-design**. Detta kriterium uppfylls genom att alla intressenter är inblandade i valideringen av själva implementationen eller prototypen.
- **Dokumentation**. Implementationen bör vara väldokumenterad, både kodmässigt samt med fokus på en manual för hur man faktiskt använder API:et.
- **Outside in (tester)**. Det är viktigt att implementationen passerar de tester som satts upp, för att därigenom validera att implementationen faktiskt stämmer överens med de förväntningar som satts upp.
- **Prototyper**. Det är viktigt att tidigt skapa prototyper, så att konsumenten får en känsla för API:et och kan komma med feedback.

Vi anser att dessa kriterier är väl uppfyllda, då det finns delar av både co-design, Test-Driven Development samt prototyping i denna fas.

6.2.4 Delivery

Denna fas behandlar det sista steget i att ta fram ett API: Att leverera detta, det vill säga att slutligt definiera kontraktet för API:et, genom att revidera det kontrakt som tagits fram i *Definition*, och mappa detta mot det faktiska kontrakt som har implementerats. Detta görs i *Final Contract Definition, For Current Version*. Det är också viktigt att paketera dokumentation för API:et på ett överskådligt, lättförståeligt och utförligt sätt. Denna dokumentation måste vara i fas med den nuvarande versionen av API:et. Detta görs i aktiviteten *API Documentation*.

Final Contract Definition, For Current Version

I denna aktivitet definieras det slutgiltiga kontraktet för denna version av API:et. Det är alltså ej ett slutgiltigt kontrakt för API:et, utan endast ett slutligt kontrakt för denna version av API:et. Detta kontrakt baseras både på det kontrakt som definierats i *Definition*, samt den faktiska implementation som gjorts. När kontraktet är klart, beskrivs det i *API Documentation*.

API Documentation

I denna aktivitet sammanställs den dokumentation som gjorts under tidigare faser i ramverket. Det är viktigt att dokumentationen är i en sådan form att den är lättförståelig, strukturerad och lätt att ta till sig, samt i fas med den nuvarande versionen av API:et.

Koppling till kriterier

Denna fas är främst kopplad till kriteriet Dokumentation, då den enbart bygger på tidigare dokumentation. Den är löst kopplad till de övriga kriterierna, då dessa är relaterade till slutprodukten. Dessa kriterier är dock främst kopplade till de tidigare faserna.

6.3 Sammanfattning

SOAPIF (Service-Oriented Application Programming Interface Framework) är ett ramverk för utveckling av API:er för tjänsteorienterade arkitekturer. Det innehåller fyra faser: *Conceptualization*, *Definition*, *Implementation* samt *Delivery*. I Conceptualization försöker man skapa sig en bild av själva API:et som bör byggas, genom co-design i samarbete med alla intressenter (inklusive tilltänkta konsumenter). Detta mynnar ut i en konceptuell modell över API:et, som input till nästa fas, *Definition*. I denna fas försöker man utifrån den konceptuella bilden definiera ett kontrakt för API:et, som sedan valideras i samarbete med intressenterna.

I fasen Implementation implementerar man sedan API:et utefter det kontrakt som skapats, samt de förväntningar som finns på själva API:et. Denna implementation föregås av skapnadet av ett testfall, som efter implementationen sedan testas. Om implementationen passerar testet, och är ett färdigt API, så går man vidare till nästa fas, *Delivery*. I denna fas definierar man det slutgiltiga kontraktet för denna version av tjänsten, samt sammanställer den dokumentation som producerats på ett lättillgängligt och strukturerat sätt.

Det är viktigt att notera att det är tänkt att ramverket skall tillämpas i iterationer, och att faserna och aktiviteterna är tätt sammanknutna.

6.3.1 Koppling till kriterier

Vi anser att ramverket uppfyller samtliga kriterier. Flexibiliteten hos API:et som utvecklas med hjälp av ramverket blir stor, då konsumenten genom co-design är involverad i processen med att ta fram detta. Konsumentens förväntningar vägs även mot en risk- och delningsanalys (aktiviteterna *Risk Analysis* och *Sharing Analysis*), vilket involverar ytterligare intressenter. Implementationen är även den flexibel, då det går att gå tillbaka till tidigare faser och aktiviteter. Genom att jobba outside in på detta sätt, blir utvecklingen även som en konsekvens testdriven. Alla dessa steg skall enligt dessutom dokumenteras på ett lättförståeligt sätt, för att kunna återkoppla till tidigare faser och aktiviteter, och även underlätta för slutkonsumenten.

Vi har i framtagandet av detta ramverk strävat efter att uppfylla ytterligare ett högprioriterat kriterium: Att ramverket skall vara enkelt och tydligt. Vi har försökt hålla modellen på en relativt övergripande nivå, samtidigt som det ändå framgår vad som egentligen skall göras. Vi har även i förklaringarna av faserna och aktiviteterna försökt vara så tydliga som möjligt.

De medelprioriterade kriterierna anser vi uppfylls som en konsekvens av att co-design och outside in-perspektivet anläggs: Genom att ta vara på konsumentens förväntningar får man automatisk den funktionalitet, användning av standardiserade tekniker samt kompabilitet som är tillräcklig för just det API:et som tas fram. Om konsumenten inte har några krav på till exempel användning av standardiserade tekniker, är det heller inte nödvändigt att implementera detta.

7 VALIDERING AV RAMVERK

I detta kapitel beskrivs den validering vi gjort av vårt ramverk. Denna validering har skett genom en kvalitativ intervjustudie. Detta kapitel mynnar ut i eventuella förändringar som bör implementeras i vårt ramverk.

We must learn our limits. We are all something, but none of us are everything.
- Blaise Pascal

För att kontrollera att vi har förstått utvecklarna av e-Me rätt, och för att validera att den input vi har fått från dem lyser igenom vårt ramverk, har vi genomfört en kvalitativ validerande intervjustudie.

7.1 Respondenter

Vi har intervjuat samtliga utvecklare igen, denna gång genom två e-postintervjuer med respondenterna B och C (på grund av en av utvecklarnas tidsbrist och en av utvecklarnas ledighet) samt en fysisk intervju på InnovationLab, med respondent A. För en kort beskrivning av respondenterna, se *4.1 Respondenter*.

7.2 Validering av kriterier

När det gäller de kriterier vi tagit fram för vårt ramverk tyckte respondent A vi diskuterade detta med att vi hade hittat rätt. Han håller på med om att API:er bör vara flexibla, han gav dock ett exempel på en situation där de absolut inte bör vara det. Detta exempel handlade om ett API som han och en medarbetare utvecklade, där integritetsfrågan var av högsta vikt, därför var det viktigt att inte lämna något som helst utrymme för att kränka integriteten, och därav blev API:et ej flexibelt. Han anser dock att i 95% av fallen så bör ett API vara flexibelt. Flexibilitet är även väldigt viktigt inom agila systemutvecklingsmetoder, samt co-design, då man som utvecklare måste vara mottaglig för förändringar och förbättringar i förhållande till konsumenterna. Att ha en hög flexibilitet är även viktigt för metautvecklingsprocessen, där flexibilitet skapas genom att man låter användaren välja olika vägar inom en metodik.

Han anser att outside in-tänket är väldigt bra, och att man för att bygga API:et först måste ta reda på vem som vill använda API:et till vad. Också riskanalysen tycker han är viktig, speciellt ur säkerhets- och integritetsperspektiven. Han anser, precis som i outside in-tänket, att det är viktigt att blanda in slutanvändaren (konsumenten) för att få ett så bra API som möjligt, och ett led i detta kan vara att göra prototyper, vilket han definitivt tycker var ett bra kriterium. Detta kriterium kan även grundas i teorin om test-driven development, där man anammar precis detta perspektiv.

Dokumentation av API:et är enligt utvecklare A oerhört viktigt, och speciellt att dokumentationen uppdateras i takt med det faktiska API:et. Han påpekar dock att detta kan vara relativt svårt att göra, även för väldigt stora aktörer som skapar API:er. Som exem-

pel nämnde han Googles Calendar-API, som han har arbetat med i e-Meprojektet. Google hade uppdaterat sitt plugin, och utvecklaren installerade den nya versionen. Han fick då ett fel han inte fått innan, och det tog två timmar innan han kom på att det hade tillkommit ett beroende i API:et som han var tvungen att hantera, genom att ladda ner filerna för detta beroende och inkludera i projektet. Det nämndes ingenting om detta i Googles egna dokumentation, utan han hittade det genom att använda just sökmotorn Google. Man kan i detta fall dra paralleller till utvecklingsmetoden Scrum: Där man dokumenterar det som är viktigt, och inte bara dokumenterar för sakens skull. I Scrum dokumenterar man till exempel vad kunden vill ha, och i Googles fall borde man ha dokumenterat vilka beroenden gränssnittet vill ha.

Enkelhet och tydlighet är enligt respondent A nödvändigt för att snabbt kunna ta till sig en ny metod, modell eller ett nytt ramverk. Han höll också med författarna att de tre kriterier som identifierats med prioritet medel var korrekt prioriterade, detta eftersom dessa tre kriterier (funktionalitet, användande av standardiserade tekniker och kompabilitet) i någon mån är beroende av vad som framkommer under co-design: Hur och vad vill slutanvändaren egentligen ha? Det kan enligt utvecklaren i flera fall till exempel vara lämpligt med ett API som har en egen kommunikationsstandard, som passar för just det API:et specifika ändamål.

När det gäller kriterier att man skall jobba outside in, med tester, anser respondent B att det är oklart vilken typ av tester som menas. Han anser att den typ av testning som vi avser, programmatiska tester, ej är den enda typen av tester som kan användas, han föreslår till exempel användartester. Med fasen *Test Case(s) Identification & Specification* avser vi dock att mappa implementationen mot kontraktet, och därigenom säkerställa att konsumenternas förväntningar programmatiskt infrias, vilket kan ha varit lite otydligt. Denna utvecklare var inte heller helt säker på vad vi menade med grafiska gränssnitt under kriteriet *Kompabilitet*, det finns nämligen ingen närmare förklaring av detta i det underlag som respondenterna tog del av. Detta förklaras dock tidigare i denna studie.

Sammanfattningsvis så verkar det som om de kriterier vi tagit fram med hjälp av den tidigare studien samt benchmarkingen är bra för ändamålet, och tillför något till utvecklingen av själva ramverket. När det gäller den teoretiska valideringen av ramverket, så har kriterierna främst sin grund i agila systemutvecklingsmetoder, co-design och metautveckling.

7.3 Validering av ramverk

När det gäller själva ramverket hade de tre respondenterna lite skiljda åsikter: Två av utvecklarna (respondenterna A och B) tyckte direkt att ramverket och modellen var klar och tydlig, medan den tredje (respondent C) ansåg att det inte var så lätt att se att arbetet sker i iterationer. Den utvecklaren blev ”lurad” av att vi använder ordet faser, han anser att man aldrig kan gå tillbaka till en föregående fas. Han ansåg också att modellen såg lite ”vattenfall” ut, vilket de andra utvecklarna inte ansåg. Respondent C som ansåg att modellen såg ut som en vattenfallsmodell föreslog att vi döper om ”fas” till ”arbetsmoment” eller något liknande, samt att vi försöker förändra den grafiska representationen av ramverket till något som tydligare visar att arbetet sker iterativt, till exempel att göra model-

len i cirkulär form istället. När det gäller namnet på ramverket så anser två av utvecklarna (respondent A och B) att det är bra, då det fångar SOA, API och SOAP, och är relativt lätt att komma ihåg och uttala.

Respondenterna A och B tyckte direkt att modellen var klar och tydlig, och fick direkt ett positivt intryck av den. De anser att det ramverket tar upp är sådant som man som utvecklare av ett API faktiskt behöver ta upp, även om den kanske är lite av en idealbild: Man hinner eller har inte alltid möjlighet att ta upp allt i ramverket. De anser dock att man egentligen bör ta upp det. Respondent A tycker också att modellen går att applicera på projektet e-Me, och att vissa av aktiviteterna har genomförts i arbetet med e-Me. Han anser dock att vissa av aktiviteterna kan vara svåra att förstå enbart grundat på namnet: Till exempel Sharing analysis. Han anser att det behövs en skriftlig förklaring till hur man arbetar med ramverket, och att denna skrift framförallt beskriver de olika aktiviteterna.

Det respondent A tycker saknas i modellen är någon form av versionshantering: Hur ser det ut om man levererar API:et i olika omgångar? Han anser därför att det behövs någon form av inkrementell leverans av själva API:et, och att detta framgår av ramverket. Om det inte finns någon bra versionshantering, så kan det vara lätt att bryta kontraktet vid uppdateringar eller tillägg till API:et. Det är därför viktigt att kommunicera med konsumenterna, och att kommunicera ändringar av API:et i god tid. Ett exempel på detta är Ladoks API:er, som utvecklaren har jobbat med. Tidigare var Ladok benägna till att bryta kontrakten för sina API:er vid uppdateringar, nu har de enligt utvecklaren lärt sig av sina misstag, och försöker så gott det går med att inte bryta kontraktet. Detta tog respondent C upp under den första omgången intervjuer.

Det är också enligt utvecklarna A, B och C viktigt att iterativt förbättra API:er, genom utvärderingar av dessa i samarbete med alla intressenter. Som respondent A uttryckte det: ”Bugghantering och förvaltning kommer man aldrig ifrån, och det är något man måste leva med”. Det är därför viktigt med utvärdering.

Respondent B anser att fasen *Conceptualization* har ett mycket bra innehåll, men han ifrågasätter ordningen i den. Han anser att aktiviteten *Consumer Expectations Identification* eventuellt bör ligga före aktiviteten *Sharing Analysis*, samt att det bör ske någon sorts iterativt flöde mellan dessa. Han anser att konsumentens behov bör styra vad man delar med sig av. Denna utvecklare tycker också att det skulle vara intressant att förklara mer hur aktiviteten *Contract Validation* fungerar. Han anser också att det kan vara lite svårt att förstå vad termen *outside in* egentligen betyder.

När det gäller hur utvecklarna A och C upptäcker nya ramverk, metoder och idéer inom systemutveckling, så sade dem att det är relativt mycket som sprids från jobbarkompisar och bekanta, och därigenom når utvecklarna. De håller sig också uppdaterade genom till exempel forum och bloggar, samt communities som till exempel ASP.NET. När dem letar efter något specifikt, till exempel för att lösa ett specifikt problem, är det främst Google Search som används, och dem anser att detta fungerar bra.

Utvecklarna (A, B och C) anser alltså att det är viktigt att klargöra att arbetet med ramverket sker iterativt, samt att klart och tydligt förklara de olika aktiviteterna. Respondent B anser också att det vore bra att på något sätt i själva modellen över ramverket klargöra vilka intressenter som utför aktiviteterna. Även utvärderingar anser respondenterna A, B och C behöver införas, samt stöd för delleveranser och versionshantering.

De teorier som främst går att applicera på ramverket är agila systemutvecklingsmetoder och co-design. I ramverket sker utvecklingen iterativt och inkrementellt, vilket oftast är fallet i agila systemutvecklingsmetoder, samt i nära samarbete med användaren i alla faser, vilket ökar graden av co-design.

7.4 Nya kriterier och förändringsåtgärder

De nya kriterier och förändringsåtgärder som vi har identifierat utifrån de validerande intervjuerna är:

- **Iterationer**. Visa klart och tydligt att arbetet kan ske i iterationer.
- **Intressenter**. Visa vilka intressenter som är inblandade i vilka aktiviteter.
- **Utvärdering**. Inför utvärdering av API:et, i samarbete med alla intressenter.
- **Delleveranser/Versionshantering**. Inför och visa att API:er kan levereras inkrementellt.
- **Förtydliga *Contract Validation***. Hur sker denna validering?
- **Justera *Conceptualization***. *Sharing Analysis* bör ske i samband med *Consumer Expectations Identification*; konsumentens behov bör styra det som API:et exponerar, ej tvärtom. Ge också en bättre bild av att detta kan ske iterativt.

Vi kommer att i nästa kapitel att införa dessa förändringsåtgärder i vårt ramverk.

7.5 Sammanfattning

Sammanfattningsvis var intrycket av ramverket positivt, och det som saknades var främst en tydlig indikation om att det går att iterera genom de olika aktiviteterna, och att införa en versionshantering och stöd för delleveranser. Det är också viktigt att visa vilka intressenter som ingår i vilka aktiviteter.

8 RAMVERK VERSION 2

I detta kapitel implementeras de förändringsåtgärder som föregående kapitel genererat. Kapitlet avslutas med en genomgång av den andra versionen av vårt ramverk.

The best thing about a boolean is even if you are wrong, you are only off by a bit.
- Anonym

Utifrån de nya förändringsåtgärder och kriterier som framkommit genom den validerande studien kommer nu ramverket att förändras utifrån dessa.

8.1 Kriterier och förändringsåtgärder

I det föregående kapitlet (genom vår validerande studie av ramverket) identifierade vi fyra olika förändringsåtgärder vi nu ämnar införa. Dessa åtgärder är:

- **Iterationer**. Visa klart och tydligt att arbetet kan ske i iterationer.
- **Intressenter**. Visa vilka intressenter som är inblandade i vilka aktiviteter.
- **Utvärdering**. Inför utvärdering av API:et, i samarbete med alla intressenter.
- **Delleveranser/Versionshantering**. Inför och visa att API:er kan levereras inkrementellt.
- **Förtydliga *Contract Validation***. Hur sker denna validering?
- **Justera *Conceptualization***. *Sharing Analysis* bör ske i samband med *Consumer Expectations Identification*; konsumentens behov bör styra det som API:et exponerar, ej tvärtom. Ge också en bättre bild av att detta kan ske iterativt.

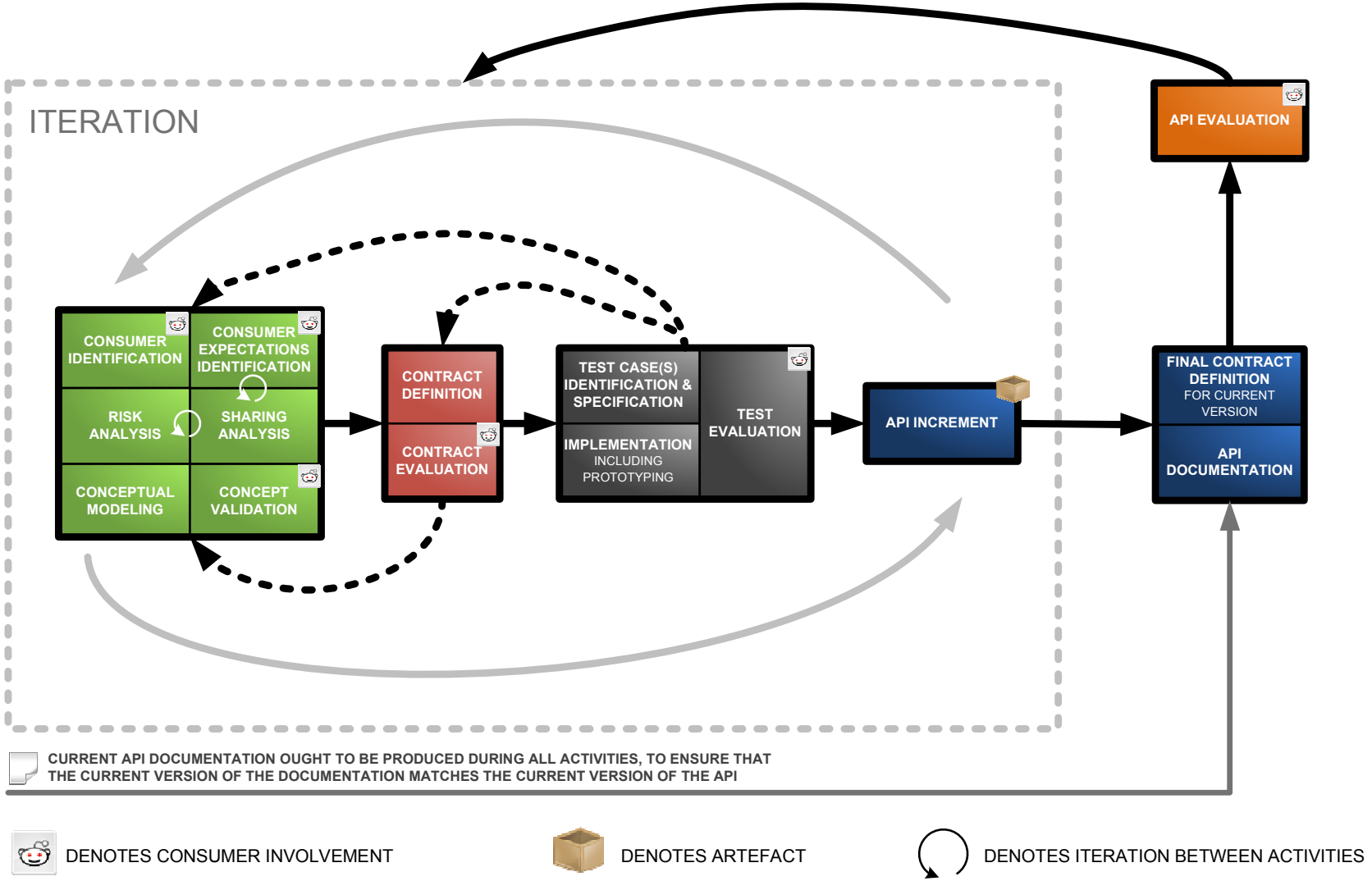
Alla dessa åtgärder ämnas införa i den nya versionen av ramverket.

8.2 SOAPIF 2.0

Vi har nu döpt om ”fas” till arbetsmoment, och infört ett nytt sådant arbetsmoment: *API Evaluation*. Detta nya arbetsmoment och dess relation till de andra momenten framgår i *Figur 15: Steps in SOAPIF 2.0*.

Figur 15: Steps in SOAPIF 2.0

Vi har även infört en tydligare indikering om att arbetet sker iterativt, samt en tydligare koppling till inkrementella leveranser och versionshantering. Utvärdering ingår numera också som arbetsmomentet *API Evaluation*. Det visas nu också i vilka aktiviteter som konsumenten är inblandad i. Dessa ändringar återspeglas i *Figur 16: SOAPIF 2.0 Steps & Activities*.



Figur 16: SOAPIF 2.0 Steps & Activities

8.2.1 Contract Validation

Contract Validation förklaras nu som:

> Denna aktivitet syftar till att validera det kontrakt för API:et som tagits fram i aktiviteten *Contract Definition*. Denna validering görs i samarbete med alla intressenter, genom att säkerställa att kontraktet fyller de förväntningar de har på detta. Om valideringen fallerar går man tillbaka till föregående fas, *Conceptualization*, och undersöker på nytt förväntningar eller risker, beroende på vad som krävs. Därefter försöker man skapa ett nytt kontrakt.

Det bör nu lite tydligare framgå hur vi har tänkt oss att denna utvärdering skall genomföras.

8.2.2 API Increment

Detta är en artefakt, och ingen aktivitet. Det är antingen ett färdigt API, om det räcker med en iteration för att skapa det, eller en delleverans av ett API. Denna artefakt hör därför till arbetsmomentet *Delivery*. Det är också värt att notera att det ej behöver levereras något i varje iteration; en leverans kan ha föregåtts av flera iterationer genom arbetsmomenten. Eftersom *API Increment* är en artefakt, och ingen aktivitet, har en notation för artefakter införts i modellen, vilket visas i *Figur 16: SOAPIF 2.0 Steps & Activities*.

8.2.3 API Evaluation

Denna aktivitet har tillkommit efter det identifierade behovet av en återkoppling gällande API:et. Det är viktigt att tillsammans med konsumenten utvärdera API:et, och ta vara på eventuella nya förväntningar eller problem. Denna utvärdering kan fungera som input till en ny iteration genom de olika stegen.

8.2.4 Koppling till kriterier

Vi har som sagt infört alla de förändringsåtgärder vi satte upp som en konsekvens av den validerande studien. Vi har för att tydligare visa att ramverket är iterativt ”boxat in” själva arbetsmomenten som leder fram till en inkrementell del av ett API, samt visat detta med tydligare återkopplingspilar. Detta uppfyller förändringsåtgärden, och gör samtidigt modellen enklare och tydligare. Själva ramverket blir också flexiblare genom denna tydligare iterationsmodell, då intressenterna får ännu större feedbackmöjligheter, vilket också ökar co-designinblandningen. Vi har utefter förslagen om arbetsmomentet *Conceptualization* även infört en tydligare bild av iteration i detta, samt ändrat ordningen så att det är konsumentens förväntningar som ligger till grund för *Sharing Analysis*, och inte tvärtom.

Även det nya arbetsmomentet *API Evaluation* bidrar till detta, då intressenterna tillsammans utvärderar ramverket efter leverans, för att säkerställa att det fungerar som planerat enligt kontraktet, och även för att fånga upp nya förväntningar från konsumenterna. *API Evaluation* uppfyller dessutom förändringsåtgärden som syftar till att utvärdera API:et. För att öka flexibiliteten i ramverket har artefakten *API Increment* införts. Denna artefakt är ett resultat av en iteration, och ökar delleverans- och versionhanteringsmöjligheterna i ramverket.

Det har också införts en notation för att markera att konsumenten är med i vissa aktiviteter, detta för att ytterliga öka tydligheten och enkelheten i modellen. Även en notation för artefakter har implementerats, och den appliceras på *API Increment*. Detta ökar enkelheten och tydligheten i presentationen av ramverket.

8.3 Teoretisk grund

Vårt ramverk kan kopplas till agil systemutvecklingsmetodik, och då främst Scrum. Inom den agila metodiken anser man att olika projekt har olika behov, och är därigenom beroende av olika typer av verktyg, färdigheter och kommunikation. Vi har därför anpassat vårt ramverk efter just API:er för tjänsteorienterade arkitekturer, och därigenom försökt säkerställa att ramverket passar för denna typ av utveckling. Ramverket är framtaget för att snabbt kunna svara på förändrade förutsättningar, och att kunna hantera variationer på ett bra sätt: Flexibilitet är verkligen ett nyckelord. Detta cementerar kopplingen till agilitet, och framför allt Scrum.

Inom Scrumcommunityn anser man att det i början av projektet ej är helt klart vad som projektet syftar till att utveckla: Projektet är en black box. Detta har vi i vårt ramverk tagit hänsyn till, med en stor användarmedverkan och regelbundna utvärderingar, vilket leder till att ramverket är förberett för snabbt skiftande behov och förutsättningar. Utvecklingen sker också iterativt och inkrementellt, vilket kan kopplas till Scrum (uppdelningen i sprintar). Användarnas förväntningar och krav på API:et dokumenteras också, vilket kan relateras till de backlogs som skapas inom Scrum. En ytterligare koppling till systemutvecklingsmetodik är kopplingen till Test-Driven Development: Tester är en stor och viktig del av vårt ramverk, och utvecklingen skall ske outside in, man tar alltså sin utgångspunkt i tester som är baserade på användarnas förväntningar vid utvecklingen.

Att användarna är med under hela utvecklingsprocessen och att man alltid tar hänsyn till deras förväntningar är en stor del av ramverket, och kan kopplas till co-design. Det är viktigt att försöka utveckla ett API som passar en ideal användningssituation så som konsumenterna som ser på den. En del av vårt ramverk är också framtagningen av prototyper (som sker under implementationsprocessen). Detta är för att validera att konsumenternas förväntningar på API:et uppnås.

En viktig del av SOA är att tjänster är separerade och konsekventa, och eftersom ett API skall ses som en black box så blir separeringen tydlig. Genom att man klart definierar och dokumenterar ett kontrakt för API:et, så blir det också konsekvent. Med hjälp av API:er kan också ett större problems lösning delas upp på mindre lösningar, som tillgängliggörs genom API:er. Dessa lösningar skall också vara väl avgränsade, vilket också är viktigt att tänka på vid utvecklingen av API:er. Detta leder också till att återanvändningen av API:et kan ske på ett bra sätt, och att det verkligen är återanvändbart.

Utav de principer som finns för en god SOA så anser vi att *Service contract*, *Autonomy*, *Abstraction*, *Composability* samt *Discoverability* täcks in i vårt ramverk, och med användningen av API:er. I ramverket är det viktigt att klart och tydligt definiera ett kontrakt, så att användningen blir konsekvent. API:et skall vara en black box, det vill säga användaren skall ej veta vad som händer bakom API:et. API:et skall också kunna använ-

das för att kombinera funktionalitet till nya tjänster eller API:er: Det är därför viktigt att blanda in konsumenten i alla arbetsmoment, för att säkerställa att dessa mål uppnås genom API:et. Dokumentation är också av yttersta vikt, och speciellt att göra denna lättillänglig och upptäckbar, för att få framtida konsumenter att upptäcka API:et.

Ramverket kan också kopplas till SOMA, då utvecklingen sker iterativt och inkrementellt, precis som i SOMA, och att alla arbetsmoment kan kopplas till varandra, och är beroende av varandra. Det går även att koppla ramverket till de olika faser som finns i SOMA: *Business Modeling And Transformation* kan mappas mot *Conceptualization*, *Specification* kan mappas mot *Definition*, och *Realization* kan kopplas till *Testing & Implementation*. Dessa faser och arbetsmoment har liknande innehåll, och det tyder på att vi har täckt in mycket av ”SOA-tänket” i vårt ramverk.

Ramverket och dess aktiviteter kan även i viss mån kopplas till SOMF, och tre av dess artefakter: *Conceptual Service*, *Design Service* samt *Solution Service*. Om man applicerar vårt ramverk på SOMF, så är en Conceptual Service det som är outputen från vårt första arbetsmoment (*Conceptualization*), *Design Service* blir det definierade kontraktet för API:et, och *Solution Service* blir det slutgiltiga API:et.

Även inom metautveckling är det viktigt med flexibilitet vid framtagandet av ett ramverk eller en metod; genom att vi i nära samarbete med utvecklare, och genom en benchmarking, tagit fram vårt ramverk anser vi att vi har varit flexibla i metautvecklingen. Den typ av metautveckling vi har bedrivit är en situationsmetautveckling, där vi har försökt anpassa vårt ramverk för en viss utvecklingssituation (det vill säga utveckling av API:er).

8.4 Sammanfattning

I den validerande studien identifierade vi fyra förändringsåtgärder: Att tydligare visa att ramverket är iterativt, att tydligare visa i modellen var konsumentinvolvering sker, att införa bättre stöd för delleverans- och versionshantering, samt att införa ett nytt arbetsmoment som rör utvärdering av API:et efter leverans. Samtliga dessa har införts, och det har i ännu högre grad gjort att några av de tidigare kriterierna uppfylls, bland annat codesign och modellens enkelhet och tydlighet. Vi har också påvisat att ramverket även kan grundas i den teoretiska grund vi tagit fram.

9 SLUTDISKUSSION

I detta kapitel diskuteras hur vi har uppfyllt de frågeställningar vi tagit fram i början av studien, samt de slutsatser vi kommit fram. Vi utvärderar även studien utifrån de kriterier vi tagit fram. Kapitlet avslutas med en diskussion om fortsatt forskning.

I want the answers now or eventually!
- Homer Simpson

9.1 Svar på frågeställningar

9.1.1 Vilka aspekter är viktiga att tänka på vid utveckling av gränssnitt för exponering av tjänster?

Genom en kvalitativ intervjustudie och en benchmarking av de mest populära gränssnitten (API:erna) på webben har vi kommit fram till att man främst bör tänka på följande kriterier när man utvecklar ett gränssnitt för exponering av tjänster:

- Flexibilitet
- Outside in (tester)
- Riskanalys
- Prototyper
- Co-design
- Dokumentation
- Enkelhet och tydlighet

Det finns även vissa kriterier som ej är av lika stor vikt:

- Funktionalitet
- Användning av standardiserade tekniker
- Kompabilitet

För en utförligare diskussion kring kriterierna, se *Kriterier för ramverk*. Det ramverk vi har skapat fångar de viktigaste kriterierna, och kan därför med fördel användas för just utveckling av gränssnitt för exponering av tjänster. De kriterier som är av mindre vikt fångar ramverket genom co-design. Genom en validering av ramverket har även kvaliteten hos ramverket stärkts, och det har validerats att de kriterier som nämns ovan är viktiga. I vår validerande studie framkom även några andra kriterier, dessa kriterier rör dock främst själva utformningen av ramverket, och ej vad man bör tänka på när man utvecklar ett API.

9.1.2 Vilka aspekter är viktiga att tänka på vid återanvändning av tjänster från externa organisationer?

Vi har genom vår kvalitativa intervjustudie, som har sin utgångspunkt i projektet e-Me och utvecklingen av dess integrationsplattform, har vi kommit fram till att det viktigaste att tänka på när man återanvänder tjänster är att värdera riskera som detta skapar, samt att se till att verkligen sätta sig in i tjänsten. En risk kan till exempel vara att leverantörer ändrar i tjänsten, vilket kan leda till att konsumentens applikation fallerar, vilket enligt våra respondenter ibland händer. Det är som sagt också viktigt att sätta sig in i själva tjänsten, och lära sig hur den fungerar.

9.1.3 Vad finns det för risker med en integrationsplattform?

De risker som finns gäller framförallt säkerhet och integritet. Det är viktigt att se till att täta upp plattformen så att plugintjänster inte får tillgång till känsliga systemresurser, samt att se till att datan i plattformens datalager behåller sin integritet. Tjänster skall ej få läsa data de inte har rättighet till. Det kan dock vara svårt att ha en bra säkerhet och integritet, vilket bidrar till att risken blir högre.

9.1.4 Vilka mål fyller en integrationsplattform?

Eftersom denna studie främst gäller projektet e-Me, kan det vara svårt att ge något generellt svar på denna fråga. En integrationsplattform bör dock främst ha målet att aggregera tjänster, ut mot en slutanvändare, för att förenkla IT-sysslor för denna användare. Användaren kan vara en människa, eller ett annat informationssystem. I e-Mes fall är slutanvändaren en student, och plattformens mål är att hjälpa studenten med sysslor som är relaterade till dennes studiegång samt fritid.

9.1.5 Vilka behov fyller en integrationsplattform?

Ett av de största behoven som en integrationsplattform fyller, och som vi har identifierat, är flexibilitet. En integrationsplattform möjliggör detta genom att skapa en grund som man sedan utveckla vidare för, och därigenom skapa i stort sett vilka informationssystem som helst. Plattformen e-Me kan enligt utvecklarna göra just detta.

9.1.6 Vad finns det för några orsaker till att man utvecklar en integrationsplattform?

Orsaken till att man utvecklar en integrationsplattform är egentligen att det finns ett behov av den, men även att den skall uppnå någon sorts mål. I fallet med e-Me är behovet främst flexibilitet, för att därigenom uppnå målet att kunna förenkla studenternas vardag.

9.1.7 Vad är en integrationsplattform?

En integrationsplattform är en plattform som möjliggör aggregering av tjänster ut mot en slutanvändare. Denna slutanvändare kan vara en mänsklig användare eller ett annat informationssystem.

9.1.8 Hur ser utvecklingen för en integrationsplattform ut?

Det empiriska materialet från studien e-Me visar att utvecklingen startar baserat på någon sorts mål eller behov, och därifrån går vidare. Därefter görs en översiktlig systemarkitekturskiss, som visar hur den funktionalitet (sprungen ur behoven och målen) som plattfor-

men skall innehålla skall implementeras. Efter detta börjar implementationen, och API:erna bör här tas fram, baserat på de kriterier vi i denna studie tagit fram. För att ta fram API:erna kan man använda SOAPIF, då detta ramverk är baserat på dessa kriterier. Det är också viktigt att tänka på sådana fenomen som co-design och flexibilitet under utvecklingsprocessen.

9.2 Sammanfattande slutsatser

När man utvecklar ett API för exponering av tjänster bör man främst tänka på att API:et skall vara flexibelt och snabbt reagera på ändrade förändringar hos konsumenten, och att det är oerhört viktigt att testa att ramverket uppfyller konsumentens förväntningar. Man måste även jobba med riskanalys, så att konsumentens förväntningar inte riskerar säkerheten eller integriteten hos det som exponeras.

För att säkerställa att konsumenten snabbt kan komma igång med API:et, är utförlig och lättillgänglig dokumentation av yttersta vikt. För att kunden tidigt kan testa API:et, och reagera på det, är det viktigt att jobba mycket med co-design, bland annat genom prototyper. Kompabilitet och användning av standardiserade tekniker är andra faktorer som är viktiga att tänka på, men inte alltid nödvändiga.

Utvecklingen av en integrationsplattform börjar med en orsak, som är behov och mål. Dessa behov och mål är oftast flexibilitet, samt att underlätta för slutanvändaren av själva plattformen, ofta genom att aggregera information.

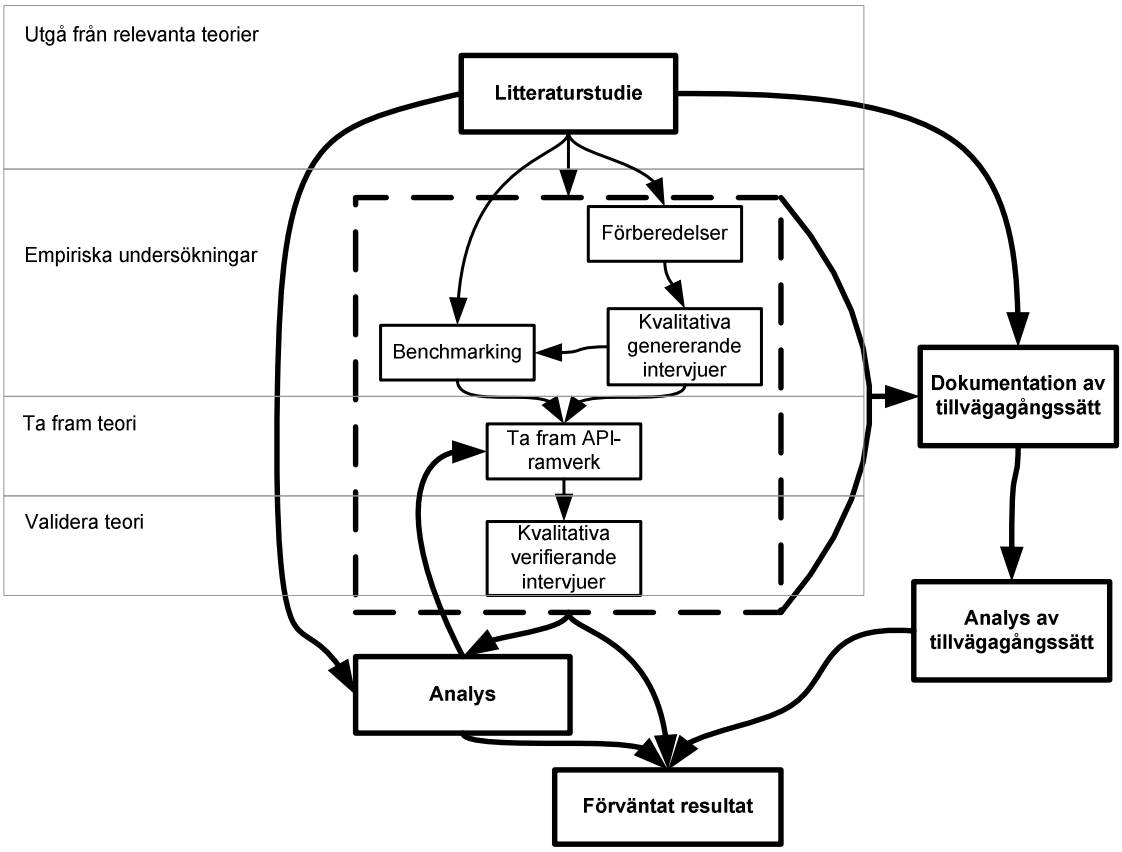
När man återanvänder tjänster från externa organisationer är det viktigt att tänka på att den tjänst återanvänder bör vara väldokumenterad, så att det är lätt att komma igång med den. Det är även viktigt att se över vad som kan hända med sin egen applikation om tjänsten på något sätt uppdateras.

9.3 Utvärdering av studien

Vi ämnar först utvärdera studiens forskningsstrategi, och sedan det resultat som skapats med hjälp av denna strategi.

9.3.1 Utvärdering av studiens forskningsstrategi

I detta kapitel utvärderar vi det vetenskapliga förhållningssätt och de metoder vi har använt under arbetet med denna studie. Dessa förhållningssätt och metoder diskuteras i *2 Vetenskapligt förhållningssätt och metod*. Vi återvänder därför till den strategi vi ämnat följa, vilken visas i *Figur 17: Återbesök till studiens forskningsstrategi*.



Figur 17: Återbesök till studiens forskningsstrategi

I vår litteraturstudie tog vi fram relevant teori och material som låg tillgrund för vår empiriska undersökning. Detta visade sig vara relativt enkelt att hitta teori om systemutvecklingsmetoder, SOA och metoden SOMA. SOMF var lite värre då det inte är så många författare som har skrivit om det, men på det stora hela tycker vi att vi fick ihop en god grund för vår studie. Vi använde oss av källkritik under studien, genom att följa de kriterier vi satt upp för just detta. Dessa kriterier anser vi passade bra, då de tog upp viktiga begrepp såsom källans ursprung och sammanhang.

Efter vår litteraturstudie tog vi fram en intervjuguide (baserad på teorin) och material om SOMA och SOMF som förberedelser inför intervjuerna. Detta material skickade vi till våra respondenter före intervjuerna. Vi tycker att vi fick ut mycket mer av intervjun då vi gjorde på detta sätt. Utvecklarna kunde på så sätt vara mer insatta i ämnet och vi slapp spendera mycket tid på att förklara metoden SOMA och ramverket SOMF för dem. Vi tycker att detta var ett bra sätt att utföra intervjuer på och tack vare att vår intervjuguide inte var för mycket inrutad hade vi mycket frihet att följa utvecklarnas tankebanor utan att tappa fokus, vilket syns i de olika systemutvecklarnas kommentarer. Inför våra intervjuer tänkte vi också igenom de kriterier som finns för kvalitativa intervjuer: Vi valde till exempel att hålla intervjun på en för utvecklarna bekant plats, för att de skulle känna sig trygga i sin roll som respondent. Vi valde även att spela in intervjuerna för att underlätta

transkriberingen, och vi inledde varje intervju med bakgrundsinformation om utvecklarna, för att sätta respondenternas svar i ett sammanhang. Vi anser att de kriterier vi tog fram för intervjuerna var väldigt bra, och lätt kunde appliceras på situationen.

För att få ut så bra kriterier som möjligt använde vi oss av vår intervjustudie som grund för benchmarkingen, och tog fram kriterier för benchmarkingen dels ur denna intervjustudie, samt ur teorin. Vi tycker även att vi valde bra objekt att studera då vi utgick från en lista med de mest använda API:erna just nu. Detta var första gången vi utförde någon slags benchmarking, men vi tycker ändå att vi har lyckats ganska bra och har följt det tillvägagångssätt som vi beskrivit i metodkapitlet. Vi tycker att de empiriska undersökningar vi gjort har gett oss bra kriterier och mycket material att arbeta med då vi tog fram vårt ramverk, och genom de här kriterierna har det varit enkelt att ta fram ett ramverk som innehåller allt som utvecklarna önskar, och det som kommit fram från benchmarkingen.

Vi känner själva att efter vi gjorde vår validering av ramverket att utvecklarna var nöjda med resultatet men att de även hade väldigt bra synpunkter som vi har tagit till oss. Dessa synpunkter och ändringar tycker vi framgår väldigt tydligt i vår uppdatering av ramverket. Vi hade från början planerat att intervjua alla systemutvecklarna en gång till för valideringen, men detta var dock inte möjligt på grund av vissa förhinder. De fick ändå ta del av ramverket och komma med synpunkter och kommentarer. Detta tycker vi ändå var bra då de utvecklare som fick ta till sig ramverket på detta sätt fick ett helt annat synsätt och därför gav oss andra typer av kommentarer än den som vi intervjuade. Vi anser därför att den validerande intervjustudien ändå gick bra, och att ramverket efter valideringen var tillräckligt bra för testanvändning.

Vi anser att den abduktiva forskningsstrategi vi valt passade oerhört väl in på vår studies genomförande. Vi har tagit fram en hypotes (induktion), baserad på ett teoretiskt och empiriskt material, för att sedan validera denna genom mer empiriskt material (deduktion). Dessa två utgör abduktion, vilket vi alltså har arbetar efter i vår studie.

De metoder och tillvägagångssätt som vi har valt anser vi var de rätta för ändamålet, och genom användandet av dessa metoder har vi nått fram till ett unikt forskningsbidrag.

9.3.2 Utvärdering av studiens resultat

I utvärderingen av denna studies resultat kommer vi att använda Goldkuhls tolv generella utvärderingskriterier (se *2.8 Utvärderingsmetod*). Det första av Goldkuhls kriterier är nyfikenhet; de som utför studien måste ha ett engagemang och en vilja att verkligen studera det fenomen som studien behandlar. Vi anser att vi verkligen har uppfyllt detta kriterium, då vi fullhjärtat gått in i denna långa resa med en önskan om att verkligen tillföra något till forskningen, och att faktiskt göra detta tillsammans med personer som kan ha nytta av det. Detta har varit vår stora drivkraft under studien, och har bidragit till att vi är där vi är nu.

När det gäller tydlighet, så har vi klart och tydligt visat när vi själva anser något, eller när våra respondenter anser något. Vi har också tydligt visat hur studien har genomförts, bland annat genom den forskningsstrategi vi lagt upp i metodkapitlet (*2.4 Strategi*).

Det är viktigt att man i en studie inte förskönar eller favoriserar antaganden eller teorier som man själv gynnas av, och att man är ärlig när det gäller resultatet av en studie. Vi anser att vår validering av vårt ramverk klart och tydligt visar att vi har tagit hänsyn till just ärlighet, då vi även tar upp negativa aspekter av vårt ramverk där. Det har också funnits en risk, då en av författarna arbetar som systemutvecklare på samma arbetsplats som respondenterna. Genom valideringen av ramverket, samt den andra författarens objektivitet, har detta dock hanterats på ett bra sätt.

Ett av de kriterier som vi har tagit till oss mest, är öppenhet för nya hypoteser och teorier. Genom den validering av ramverket vi genomfört, samt den benchmarking och den kvalitativa studie som ledde fram till ramverket, har vi klart och tydligt visat oss öppna för nya tankar och idéer, och även implementerat dessa i ramverket, i två iterationer. Detta hör även ihop med kriteriet kontext: Att det är viktigt att kunna ändra synsätt på det som studeras. Detta har vi gjort, som en konsekvens av den teoretiska och empiriska studie vi har genomfört.

Vi har genomfört denna studie på ett så noggrant sätt som möjligt, och verkligen försökt dokumentera alla de beslut vi tagit, och baserat dessa på en god grund. Som hjälp för detta har vi en väldefinierad forskningsstrategi, där det klar och tydligt framgår hur vi ämnade arbeta. Detta leder oss även in på kriteriet rationalitet, som vi anser att vi har uppfyllt, då alla de beslut som tagits är grundade på teoretiskt eller empiriskt material.

Vi har även varit nyskapande, och inte varit rädda för att ge oss in på outforskade områden. Vi har varit kreativa i framtagandet av vårt ramverk, och verkligen skapat något nytt och unikt. Vi har även validerat detta ramverk, och genom denna validering fått ett kvitto på att det ramverket faktiskt skulle kunna tillämpas, och att vi på rätt sätt har tagit tillvara på den input vi har fått. Vi tar även ansvar för den studie vi har gjort, och är beredda på att svara på önskemål om förändringar.

Det ramverk, och den kunskap vi har genererat, anser vi är relevant för ämnesområdet informatik, vilket vi även påvisat i *1 Inledning*. Vi anser också att vi med hjälp av denna uppsats klart och tydligt tillgängliggör vår studie, och att den är lätt att följa från sin start till sitt slut. Vi anser också att vårt resultat (främst vårt ramverk) är lättförståeligt, då vi har validerat detta genom intervjuer.

Genom att genomföra utvärdering, och att vi har utvärderat under studiens gång (vilket visas i *2.4 Strategi*), anser vi också att vi klart och tydligt har visat att vi har uppfyllt kriteriet reflektion, och kritiskt granskat vår egen studie, samt andra forskares bidrag inom kunskapsområdet.

9.4 Förslag på fortsatt forskning

I denna studie har vi tagit fram ett ramverk för utveckling av API:er för exponering av tjänster. Det skulle vara intressant att se detta ramverk använt i praktiken, och ytterligare validera dess kvalitet. En sådan studie skulle till exempel kunna syfta till att applicera detta ramverk på ett riktigt fall, och därigenom ytterligare validera ramverket och eventuellt vidareutveckla detta.

Det skulle också kunna vara intressant att göra om samma studie, fast med andra forskningsobjekt, och se om den studien når samma resultat, och på detta sätt ytterligare bidra till ramverket och öka dess kvalitet och generaliserbarhet.

Det skulle också vara intressant att göra en uppföljning av projektet e-Me, och se hur utveckligen av API:erna har gått, kanske även med feedback från konsumenterna av API:et. Det är också intressant att utvecklarna nämner att e-Me egentligen skulle kunna användas till vad som helst: Det skulle vara intressant att undersöka om detta är möjligt.

Ett ytterligare uppslag till eventuell framtida forskning är att undersöka hur förändringar i ett API eller en tjänst påverkar konsumenterna, och om det är möjligt att göra dessa uppmärksamma på förändringar av till exempel ett kontrakt för ett API. Detta har varit ett problem som de utvecklare vi har intervjuat har stött på.

När det gäller utveckling av informationssystem så är det viktigt med co-design: Hur tar man vara på kundens eller slutanvändarens önskemål och förväntningar på systemet? Vi tycker att det finns relativt bra litteratur och forskning om detta, dock saknar vi mer forskning om hur dessa önskemål och förväntningar når själva utvecklaren av systemet, och vi har i tidigare artiklar även sett att andra rätt så stora företag har problem med detta.

Utvecklarna i e-Meprojektet var överens om att säkerhet är en väldigt viktig aspekt att tänka på vid utveckling av tjänster och API:er. Vi tycker därför att det skulle vara intressant med vidare forskning som berör just detta: Hur påverkas säkerheten genom exponering av funktionalitet genom API:er eller tjänster?

KÄLLFÖRTECKNING

Adobe (2009). *Adobe Flash Player*. (Elektronisk) Tillgänglig: <http://www.adobe.com/products/flashplayer/> (2009-05-15)

Albinsson, Lars, Forsgren, Olov & Lind, Mikael (2008). *Towards a Co-Design Approach for Open Innovation*. (Elektronisk) Tillgänglig: <http://www.adm.hb.se/~ML/pdf-filer/2008_PDC-LAL-OFO-ML.pdf> (2009-05-17)

Amazon (2008). *Amazon Associates Web Service*. (Elektronisk) Tillgänglig: <http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1662&categoryID=17> (2009-05-15)

Amazon (2009). *Product Advertising API*. (Elektronisk) Tillgänglig: <https://affiliate-progrm.amazon.com/gp/advertising/api/detail/main.html?ie=UTF8&pf_rd_t=501&pf_rd_m=ATVPDKIKX0DER&pf_rd_p=&pf_rd_s=assoc-right-1&pf_rd_r=&pf_rd_i=assoc_join_menu_paapi> (2009-05-15)

Andersen, Bjørn & Pettersen, Per-Gaute (1997). *Benchmarking – En praktisk handbok*. Lund: Utbildningshuset Studentlitteratur.

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S. & Holley, K. (2008). *SOMA: A method for developing service-oriented solutions*. IBM Systems Journal, vol. 47, nr. 3.

Avison, D. & Fitzgerald, G. (2003). *Information Systems Development*. Maidenhead: McGraw-Hill Education.

Axelsson, Karin (2007). *Informationssystemsarkitektur ur ett verksamhetsperspektiv*. (Elektronisk) Tillgänglig: <http://www.iei.liu.se/content/1/c6/08/83/61/Fo_AoO07_KA.pdf> (2009-05-11)

Axelsson, Karin & Goldkuhl, Göran (1998). *Strukturering av informationssystem*. Lund: Studentlitteratur.

Beck, Kent (2003). *Test-Driven Development By Example*. Indianapolis: Addison-Wesley Professional.

Bell, Michael (2008). *Service-Oriented Modeling – Service Analysis, Design, and Architecture*. Hoboken, NJ: John Wiley and Sons.

Berners-Lee, Tim (1999). *Glossary*. (Elektronisk) Tillgänglig: <http://www.innovatia.com/software/papers/com.htm> (2009-05-15)

Beynon-Davies, Paul (2002). *Information Systems – An Introduction to Informatics in Organizations*. Basingstoke: Palgrave.

Brinkkemper, Sjaak, Lyytinen, Kalle & Welke, Richard J. (1996). *Method Engineering – Principles of method construction and tool support*. Dordrecht: Chapman & Hall.

Bryman, Alan (2002). *Samhällsvetenskapliga metoder*. Malmö: Liber.

Cockburn, Alistair (2000). *Agile Software Development*. (Elektronisk) Tillgänglig: <http://www.imamu.edu.sa/Scientific_selections/Documents/IT/AgileSwDevDraft3.pdf> (2009-05-13)

Corbin, Juliet & Strauss, Anselm (2008). *Basics of Qualitative Research*. Tredje upplagan. Thousand Oaks: Sage Publications, Inc.

Davis, Mark (1999). *Forms of Unicode*. (Elektronisk) Tillgänglig: <http://www.icu-project.org/docs/papers/forms_of_unicode/> (2009-05-15)
Denscombe, Martin (2009). *Forskningshandboken – för småskaliga forskningsprojekt inom samhällsvetenskaperna*. Lund: Studentlitteratur.
Dylan, Bob (1964). *The Times They Are a-Changing*. Från albumet *The Times They Are a-Changing*. New York City: Columbia.
e-Me (2009). *Welcome to the e-Me collaboration platform!* (Elektronisk) Tillgänglig: <http://drax.ida.hb.se/eme-insite/> (2009-02-17)
Eriksson, Owen, Goldkuhl, Göran, Axelsson, Karin & Hultgren, Göran (1997). *Informationsteknik som stöd för transport, resande och inter-organisatorisk samverkan*. (Elektronisk) Tillgänglig: <http://users.du.se/~oer/forskning/rapporter/erfl97arap.PDF> (2009-05-11)
Erl, Thomas (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River: Pearson Education.
Facebook (2009a). *Facebook*. (Elektronisk) Tillgänglig: <http://www.facebook.com/facebook> (2009-04-01)
Facebook (2009b). *Facebook Developers*. (Elektronisk) Tillgänglig: <http://developers.facebook.com/> (2009-04-01)
Flickr (2009a). *About Flickr*. (Elektronisk) Tillgänglig: <http://www.flickr.com/about/> (2009-05-15)
Flickr (2009b). *Flickr Services*. (Elektronisk) Tillgänglig: <http://www.flickr.com/services/api/> (2009-05-15)
Funcke, D. (2005). *Grounded Theory*. (Elektronisk) Tillgänglig: <http://dtserv1.compsy.uni-jena.de/ws2005/mikrosoz_uj/50955644/content.nsf/Pages/EC5515D3BE4C9D3EC12570D70023F9F2/$FILE/Handout_14122005.pdf> (2009-06-15)
Goldkuhl, Göran (1998). *Kunskapande*. (Elektronisk) Tillgänglig: <http://www.vits.org/publikationer/dokument/409.pdf> (2009-03-19)
Google (2009a). *Explore Google Maps*. (Elektronisk) Tillgänglig: <http://maps.google.se/help/maps/tour/> (2009-05-15)
Google (2009b). *Google Maps API Concepts*. (Elektronisk) Tillgänglig: <http://code.google.com/intl/sv-SE/apis/maps/documentation/index.html > (2009-05-15)
Google (2009c). *Google Maps API Reference*. (Elektronisk) Tillgänglig: <http://code.google.com/intl/sv-SE/apis/maps/documentation/reference.html > (2009-05-15)
Google (2009d). *YouTube APIs and Tools*. (Elektronisk) Tillgänglig: <http://code.google.com/intl/sv-SE/apis/youtube/overview.html> (2009-05-15)
Highsmith, Jim & Cockburn, Alistair (2001). *Agile Software Development: The Business of Innovation*. (Elektronisk) Tillgänglig: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=947100> (2009-05-13)
von Hippel, Eric & Katz, Ralph (2002). *Shifting Innovation to Users via Toolkits*. Management Science. Vol. 48, nr 7.

Högskolan i Borås (2002). *Innebörden av informatik vid IDA/HiB*. (Elektronisk) Tillgänglig: <http://www.hb.se/ida/attachments/Informatik%20020925.pdf> (2008-03-04)
InnovationLab (2009). *InnovationLab – Högskolan i Borås*. (Elektronisk) Tillgänglig: <www.innovationlab.se> (2009-05-17)
JSolutions (2007). *UTF-8?* (Elektronisk) Tillgänglig: <http://jsolutions.se/?p=114> (2009-05-15)
JSON (2009). *JSON*. (Elektronisk) Tillgänglig: <http://www.json.org/> (2009-05-15)
Krohn, Andreas (2008). *Vad är en mashup?* (Elektronisk) Tillgänglig: <http://www.mashup.se/ommashups/vad-ar-en-mashup> (2009-05-15)
Le Duc, Michaël (2007). *Induktion, deduktion och abduktion*. (Elektronisk) Tillgänglig: <http://www.eki.mdh.se/personal/mlc01/metod_0_7/Induktion,deduktionochabduktion.html> (2009-03-22)
Leth, Göran & Thurén, Torsten (2000). *Källkritik för Internet*. (Elektronisk) Tillgänglig: <http://www.psycdef.se/Global/PDF/Publikationer/kallkritid%20for%20internet.pdf> (2009-05-17)
Lind, Mikael, Albinsson, Lars, Forsgren, Olov & Hedman, Jonas. *Integrated Development, Use and Learning in a Co-design Setting: Experiences from the Incremental Deployment of e-Me*. (Elektronisk) Tillgänglig: <http://www.calistoga.se/2007_eChallenges_ML-LAL-OFO-JHE.pdf> (2009-05-17)
Lindell, Staffan, Lind, Mikael & Forsgren, Olov (2006). *Students as e-Citizens - Deriving Future Needs of e-Services for Students*. (Elektronisk) Tillgänglig: <http://www.e-Me.se/images/PDF-er/WESPA-Lindell_Lind_Forsgren.pdf> (2009-05-17)
Mackie, Kurt (2007). *SOA Synergy and Obstacles*. (Elektronisk) Tillgänglig: <http://www.adtmag.com/article.aspx?id=20719> (2009-03-03)
Mountain Goat Software (2008). *Supporting Material: Scrum Overview*. (Elektronisk) Tillgänglig: <http://epf.eclipse.org/wikis/scrum/Scrum/guidances/supportingmaterials/scrum_overview_610E45C2.html> (2009-05-12)
Mozilla Foundation (2009). *About JavaScript*. (Elektronisk) Tillgänglig: <https://developer.mozilla.org/en/about_javascript> (2009-05-15)
Nordqvist, Carl-Johan (2006). *Introduktion till SOA*. (Elektronisk) Tillgänglig: <http://iwtjanster.idg.se/webbstudio/pub/artikel.asp?id=329> (2009-02-17)
O'Reilly Media (2005). *An Introduction to Service-Oriented Architecture from a Java Developer Perspective*. (Elektronisk) Tillgänglig: <http://www.onjava.com/pub/a/onjava/2005/01/26/soa-intro.html> (2009-05-11)
O'Reilly, Tim (2005). *What Is Web 2.0*. (Elektronisk) Tillgänglig: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html> (2009-05-15
Oosterholt, Ron, Kusano, Mieko & de Vries, Govert (1996). *Interaction design and human factors support in the development of a personal communicator for children*. (Elektronisk) Tillgänglig:

<http://sigchi.org/chi96/proceedings/desbrief/Oosterholt/rho_txt.htm> (2009-05-17)
The PHP Group (2009). *PHP: Hypertext Preprocessor*. (Elektronisk) Tillgänglig: <http://www.php.net/> (2009-05-15)
Princeton (2009). *Wordnet Search: Widget*. (Elektronisk) Tillgänglig: <http://wordnet.princeton.edu/perl/webwn/> (2009-05-15)
ProgrammableWeb (2009). *API Directory*. (Elektronisk) Tillgänglig: <http://www.programmableweb.com/apis/directory/1?sort=mashups> (2009-05-15)
Running, Jordan (2007). *Understanding Basic Internet Jargon*. (Elektronisk) Tillgänglig: <http://www.tucows.com/article/1903> (2009-05-15)
Schwaber, Ken (1997). *SCRUM Development Process*. (Elektronisk) Tillgänglig: <http://www.jeffsutherland.com/oopsla/schwapub.pdf> (2009-05-07)
Schwaber, Ken (2007). *What Is Scrum?*. (Elektronisk) Tillgänglig: < http://volaroint.com/file/DC-%20VOLARO-%20Training-Scrum-What_Is_Scrum.pdf> (2009-05-07)
Two Crows (2009). *Data Mining Glossary*. (Elektronisk) Tillgänglig: <http://www.twocrows.com/glossary.htm> (2009-04-01)
Tyagi, Sameer (2006). *RESTful Web Services*. (Elektronisk) Tillgänglig: <http://java.sun.com/developer/technicalArticles/WebServices/restful/> (2009-05-15)
UTF-8 (2008). *UTF-8 and Unicode*. (Elektronisk) Tillgänglig: <http://www.utf-8.com/> (2009-05-15)
W3C (1998). *VML – the Vector Markup Language*. (Elektronisk) Tillgänglig: <http://www.w3.org/TR/NOTE-VML> (2009-05-15)
W3C (2001). *XML i 10 punkter*. (Elektronisk) Tillgänglig: <http://www.w3c.se/resources/office/translations/XML-in-10-points_sw.html> (2009-05-15)
W3C (2002a). *Web Services Architecture Usage Scenarios*. (Elektronisk) Tillgänglig: <http://www.w3.org/TR/2002/WD-ws-arch-scenarios-20020730/> (2009-05-15)
W3C (2002b). *XHTML 1.0: The Extensible Hypertext Markup Language (Second Edition)*. (Elektronisk) Tillgänglig: <http://www.w3.org/TR/xhtml1/> (2009-05-15)
Wake, William C. (2004). *Scrum Development Process*. (Elektronisk) Tillgänglig: <http://xp123.com/xplor/xp0401/Scrum-dev.pdf> (2009-05-07)
Wikipedia (2009). *Service-Oriented Modeling*. (Elektronisk) Tillgänglig: <http://en.wikipedia.org/wiki/Service-oriented_modeling> (2009-02-23)
Wipcore (2008). *Integrationsplattform*. (Elektronisk) Tillgänglig: <http://www.wipcore.se/ehandel/integrationsplattform/> (2009-02-17)
YouTube (2009). *Företagshistorik*. (Elektronisk) Tillgänglig: <http://www.youtube.com/t/about> (2009-05-15)
Zimmermann, Olaf, Krogdahl, Pal & Gee, Clive (2004). *Elements of Service-Oriented Analysis and Design*. (Elektronisk) Tillgänglig: <http://www.ibm.com/developerworks/webservices/library/ws-soad1/> (2009-02-23)

BILAGOR

Bilaga 1: Intervjuguide 1

RESPONDENT

NAMN: _____

POSITION: _____

ORGANISATION: _____

DATUM: _____

INTERVJU INSPELAD ___JA ___NEJ

VILL VARA ANONYM ___JA ___NEJ

INTERVJUAD AV 1 _____

2 _____

ANTECKNINGAR

AGENDA

PRESENTATION AV FORSKARNA OCH STUDIEN	1. Kort presentation av oss själva 2. Syftet med studien 3. Studiens forskningsfrågor
STRUKTUR FÖR SJÄLVA INTERVJUN	1. Bakgrundsinformation 2. Modellering, design och implementation av SOA i E-Me med fokus på API:er 3. Avslutande frågor
VÅRA DEFINITIONER AV VIKTIGA BEGREPP	1. SOA 2. Integrationsplattform 3. API 4. Ramverk

BAKGRUNDSINFORMATION

Börja med att berätta lite om dig själv: Vad arbetar du med? Vad har du för syn på systemutveckling och SOA? (5 minuter)

Skulle du kunna ge en kort beskrivning av hur du uppfattar projektet E-Me, och vad din roll i projektet är? (5 minuter)

- Tjänster
  - Vilken typ?
  - Hur många?
  - Varför har ni valt att ha tjänster?
    - Mål, behov, risker, orsaker
- Leverantörer
  - Vilka/vilken typ?
  - Hur många?
- Vilka produkter har använts i E-Me?

MODELLERING, DESIGN OCH IMPLEMENTATION AV SOA I E-ME MED FOKUS PÅ API:ER

Kan du ge en kort beskrivning av hur SOA-miljön i E-Me ser ut? (5 min)

- Vad är enligt dig den största utmaningen med SOA?
- Vad tycker du är det viktigaste att tänka på vid en SOA?

Hur hanteras tjänsterna i E-Me? (10 min)

- Hanterar ni både interna och externa tjänster?
  - Görs det i så fall någon skillnad på dessa?
  - Hur hanteras dem?
- Återanvänder E-Me några tjänster?
  - Om ja, vilka?
  - Om nej, varför inte?
  - Om delvis, vilka? Varför delvis?
- Har några tjänster implementerats än?
  - Vilka?
  - Hur har det gått?
    - Problem?
    - Vad har gått bra?
  - Interna/externa?
    - Någon skillnad?

MODELLERING, DESIGN OCH IMPLEMENTATION AV SOA I E-ME MED FOKUS PÅ API:ER

Hur ser gränssnittet för tjänsterna ut? (10 min)

- Skillnad på interna/externa?
  - Vilken skillnad?
  - Varför krävs det olika?
- Problem med gränssnittet?
- Fördelar med gränssnittet?
- Kan tjänsterna kommunicera med varandra genom gränssnittet?
  - I så fall, hur?
  - Skillnad på interna/externa?
- Har ni något kontrakt för gränssnittet?

Är du bekant med SOMA? (10 min)

Pre-condition: Respondenterna har förberetts med SOMA-dokumentation.

- Hur anser du att er modellering/design av E-Me passar in i SOMA?
  - Utgå från modellen
- Vad anser du om denna metod?
  - Fördelar
  - Nackdelar
  - Något som saknas?

MODELLERING, DESIGN OCH IMPLEMENTATION AV SOA I E-ME MED FOKUS PÅ API:ER

Är du bekant med SOMF? (10 min)

Pre-condition: Respondenterna har förberetts med SOMF-dokumentation.

- Hur anser du att er modellering/design av E-Me passar in i SOMF?
  - Utgå från modellen
- Vad anser du om detta ramverk?
  - Fördelar?
  - Nackdelar?
  - Något som saknas?

Av de erfarenheter du har fått genom arbetet med E-Me + SOA, vilka du lyfta fram? (5 min)

- Problem?
- Något som har gått bra?

AVSLUTANDE FRÅGOR

Tack för att du har tagit dig tid att svara på våra frågor! (5 min)

Om vi har ytterligare frågor eller oklarheter, är det ok om vi kontaktar dig då? ___ JA ___ NEJ

Vill du läsa igenom och godkänna vår sammanställning av denna intervju före publicering? ___ JA ___ NEJ

Bilaga 2: Intervjuguide 2

RESPONDENT

NAMN: .....

POSITION: .....

ORGANISATION: .....

DATUM: _____

INTERVJU INSPELAD ___ JA ___ NEJ

VILL VARA ANONYM ___ JA ___ NEJ

INTERVJUAD AV 1 _____

2 .....

ANTECKNINGAR

AGENDA

RAMVERKET

1. Förklaring (om nödvändigt)
2. Frågor om ramverket
3. Frågor om paketering av ramverket

RAMVERKET

Vad tycker du om SOAPIF?

- Kan du följa flödet?
- Är det enkelt att förstå?
- Är det tydligt?
- Är det något du anser saknas?
- Är det något du vill ändra eller ta bort?

Anser du att detta ramverk hade kunnat appliceras på E-me?

- Om ja/nej, varför?
- På vilket sätt?

MODELLERING, DESIGN OCH IMPLEMENTATION AV SOA I E-ME MED FOKUS PÅ API:ER

Hur brukar du vanligtvis komma i kontakt med nya ramverk/metoder/liknande?

Vad tycker du om paketeringen av själva ramverket?

- Förslag?

Högskolan i Borås är en modern högskola mitt i city. Vi bedriver utbildningar inom ekonomi och informatik, biblioteks- och informationsvetenskap, mode och textil, beteendevetenskap och lärarutbildning, teknik samt vårdvetenskap.

På **institutionen för data- och affärsvetenskap (IDA)** har vi tagit fasta på studenternas framtida behov. Därför har vi skapat utbildningar där anställningsbarhet är ett nyckelord. Ämnesintegration, helhet och sammanhang är andra viktiga begrepp. På institutionen råder en närhet, såväl mellan studenter och lärare som mellan företag och utbildning.

Våra **ekonomiutbildningar** ger studenterna möjlighet att lära sig mer om olika företag och förvaltningar och hur styrning och organisering av dessa verksamheter sker. De får även lära sig om samhällsutveckling och om organisationers anpassning till omvärlden. De får möjlighet att förbättra sin förmåga att analysera, utveckla och styra verksamheter, oavsett om de vill ägna sig åt revision, administration eller marknadsföring. Bland våra **IT-utbildningar** finns alltid något för dem som vill designa framtidens IT-baserade kommunikationslösningar, som vill analysera behov av och krav på organisationers information för att designa deras innehållsstrukturer, bedriva integrerad IT- och affärsutveckling, utveckla sin förmåga att analysera och designa verksamheter eller inrikta sig mot programmering och utveckling för god IT-användning i företag och organisationer.

Forskningsverksamheten vid institutionen är såväl professions- som design- och utvecklingsinriktad. Den övergripande forskningsprofilen för institutionen är handels- och tjänsteutveckling i vilken kunskaper och kompetenser inom såväl informatik som företagsekonomi utgör viktiga grundstenar. Forskningen är välrenommerad och fokuserar på inriktningarna affärsdesign och Co-design. Forskningen är också professionsorienterad, vilket bland annat tar sig uttryck i att forskningen i många fall bedrivs på aktionsforskningsbaserade grunder med företag och offentliga organisationer på lokal, nationell och internationell arena. Forskningens design och professionsinriktning manifesteras också i InnovationLab, som är institutionens och Högskolans enhet för forskningsstödjande systemutveckling.

HÖGSKOLAN I BORÅS
VETENSKAP FÖR PROFESSION

BESÖKSADRESS: JÄRNVÄGSGATAN 5 · POSTADRESS: ALLÉGATAN 1, 501 90 BORÅS
TFN: 033-435 40 00 · E-POST: INST.IDA@HB.SE · WEBB: WWW.HB.SE/IDA