Handelshögskolan
VID GÖTEBORGS UNIVERSITET
Institutionen för informatik
2003-03-11

UTVECKLING AV WEB SERVICES

– en komparativ analys av utvecklingsmetoder

Web service är en ny teknik för att utbyta information och tjänster över Internet som baseras på öppna standarder och protokoll. Web services syftar till att underlätta kommunikation och informationsutbyte oavsett teknisk plattform eller programspråk. Arbetet har haft som syfte att jämföra utveckling av web services med utveckling av traditionella system med vattenfallsmodellen som grund. Vilka skillnader finns vid utveckling av web services gentemot utveckling av applikationer? Fungerar vattenfallsmodellen för att beskriva systemutvecklingsprocessen kring web services eller behövs en ny modell? För att besvara den frågan har litteraturstudier om tekniken web services och systemutvecklingsmodeller gjorts samt en jämförelse mellan två fallstudier. Resultatet visar att vattenfallsmodellen är applicerbar som en beskrivningsmodell av web services men att tidsåtgången vid de olika stegen skiljer sig från traditionell systemutveckling.

Nyckelord: Web service, vattenfallsmodellen, systemutveckling

Författare: Jonas, Henriksson
Hans, Strandberg
Handledare: Johan, Magnusson
Examensarbete I, 10 poäng

Inledning

Bakgrund

Web services är ett nytt sätt att bygga it-baserade system vars angreppsätt har blivit mycket populärt under de senaste åren. Tekniken ger goda möjligheter att bygga system som kan kommunicera med klienter oberoende av plattform eller programspråk. Detta nya angreppsätt inom systemutveckling ger stora möjligheter att inkorporera befintliga system med nya vilket kan spara resurser vid utvecklingsarbetet.
Det är intressant att veta skillnaderna mellan traditionell systemutveckling och utveckling av web services. Är det nya sättet att utveckla mera resurseffektivt? Är det möjligt att använda befintliga beskrivningsmodeller exempelvis vattenfallsmodellen[1] eller måste en helt ny modell utvecklas?
Arbetet fokuserar på att pröva den utvecklingsmodell som normalt används vid traditionell systemutveckling mot utveckling av web services.

Syfte

Syftet med uppsatsen är att kartlägga skillnader mellan traditionell systemutveckling och utveckling av web services.

Problemformulering

Fungerar vattenfallsmodellen för att beskriva systemutvecklingsprocessen kring web services eller behövs en ny modell? Vilka skillnader finns vid utveckling av web services gentemot utveckling av applikationer?

Avgränsningar

Arbetets inriktning och problemformulering leder fram till ett antal avgränsningar som gjorts för att göra arbetet mer hanterbart. Dessa avgränsningar rör:

- Säkerhetsaspekter kring tekniken
- Ekonomiska förutsättningar för att tekniken ska bli framgångsrik
- Tänkta användningsområden

Prototypen som ligger till grund för en av fallstudierna blir begränsad i sin funktionalitet och felhantering varför ingen omfattande testning sker.
Slutsatserna kring vattenfalssmodellen begränsas till de fyra första stegen då fallstudierna som ligger till grund för resultaten ej omfattar drift och underhåll av ett färdigt system.

[1] Sommerville (2001)

Dokumentstandarder och konventioner

Eftersom många uttryck och termer som används i litteraturen härstammar från engelskan och då det ännu inte finns relevanta översättningar har vi i vissa fall använt de engelska termerna och i vissa fall gjort egna översättningar.
Det engelska uttrycket web service översätts inte för att undvika hopblandning med uttrycket webbtjänster som kan innebära andra lösningar än de som avses.

Webbadresser med hänvisningar till Internet skrivs som hela sökvägar och i det elektroniska dokumentet visas de som klickbara länkar.
Ex:
http://www.w3.org

Förkommande programkod blandas i vissa fall med den löpande texten och för den fullständiga prototypen som en bilaga i slutet av arbetet.

Metod

Metodval

Arbetet syftar till att jämföra traditionell systemutveckling med utveckling av web services ur perspektivet att vattenfallsmodellen används i båda fallen. Då teorier kring systemutveckling av web service-baserade system saknas fann det sig naturligt att skapa en prototyp för att kunna bilda sig en uppfattning baserad på praktisk kunskap. Denna kunskap kompletterades med teoretiska kunskaper baserade på litteraturstudier. För att skapa en bas för referenser till vår prototyp beskriver även arbetet en tidigare laborationsuppgift vid institutionen för informatik.[2] Dessa två systemutvecklingsprojekt beskrivs som fallstudier där resultat i form av tankar och känslor samt kvantitativa resultat redovisas.

Litteraturstudier

Litteraturstudier har använts för att inhämta kunskaper kring de begrepp och teorier arbetet baseras på. Denna metod kan användas för att ge en överblick kring det forskningen tidigare kommit fram till. Litteraturstudierna kan precisera och definiera de begrepp som arbetet hanterar.[3]

Fallbeskrivningar

Fallbeskrivningar har valts för att empiriskt undersöka problemområdet, dessa baseras på dels ett traditionellt systemutvecklingsprojekt och dels på utveckling av en prototyp av web service. Fallbeskrivningarna ligger som en grund för de resultat arbetet redovisar kring skillnader mellan utveckling av web services och traditionell systemutveckling. Valet av denna metod baseras på att web services är ett nytt angreppssätt varför nyttan av den mera traditionella intervjumetodiken skulle ge ett tveksamt och ur slutsatshänseende bristfälligt underlag. Fallstudier används vanligen när problemområdet är förhållandevis okänt eller då tidigare vetenskapliga arbeten inom området saknas. Den kan vara som

[2] Handelshögskolan i Göteborg
[3] Backman, (1998)

hjälpmedel för att generera en hypotes.[4] Syftet med att göra fallstudier är att försöka få en helhetsinriktad förståelse av det som studeras.[5]Backman säger I sin bok om uppsatsskrivning:

> ”Fallstudier anses vara särskilt tillämpliga I utvärderingar, där studieobjekten ofta är mycket komplexa. Man söker exempelvis förklara, förstå eller beskriva stora företeelser, organisationer eller system, som inte enkelt låter sig undersökas med annan metodik…”[6]

Fallbeskrivningar används även då det som ska undersökas är ett nytt fenomen. Man kan då följa Yins definition av en fallstudie:

> ” En empirisk studie som undersöker ett samtida fenomen i sin verkliga kontext, speciellt då gränsen mellan fenomenet och kontexten inte är tydlig. Studien baseras på flera olika källor, där datainsamling och analys vägleds av teoretiska antaganden.”[7]

Syftet med den typen av undersökningar är att inhämta så mycket kunskap som möjligt om intresseområdet.[8]

[4] Kjellén och Söderman, (1980)
[5] Merriam, (1994)
[6] Backman, (1998)
[7] Yin, (1994)
[8] Patel & Davidsson, (1994)

Teori

Web services

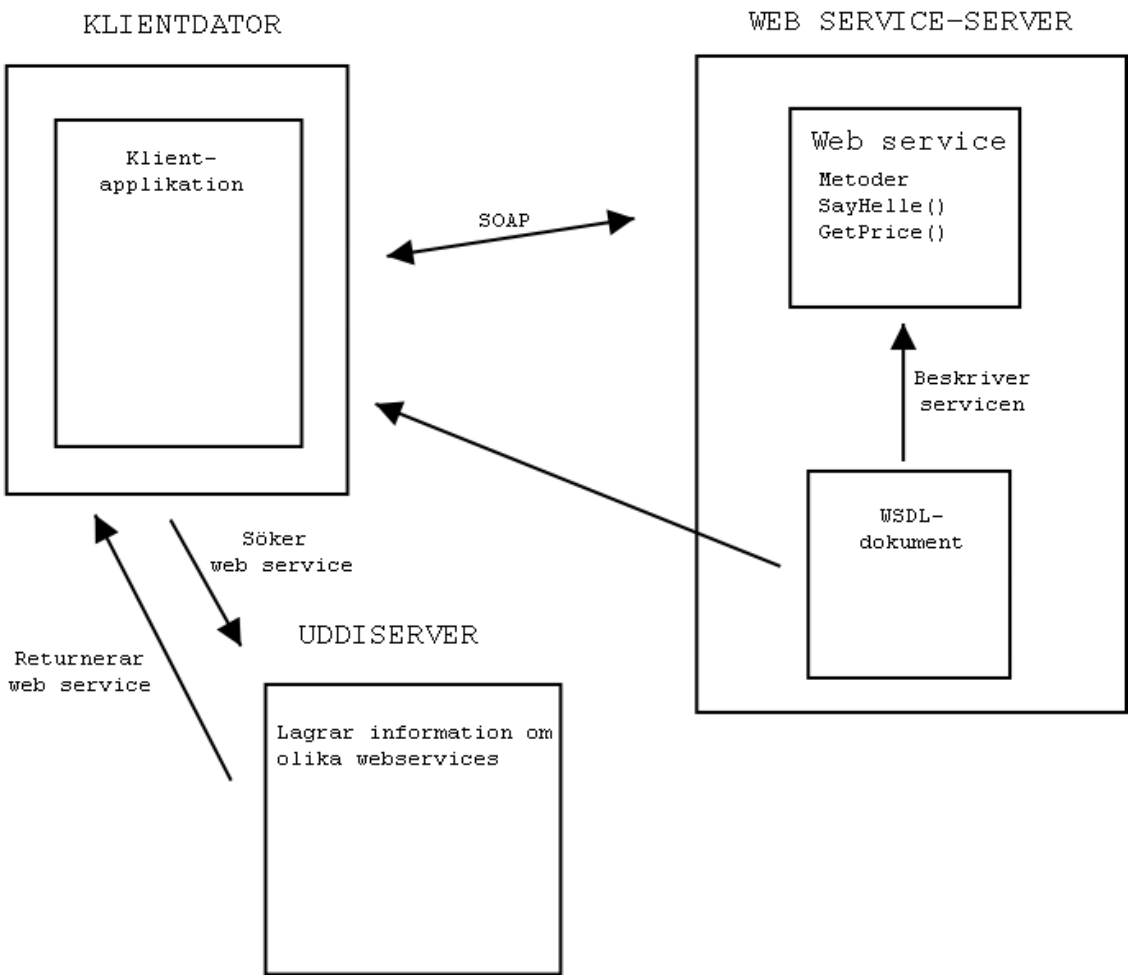
Begreppet web services är just nu mycket omtalat men få känner till vad det egentligen innebär eller vad de kan användas till.

Definitionerna av Web services är många men en av de enklaste gör Ethan Cerami[9] som säger att en web service är

> ”Ett program som görs tillgängligt på Internet och använder sig av standardiserad XML meddelandeöverföring”

Web service som begreppet används idag är någon form av gemensam uppfattning om vad som ska ingå, dock finns ingen fastställd standard över hur det ska byggas upp men det verkar finnas en förhållandevis enad bild över en minsta gemensamma nämnare för vad en web service är. En standard för web services håller på att utarbetas av ett konsortium av de stora aktörerna på marknaden.
Tills vidare används den minsta gemensamma nämnaren:
En web service är en applikation som kan nås via nätet och som kan ta emot metodanrop med tillhörande data från en klient som är intresserad av ett svar i någon form.[10] Servicen ska i sin tur kunna generera ett svar på anrop eller ett felmeddelande om anropet av olika anledningar inte kan utföras enligt förfrågan. De anrop och data som överförs mellan klienten och servicen skickas med det XML-baserade protokollet SOAP[11].
För att klienten ska veta hur servicen ska anropas och vilken typ av data som kan accepteras beskrivs servicen med hjälp av ett särskilt beskrivningsspråk Web Services Description Language WSDL[12]. Dessa dokument ger möjlighet att på ett automatiserat sätt skapa klienter som kan kommunicera med en web service eller för en utvecklare att förstå på vilket sätt en web service är uppbyggd. För att web servicarna ska kunna hittas av de som har behov av dem har man skapat UDDI-servrar som lagrar information om tillgängliga web servicear.[13]

[9] Cerami, E, (2002)
[10] Fensel, D & Bussler, C (2002)
[11] SOAP Se denna uppsats
[12] WSDL Se denna uppsats.
Specifikation: http://www.w3.org/TR/wsdl
[13] Se figur 1



Figur 1 Struktur över hur web services är uppbyggt.

Extensible Markup Language (XML)

XML är ett språk skapat för att beskriva och transportera data, XML är inte designat för att göra någonting med den utan kan ses som en behållare där man stoppar in data. Språket är uppbyggt med hjälp av taggar som beskriver datan mellan dem. Dessa taggar kan man själv definiera[14].

```
<komihåg>
  <till>Jonas</till>
  <rubrik>Födelsedag!</rubrik>
  <text>Glöm ej att Mamma fyller år idag</text>
</komihåg>
```

Detta exempel beskriver datan Jonas, Födelsedag och Glöm ej att Mamm … Men den gör inget med den, det krävs att någon skapar en applikation som hämtar datan från dokumentet och presenterar den för ”Jonas” så att han ej glömmer ”Mammas”

[14] http://www.w3schools.com/xml/xml_whatis.asp

födelsedag. Ett XML dokument är skrivet som ett vanligt text dokument och är därför mycket användbart för informationsutbyte mellan system och program. Den syntax man använder är enkel och självbeskrivande:

<?xml version="1.0" encoding="ISO-8859-1"?>
Detta ska vara den första raden i ett XML dokument, det är en beskrivning av vilken version av XML som används och vilken text uppsättning som används. Om man anger version 1.0 måste dokumentet följa standarden enligt W3C som finnes på http://www.w3.org/TR/2000/REC-xml-20001006 .

<komihåg> taggen beskriver själva basen i dokumentet, den säger att detta dokument är ett komihåg. Efter följer de taggar som beskriver själva komihåg elementet, <till>, <rubrik> och <text>. Avslutas med </komihåg> för att stänga handlingen. Alla taggar man skapar måste även stängas i den ordningsföljd man har skapat dem. Eftersom man kan definiera sina egna taggar i XML blir det ganska enkelt att läsa och tyda vad som sker även för en person som inte har kunskap om språket.

Ett XML meddelande måste starta med, efter XML deklarationen på första raden en bastagg som beskriver vad som kommer.

```
<bastagg>
  <nod>
    <subnod>.....</subnod>
  </nod>
</bastagg>
```

Det måste vara exakt ett bas element i varje dokument som inte har någon del i något annat underliggande element.

För att skicka ett attribut med ett värde går det att skriva <nod date="12/11/99"> med då tappar man lite av idén med XML. Attributvärden sätts bara i speciella fall som när ett anrop har ett id eller ett nummer ex <nod id=”635”>. Enkelt förklarat med att metadata(data om data) ska sparas som attributvärden och att själva datan(värdena) ska sparas som element. Om inte namn är en unik identifierare bör namnet Kalle skickas som på andra raden.
<namn="Kalle">
<namn>Kalle</namn>[15]

SOAP

SOAP är att XML baserat protokoll för utbyte av data i en decentraliserad, distribuerad miljö[16]. Protokollet skapar möjligheter för kommunikation mellan olika plattformar och tillåter olika språk att tala med varandra. Om vi tar ett program implementerat i Java som körs på en dator med Linux, så kan vi genom SOAP kommunicera med ett annat program

[15] Bray et. al. (2001)
[16] Gudgin, M et al, (2000)

skapat i C# som exekveras på en Windows server. Det är anledningen till att SOAP har blivit en viktig del i web services eftersom själva grundtanken är att det ska vara plattformsoberoende och språkoberoende.

SOAP protokollet består av tre delar:[17]

- envelope, ett övergripande ramverk för vad som finns i meddelandet och beskriver även vem som ska ta hand om det.
- encoding rules, regelverk för beskrivning av olika data typer, två datorer som kommunicerar måste komma överens om hur man packar sina olika typer för att det ska vara möjligt för den andra datorn att packa upp och använda datan på ett korrekt sätt.
- RPC, definierar en konvention för hur man ska representera anrop och svar över Internet från procedurer som ligget på ett fjärran objekt[18].

En applikation som använder SOAP har krav på sig att den måste bearbeta ett inkommande meddelande på följande sätt[19]:

1. Identifiera alla delar i meddelandet som är avsedda för applikationen.
2. Verifiera att det finns stöd för alla delar identifierade i steg ett på applikationen. Om det inte stämmer ska applikationen kassera meddelandet och returnera ett felvärde.
3. Om applikationen ej är slutstationen för meddelandet ska alla delar identifierade i steg ett tas bort före det att meddelandet skickas vidare.

Envelope (kuvert)
Ett meddelande i SOAP är ett XML dokument som består av ett obligatoriskt SOAP kuvert, en frivillig SOAP header (huvud) och en obligatorisk SOAP body (stommen). Kuvertet är topp elementet i det XML dokument som representerar meddelandet. Huvudet är ett flexibelt ramverk där man kan definiera krav på applikationen. Om man vill ange ett lösenord eller det kanske är en tjänst som tar betalt då kan man ange ett speciellt konto i SOAP huvudet[20].

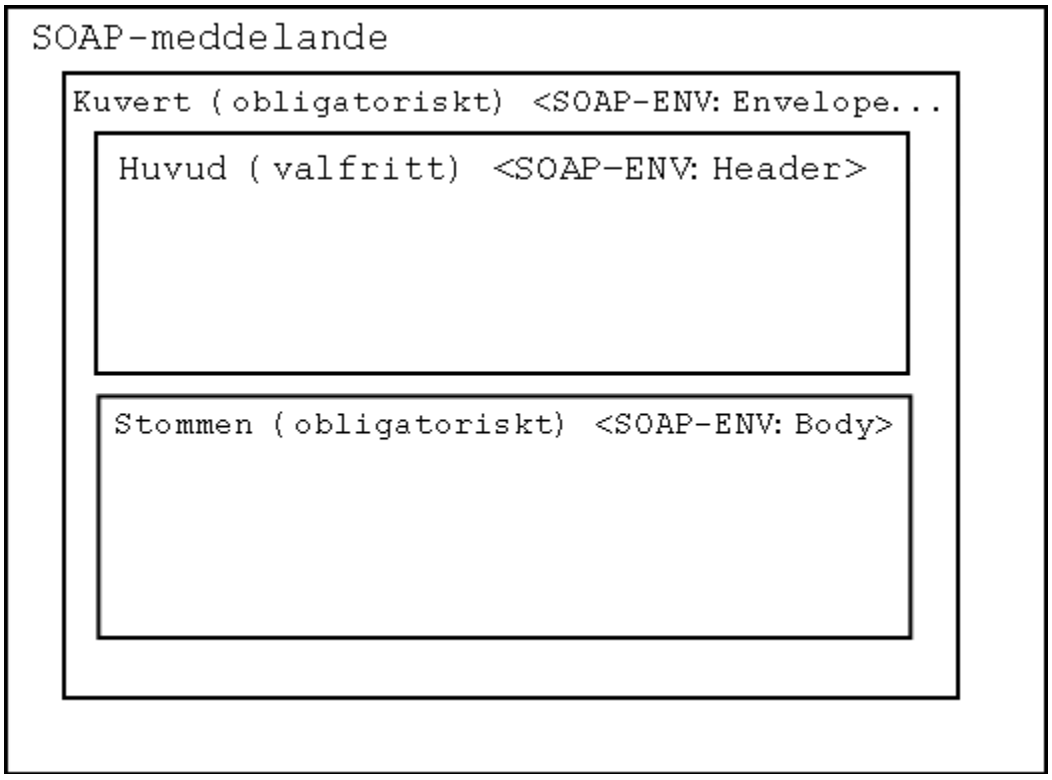
```
<SOAP:ENV:Header>
  <ns1:PaymentAccount xmlns:ns1=”urn:ecerami” SOAP-ENV
    mustUnderstand=”true”>mittKonoNr</ns1:PaymentAccount>
</SOAP-ENV:Header>
```
[21]

Detta huvudet specificerar ett konto som måste förstås och bli behandlat av applikationen som tar emot meddelandet. Stommen i SOAP meddelandet håller informationen som är avsedd för slut mottagaren det vill säga datan som man avsåg att skicka till en fjärroperation. Här följer ett exempel på ett komplett SOAP meddelande:

[17] Se figur 2
[18] UserLand Software, Inc. (2001)
[19] The SOAP Message Exchange Model, http://www.w3.org/TR/SOAP/
[20] SOAP Envelope, http://www.w3.org/TR/SOAP/#_Toc478383494
[21] Cerami, E, (2002)

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
     <t:Transaction xmlns:t="some-URI"
        SOAP-ENV:mustUnderstand="1">
           5
     </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
        <symbol>DEF</symbol>
     </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Struktur



Figur 2 Struktur över SOAP-meddelande.

Web Services Description Language (WSDL)

En beskrivning av WSDL har gjorts i boken Web Services essentials[22]. WSDL är en av grundstenarna i web services då det är en specifikation av tjänsten i ”vanlig” XML-grammatik som samlas i ett dokument som publiceras på Internet. WSDL beskriver tjänsten och tar främst fram de fyra viktigaste typerna av data om hur kontakt kan ske mellan klient och tjänst[23]:

- Gränssnittet, information som anger alla offentligt tillgängliga funktioner.
- Data typer, beskrivning av alla anrop och av svar, hur de ser ut och vilken data typ som förväntas.
- Kontakt, information om det protokoll som används för att transportera data mellan service och klient.
- Adress, beskrivning av var man hittar den specifika servicen.

Enkelt beskrivet så är ett WSDL-dokument ett kontrakt mellan den som vill använda en tjänst (klienten) och mellan den som tillhandahåller tjänsten. Vid användandet av WSDL kan en klient lokalisera en tjänst och sedan anropa dess publikt tillgängliga tjänster.

WSDL Specifikation

WSDL är ännu inte någon officiellt godkänd standard enligt W3C. WSDL version 1.1 som blev upplagd på W3C i mars 2001 har inte blivit en standard men det väntas den att bli eftersom flera stora aktörer som IBM, Microsoft, Ariba och med flera har enats om detta utkast. Den senaste versionen finns tillgänglig på http://www.w3.org/TR/wsdl , där även specifikationen finns att tillgå.

WSDL bygger på sex stycken större element som tillsammans beskriver tjänsten:

definitions

Detta är dokumentets root (huvud) element, det definierar namnet på tjänsten, deklarerar de olika ”namespaces” eller namnrymder som används i dokumentet. En namnrymd är den plats där definitionen ligger för det protokollet som anges. TargetNamespace är en unik adress som i detta fallet refererar till sig själv, men dokumentet måste inte finnas på denna platsen utan det viktiga är att det är en unik identifierare[24].

```
<definitions name=”HelloService”
        targetNamespace=”http://hobbe.informatik.gu.se/HelloService.wsdl”
        xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
        xmlns:xsd=”http//www.w3.org/2001/XMLSchema”>
```

types

Type elementet beskriver vilken eller vilka datatyper som används mellan klienten och servern. WSDL använder en specifikation från W3C XML Schema som ett förval. Här beskriver type att det kommer en textsträng:

[22] Cerami, E, (2002)
[23] Chinnici et. al.(2003)
[24] Chinnici et. al.(2003)

<part name=”firstName” type=”xsd:string”/>
xsd prefixet refererar till namnrymden för XML Schema som är definierat tidigare i definitions elementet.

message

Vilket meddelande kommer att skickas? Message skildrar ett envägs meddelande, ett anrop eller ett svar. Det definierar namnet på meddelandet och innehåller noll eller flera message delelement som kan refereras till som parametrar eller som svarsvärden från en tjänst.

```
<message name=”SayHelloRequest”>
    <part name=”firstName” type=”xsd:string”/>
</message>
```

Om nu funktionen väntar sig ett flertal argument eller returnerar multipla värden anger man ett flertal part element.

```
<message name=”SayHelloRequest”>
    <part name=”firstName” type=”xsd:string”/>
    <part name=”lastName” type=”xsd:string”/>
</message>
```

portType

portType elementet kan kombinera flera message element för att skapa en komplett operation som utför ett flertal olika typer av underoperationer. I detta exempel beskrivs en operation ”sayHello” som i sin tur utför två olika underliggande procedurer, ett inkommande meddelande ”SayHelloRequest” och ett utgående ”SayHelloResponse”.

```
<portType name=”Hello_PortType”>
    <operation name=”sayHello”>
        <input message=”tns:SayHelloRequest”/>
        <output message=”tns:SayHelloResponse”/>
    </operation>
</portType>
```

Prefixet tns refererar till det targetNamespace som vi angav i definitons elementet.
WSDL använder fyra olika mönster för att fastställa olika operationer:
Envägskommunikation, tjänsten tar emot ett meddelande, operationen har ett enkelt input element.
Fråga-svar, servicen tar emot en hälsning och skickar ett svar. Man anger därför ett input och ett output element.
Söka-svar, Web servicen skickar ett anrop och tar emot ett svar, operationen börjar med ett output följt av ett input element.
Tillkännagivande, tjänsten skickar ett meddelande och använder bara ett output element.

binding

Denna enhet ge oss detaljer om hur en portType operation kommer att sändas över nätet. Man kan ange en mängd olika transport protokoll som HTTP GET, HTTP POST eller SOAP. Det är möjligt att definiera multipla bindings för en portType. Själva binding enheten specificerar name och type attributen.

```
<binding name=”Hello_Binding” type=”tns:Hello_PortType”>
```

service

Här beskrivs på vilken adress tjänsten ligger.

```
<service name=”Hello_Service”>
        <soap:adress location=”http://localhost:8080/soap/…”/>
</service>
```

Exempel WSDL dokumentet i helhet:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<definitions name=”HelloService”
  targetNamespace=http://www.ecerami.com/wsdl/HelloService.wsdl
  xmlns=http://schemas.xmlsoap.org/wsdl/
  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns:tns=http://www.ecerami.com/wsdl/HelloService.wsdl
  xmlns:xsd=http://www.w3.org/2001/XMLSchema>

  <message name=”SayHelloRequest”>
    <part name=”firstName” type=”xsd:string”/>
  </message>
  <message name=”SayHelloResponse”>
    <part name=”greeting” type=”xsd:string”/>
  </message>

  <portType name=”Hello_PortType”>
    <operation name=”sayHello”>
      <input message=”tns:SayHelloRequest”/>
      <output message=”tns:SayHelloResponse”/>
    </operation>
  </portType>

  <binding name=”Hello_Binding” type=”tns:Hello_PortType”>
    <soap:binding style=”rpc” transport=http://schemas.xmlsoap.org/soap/http/>
    <operation name=”sayHello”>
      <soap:operation soapAction=”sayHello”/>
      <input>
        <soap:body
          encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
          namespace=”urn:examples:helloservice”
          use=”encoded”/>
      </input>
      <output>
        <soap:body
          encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
          namespace=”urn:examples:helloservice”
          use=”encoded”/>
      </output>
    </operation>
```

```
  </binding>

  <service name=”Hello_Service”>
    <documentation>WSDL för HelloService metoden</documentation>
    <port binding=”tns:Hello_Binding” name=Hello_Port”>
      <soap:adress location=”http://localhost:8080/soap/servlet/rpcrouter”/>
    </port>
  </service>
</definitions>
```
[25]

Vattenfallsmodellen

En modell som ofta används för att beskriva de olika faserna av systemutveckling är Livscykelmodellen eller som den också kallas vattenfallsmodellen.[26] Denna modell beskriver arbetet som en arbetsgång där de olika delaktiviteterna slutförs innan nästa tar vid därav namnet vattenfall.[27] Med det menas att resultatet från föregående aktivitet används som grund för att lösa nästa.

Aktiviteterna är:

Kravspecifikation – Initieras av ett behov som måste tillgodoses. Vanligen uppstår detta behov i en organisation. Behovet omformuleras till en systemspecifikation som definierar vad systemet ska lösa. Detta dokument kallas ofta för Project Scope, detta beskrivs av David Brown:

> ”Ett kontrakt över vad projektet ska producera. Project Scope berättar i generella termer vad systemet kommer att göra, vilka funktioner som kommer att vara en del av det och vilka användare det kommer att betjäna. Det deklarerar också vad som inte kommer att vara en del av systemet.”[28]

System och mjukvarudesign – Etablerar en överordnad systemarkitektur. Mjukvarudesign involverar identifikation och beskrivning av fundamentala system abstraktioner och deras inbördes relationer. Syftar till att analysera problemområdet:

> ”Den del av en omgivning som administreras, övervakas eller styrs av ett system.”[29]

Här bestäms även hur systemet ska delas upp för att bli hanterbart vid implementeringen. Beslut tas om den arkitektur som systemet ska byggas kring och på vilket sätt systemet

[25] Cerami, E (2002)
[26] Sommerville Ian, (2001)
[27] Se figur 3
[28] Brown, D, (1997)
[29] Mathiassen et al, (2000)

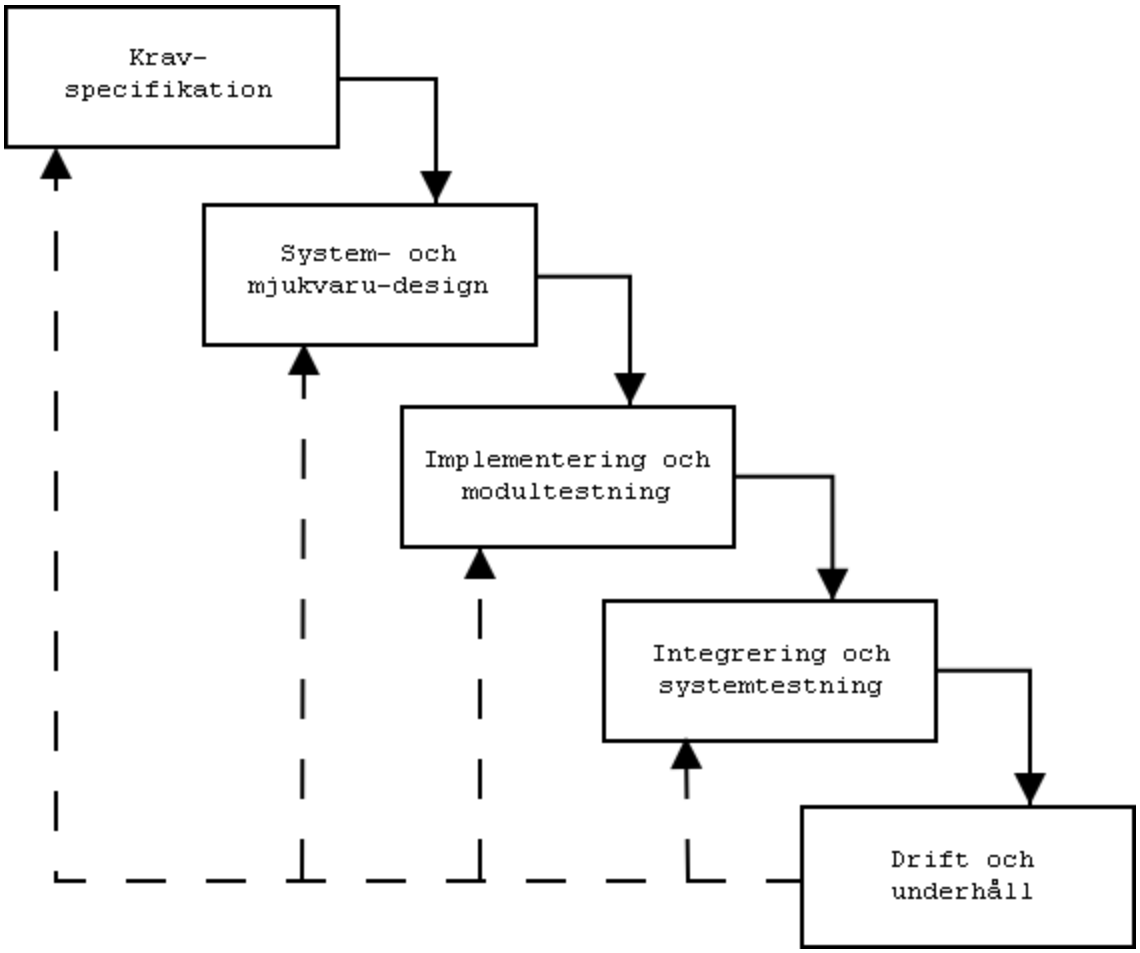
ska kommunicera med andra system. Här använder man sig av s.k. stegvis förfining[30]för att bryta ned den ursprungliga programuppgiften i mindre och mer hanterliga delar. De algoritmer och metoder som behövs i systemet designas och användargränssnitten utformas.

Implementering och modultestning – I denna fasen realiseras designen av mjukvaran som en grupp av moduler. Modultestning innebär en verifikation över att varje del, modul lever upp till sin specifikation.

Integrering och systemtestning – De olika modulerna eller de olika programmen blir integrerade och testas som en helhet för att säkerställa att systemkraven uppfylls. När testningen är avslutad kan systemet levereras till kund. Detta kan utmynna i ett allomfattande testprotokoll.[31]

Drift och underhåll – Denna fas är aktiv så länge systemet är i drift. Programfel s.k. buggar rättas till. Systemet underhålls – och uppdateras eventuellt med t.ex. nya versioner.

[30] Bell, D, (2000)
[31] Kerningham & Pike (1999)



Figur 3 Bild av vattenfallsmodellen.

Resultat

Fallstudier

För att jämföra hur väl vattenfallsmodellen fungerar vid utveckling av web services har en fallstudie kring detta genomförts. Denna fallstudie har jämförts med en annan fallstudie vilken beskriver en traditionell applikationsutvecklingsprocess som härrör från en laboration i Objekt Orienterad systemutveckling vars mål var att skapa en e-postklient med grafiskt gränssnitt realiserad i programspråket C++ och QTs[32] API för gränssnittet. Dessa två uppgifter är valda för dess likheter i omfattning och storlek samt den bedömda graden av komplexitet de innebär.

Fallstudie 1 Den grafiska e-postklienten

Den laborationsspecifikation (se appendix 1) som låg till grund för laborationen fastslog vissa krav för den färdiga e-postklienten dels för vilken funktionalitet som skulle implementeras och dels styrningar över vissa val av tekniker och protokoll som skulle användas.

Beskrivning av arbetsgången enligt vattenfallsmodellen

Kravspecifikation

Labspecifikation betraktades som en del av det första steget i den tidigare beskrivna vattenfallsmodellen som ska utmynna i kravspecifikationen. Labspecifikationen gav även svar på vissa frågor som normalt härrör till designen av systemet. Bland dessa kan nämnas implementeringsspråk. Kravspecifikationen gavs i två möjliga nivåer med grundläggande funktionalitet som var ett minimikrav och en nivå för extra funktionalitet som i ett verkligt läge skulle kunna motsvaras av en version två av ett färdigt system. Labspecifikationen gav utvecklarna tillräcklig frihet att på ett kreativt sätt utveckla produkten och var i sin enkelhet tydlig nog för att beskriva vad som systemet ska hantera.

System och mjukvarudesign

Nästa fas i vattenfallsmodellen syftar till en färdig design av systemet. Vi började med att analysera problemområdet som till stora delar redan var känd varför analysen av problemområdet blev ganska enkel. Det finns standarder kring epost-hantering som tydligt beskrivs i så kallad RFC[33]:er vilka underlättar för en utvecklare att analysera problemområdet. Vi kände att dessa RFC:er var tämligen omfattande men vi valde att endast betrakta det som var relevant för vårt projekt vilket förenklade det hela avsevärt. Mer kraft fick dock läggas på att finna de klasser systemet skulle innehålla. Denna analys utmynnade i ett klassdiagram med de ingående klasserna, dess inbördes relationer med tillhörande kopplingar samt de attribut och metoder klasserna skulle implementera.

Nästa steg i processen som ska leda fram till en färdig system och mjukvarudesign är analysen av användningsområdet. Denna analys ska leda till förståelse för vad systemet ska kunna göra. Då målsystemet till stor del var känt löste vi denna uppgift genom att

[32] © Trolltech. http://www.trollthech.com
[33] RFC Request for Comments. Ett slags standard över den teknologi som används.

”brain-storma” och ganska osystematiskt prata oss fram till funktioner och hur användaren ska interagera med systemet. Vi valde att ej skapa så kallade användarfall utan utgick från skisser av gränssnitten för att finna de olika funktioner systemet tvunget skulle behöva implementera.

Ytterligare steg i denna process är att bestämma en arkitekturdesign för systemet. För den applikation vi skulle bygga valde vi en enkel arkitektur med tre lager. En gränssnittskomponent med vars hjälp användaren interagerar med systemet. En modellkomponent som implementerar klassdiagrammet och de funktioner klasserna har. Det tredje lagret används till lagring av de data som måste lagras när systemet stängs ned. Denna lagring sker i enkla xml-liknande filer.

Implementering och modultestning

Implementeringen av epost-klienten skedde enligt labspecifikationen i C++. Vi valde att skapa all funktionalitet klass för klass utan att implementera något grafiskt gränssnitt. Detta syftade till att hela tiden ha fungerande moduler som efter hand de var klara testades. Det gav en säkerhet och robusthet som gjorde att systemet utan risk kunde byggas vidare och att man hela tiden visste att det tidigare skapade fungerade som det skulle. Vi testade dessa moduler var för sig och då nya moduler tillkom tillsammans med dessa i ett enkelt textbaserat gränssnitt. De problem vi under programmeringens gång stötte på berodde till stora delar på problem med några av de C++ programbibliotek vi använde. Dessa var inte alltid kompatibla med varandra vilket gjorde det hela lite problematiskt och oförutsägbart. När dessa problem överbryggats upplevde vi systemet som mycket robust och tillförlitligt. Den sista delen som implementerades var det grafiska gränssnittet mot användaren. Detta implementerades med ett bibliotek som det norska företaget Trolltech utvecklat som heter QT. Denna implementering gick relativt snabbt då vi redan hade funktionerna klara och testade.

Integrering och systemtestning

Integreringen av modulerna gjorde vi efter hand i implementeringsfasen. När hela systemet var färdigt var även integrationen klar. Detta innebar att vi inte hade någon egentlig fas endast för integrering. E-postklienten skulle inte heller integreras med några andra system varför denna aktivitet ej blev aktuell. Testningen av det färdiga systemet skedde ad hoc utan några speciella metoder för testning och är därför inte helt representativ.

Drift och underhåll

Då epost-klienten inte varit i någon egentlig drift är det svårt att beskriva fasen för drift och underhåll. Denna fas innefattar normalt sätt buggfixar och implementation av ny funktionalitet.

Slutsatser

Som slutsatser kring denna fallstudie kan sägas att vi utan allt för stora förändringar kunde följa vattenfallsmodellen kring systemutvecklingen. Stegen följdes i stort sätt från det första till det sista. Vattenfallsmetoden innebär per definition att man löser ett steg för att sedan gå vidare till nästa. Detta utgör dock inte ett vattentätt skott utan återkoppling är

tillåten precis figur 3 antyder. Vi kunde begränsa vår återkoppling till vissa förändringar i attributen till klasserna och att viss funktionalitet lades till efter hand.

Fallstudie 2 Webservice-prototypen

För att jämföra vanlig applikationsutveckling med utveckling av web services valde vi att skapa en egen prototyp av en web service. Detta för att det finns mycket lite att läsa om utvecklingsmetodik kring web services och för att själva bättre sätta oss in i ämnet. Den fallstudie som följer beskriver hur själva webservicen utvecklades men det går inte att helt utlämna utvecklingen av en klient som kan koppla upp sig mot web servicen även om denna inte är intressant i egentlig mening. Vi valde att skapa en web service som med ett namn och postnummer som indata returnerar telefonnummer och adressuppgifter för de personer som passar in. Själva databasen för personuppgifter lånar vi från Gula Sidornas webbsida för sökning av privatpersoner.

Kravspecifikation

Kravspecifikationen vi arbetade efter beskrev applikationens funktioner och att den skulle implementeras med hjälp av web services. Funktionaliteten skulle vara sådan att implementation av klienten är ovesäntlig så länge den skickar och tar emot information enligt SOAP-protokollet.

System och mjukvarudesign

Återigen analyserade vi problem- och användningsområdet. Vi gjorde ett enkelt klassdiagram och beskrev de funktioner som de olika klasserna skulle implementera. Det vi kunde konstatera var att denna funktionalitet endast gällde det som var relevant för själva utökningarna det vill säga inga funktioner för kommunikation behövde analyseras. Analysen av användningsområdet blev även den minimalistisk då den som skapar klienten kan välja vilken av web servicens funktionalitet som ska implementeras.
I den här fasen av utvecklingen ska även arkitekturen analyseras och väljas. Då valet av web services är gjord har även stor del av arkitekturen lagts fast. Eftersom web service i sig är en arkitektur som följer vissa standarder blev även denna punkt ganska liten. Det som vi var tvungna att lägga mer kraft på var vilken teknisk plattform vi skulle bygga web services kring och vilket programspråk vi skulle implementera i. Då vår tidigare erfarenhet av Java var stor kände vi att detta val föll sig naturligt. Nu gällde det att hitta en applikationsserver med stöd för web services som hanterar java. Vi hittade två alternativ dels WASP-servern från Systinet och en implementation av SOAP från Apacheprojektet som kördes på opensource-webserven Tomcat även den från Apacheprojektet. Efter diverse tester och försök valde vi att arbeta vidare med den Tomcatabaserade servern som fungerade bra och som var enkel att använda, dessutom gratis. Det visade sig att valet av plattform etc. tog mycket mer tid än de tidigare aktiviteterna tillsammans. Vi kände att det var ett enormt trappsteg att ta sig över att lära sig tekniken kring web services men när valet väl var gjort och förståelsen infunnit sig kunde arbetet ta fart med full kraft igen.

Implementering och modultestning

Nu inleddes arbetet med att implementera designen. Vi valde att dela upp programmeringen så att en utvecklade kommunikation och utsökning mot vår informationskälla Gula Sidorna och en började skapa själva web service-koden. Programmeringen av web service var i sig inte speciellt avancerad men problematiken låg i att stegen för att testa den kod man gjort var många. Detta berodde på att den kompilerade koden måste placeras på rätt ställe i serverns katalogstruktur. Dessutom måste den nya versionen av koden rapporteras med ett speciellt webgränssnitt till servern samt att servern måste startas om. Efterhand löste vi dessa flerstegsuppgifter med egentillverkade skript som hanterade de olika stegen men trots detta blev det ett tidskrävande jobb varför man drog sig för att testa små förändringar i koden. Ytterligare källor till arbete är det faktum att man för att testa helheten måste ha en klient som ansluter till web servicen. Denna klient kodade vi efterhand i java med minimal funktionalitet och gränssnitt mot användaren.

Integrering och systemtestning

Vi avslutade utvecklingen med att koppla ihop funktionskomponenten med själva webservicen. Detta skedde utan några större problem och vi kunde testa helheten mot vår klient. Vi fann att tjänsten fungerade på ett mycket bra sätt. SOAP-implementationen hanterade även på ett mycket bra sätt felaktiga anrop från klienten utan att systemets stabilitet påverkades. Detta gör att tekniken kring web services verkar mycket robust och tillförlitlig.

Drift och underhåll

Liksom i den första fallstudien saknar vi underlag för att beskriva drift och underhållsfasen i vattenfallsmodellen.

Slutsatser

Vi fann att utvecklingen väl följde vattenfallsmodellen, dock kan sägas att ganska lite tid lades till analys och design samt att implementeringen går relativt snabbt. Vi fann att för att snabba på programmeringsfasen bör någon form av utvecklingsmiljö användas för att undvika tidsspillan vid testning.

Analys

Arbetet har med hjälp av litteraturstudier och två fallstudier studerat hur traditionella systemutvecklingsmetoder fungerar vid utveckling av web services. Vi diskuterar utifrån de stegen som vattenfallsmodellen beskriver systemutveckling och tittar på de likheter och skillnader vi funnit mellan traditionell systemutveckling och utveckling av web services.

Kravspecifikation

Vid analysen av en traditionell applikation startar man med att skapa en system specifikation över vad systemet ska kunna göra och vad det inte ska göra. En systemspecifikation över en webbtjänst skiljer sig inte nämnvärt från ett traditionellt system. I båda fallen skapas en generell bild över vad systemen ska tillhandahålla för funktioner och vad de ej ska göra. Denna specifikation fungerar som ett avtal mellan beställaren och leverantören. I de flesta fall blir en specifikation över en webbtjänst mindre komplicerad jämfört med ett vanligt datorsystem, då en webbtjänst ofta levererar en begränsad funktion från ett befintligt system. Våra fallstudier är på en punkt lika då båda systemen ska kommunicera med andra system över Internet. Skillnaden mellan dessa är att i fallet med web services behöver ingen hänsyn tas till hur protokollet mellan systemen ska vara uppbyggt då det redan implementerats av applikationsservern som används.

System och mjukvarudesign

Analysen av epost-klienten innebar en fullständig analys av problem och användningsområdet med tillhörande användarscenarier. Utöver detta tillkom en fullständig beskrivning av de grafiska gränssnittet. När det gäller utveckling av web services var analysen av problemområdet begränsad till de klasser och funktioner som krävdes. Analysen av användningsområdet kunde göras avsevärt mindre då inga användarscenarier och ej heller några gränssnitt behövde skapas. Detta gjorde den inledande analysen mycket mindre än vid utvecklingen av epost-klienten där avsevärt arbete lades. Många webbtjänster som implementeras idag är tjänster som kopplas mot redan befintliga system och kan därför dra nytta av den tidigare gjorda analysen av problemområdet. I dessa fall kan man se webbtjänster som nya fönster för nya användningsområden och kräver därför en mindre omfattande analys.

Vid arkitektur design gällande mer traditionella system får utvecklarna göra en gedigen arbetsinsats för att skapa en robust och funktionell arkitektur. Många olika val måste ske när det gäller programmeringsspråk, protokoll mot andra system, gränssnitt mot användare/andra system och intern arkitekturdesign. En utvecklare av en webbtjänst slipper göra vissa val gällande, protokoll och gränssnitt. Detta beror på att webbtjänsten redan har en specifikation över vilka protokoll som ska användas och hur de ska användas. Det finns ej heller någon användarinteraktion att ta hänsyn till då den är obefintlig. All interaktion med systemet sker med hjälp av det förutbestämda SOAP protokollet. Det är upp till utvecklare av klienter till tjänsten att implementera och följa SOAP protokollet om de vill interagera med tjänsten. I och med användandet av

definierade protokoll är det ingen skillnad mellan system och människor som interagerar med systemet.

Då vi utvecklade epost-klienten var vi styrda till att använda C++ för implementeringen och vid utvecklingen av web servicen valde vi att använda Java.
När vi skulle börja utveckla web servicen ställdes vi inför valet av plattform. Detta val är det stora att göra innan implementeringen tar vid. Detta skiljer sig ganska mycket från utvecklingen av epostklienten där sådana val har mindre betydelse.
Valet av arkitektur vid utvecklingen av epost-klienten var mera fritt och öppet för eget tycke och smak.

Implementation och modultestning

Implementationen vid vanlig traditionell systemutveckling följer oftast analysen och designen väl. De klasser man analyserat fram implementeras på ett naturligt sätt i det programspråk man valt under förutsättning att stödet för objektorientering är gott. Detta gäller även i stora drag för implementationen av en webbtjänst då det bakomliggande ramverket i form av applikationsserver löser de specifika problemen med de protokoll som webbtjänsten kommunicerar med. En stor skillnad är dock den dokumentation som krävs för en webbtjänst. Man presenterar sin webbtjänst med ett beskrivande dokument som följer en hårt typad standard. Detta syftar till att på ett automatiserat sätt kunna skapa klienter som kommunicerar med webbtjänsten. Denna dokumentation i form av ett särskilt dokument (WSDL) publiceras på nätet tillgängligt för alla som vill kunna ansluta till tjänsten. Denna typ av dokumentation för webbtjänsten kan jämföras med de RFC som skapas för applikationer som har ett öppet systemgränssnitt. Dessa RFC blir ofta mycket omfattande eftersom de själva bildar en hel standard för kommunikationen inklusive protokoll. Fördelen med webbtjänsterna är att protokollet för kommunikationen mellan klient och server ej behöver implementeras då mellanliggande applikationer löser detta problem.

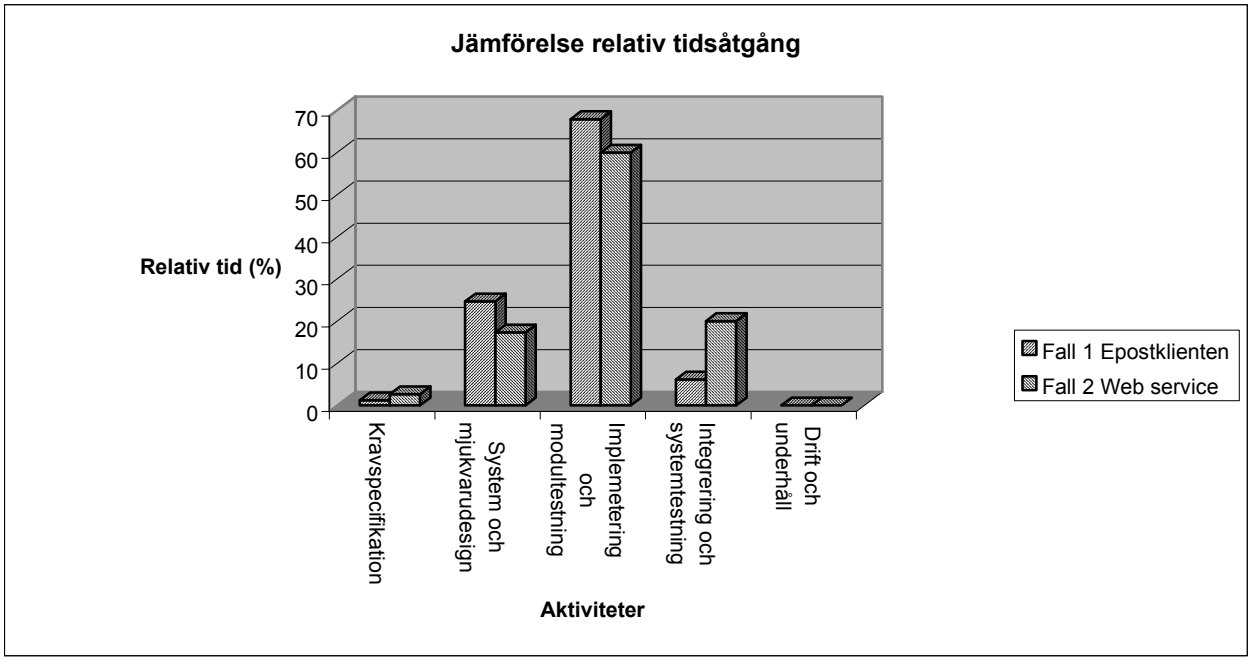
Skillnaden vid implementationen av epost-klienten och web servicen var möjligheterna att testa modulerna var för sig. I fallet med epost-klienten kunde vi med hjälp av ett testprogram provköra och säkerställa funktionaliteten i modulerna. I web servicen fanns ej denna möjlighet, dels för att allt hänger samman genom komplexa beroenden mellan modulerna och dels på grund av det faktum att testningen försvåras av den komplicerade arbetsgången för driftsättningen av nya versioner.

Integration och systemtestning

Eftersom epost-klienten kunde testas efterhand blev integrationsfasen relativt enkel och smärtfri. Eftersom web servicen redan från början integrerades med driftsmiljön fanns ej heller här några större problem. Dock kan tilläggas att epost-klientens många klasser och uppbyggnad gjorde den mera komplex än den till funktion minimala web servicen i detta hänseende.
Med web servicen får man ett robust system då den SOAP-implementation man använder hanterar felaktigheter från klienter på ett säkert sätt varför kortare tid behöver läggas på systemtestning i förhållande till epost-klienten.

Kvantitativ jämförelse mellan fallstudierna.

För att tydliggöra skillnaderna mellan de olika fallen har en tidsstudie genomförts för att mäta tidsåtgången i vattenfallsmodellens olika steg. Dessa jämförelser baserar sig på den relativa tidsåtgången vid de två fallstudierna. Tiden för val av plattform och inlärningstiden för diverse tekniker i båda fallstudierna har utelämnats. I figuren kan utläsas att relativt sett mer tid åtgår till design och implementation vid applikationsutvecklingen vilket relativt sett ger att tiden för integrering blir lägre än för web services.



Figur 4 Stapeldiagram över relativ tidsåtgång vid utveckling av web services och traditionella applikationer.

Diskussion

Arbetet visar att utvecklingen av web services inte nämvärt skiljer sig från vanlig applikationsutveckling med tanke på kopplingen till vattenfallsmodellen. Dock har vi funnit att olikheter finns mellan web services och traditionell utveckling i mängden arbete som behöver läggas vid de olika stegen i processen. Utvecklingen av web services går som helhet fortare än annan systemutveckling då ramverket redan är klart och standarder för kommunikation är implementerade i applikationsservrarna. Detta resonemang förutsätter att den tröskel i form av kunskapsinhämtning och val av teknisk plattform är överstigen. Arbetet har funnit att en utvecklingsmiljö specialkomponerad för web services nästan är ett måste för att effektivt utveckla sin tjänst. Detta för att minska tidsåtgången vid implementering och testning.

Avslutande diskussion

Arbetsgången för att nå fram till resultaten har varit har varit en krokig väg att gå. Valet att skapa en prototyp medförde att vi lärde oss tekniken kring web services på ett bra sätt men arbetets framskridande gav den dålig effekt. Man hade kunnat komma fram till samma resultat genom en traditionell intervjuundersökning som hade varit mera effektiv men i vårt tycke alltför tråkig och förutsägbar. Även om arbetet har visat att vattenfallsmodellen är en framkomlig väg för utveckling av web services skulle en fortsatt explorativ studie där vattenfallsmodellen jämförs med en modell där möjligheterna till iteration är större exempelvis RUP[34] vara mycket intressant och skulle eventuellt kunna leda till att en ny metod utvecklas.

[34] RUP Rational Unified Process http://www.rational.com/products/rup/whitepapers.jsp

Referenser

Publikationer:

Backman, J (1998). *Rapporter och uppsatser*, Lund, Studentlitteratur
Bell, D (2000). *Software Engineering – A Programming Approach 3rd edition*, New York: Addison Wesley
Brown, D (1997). *An introduction to Object-Oriented Analysis*, New York: Wiley
Cerami, E (2002), *Web services essentials,* Farnham: O'Reilly
Kerningham, B & Pike, R (1999). *The practice of programming, New York:* Addison Wesley
Kjellén, B, Söderman, S. (1980). *Praktikfallsmetodik,* Stockholm: LiberTryck
Mathiassen et al. (2001). *Objektorienterad analys och design,* Lund Studentlitteratur
Merriam, Sharan B (1994), *Fallstudien som forskningsmetod*, Lund, Studentlitteratur
Patel, R. & Davidson, B. (1994). *Forskningsmetodikens grunder – Att planera, genomföra och rapportera en undersökning.* Lund: Studentlitteratur.
Sommerville, I, (2001) 6rd edition. *Software Engineering,* Harlow: Pearson Education Limited
Yin, Robert (1994). *Case Study Research*, Sage Publ

Internet:

Tim Bray et. al.
http://www.w3.org/TR/2000/REC-xml-20001006
http://www.w3schools.com/xml/default.asp

Ethan Cerami
http://www.oreillynet.com/pub/a/webservices/2002/02/12/webservicefaqs.html

Chinnici et. Al.(2003)
http://www.w3.org/TR/wsdl12/

Fensel, D & Bussler,C(2002)
http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6X4K-46THFYB-1&_user=646099&_coverDate=09%2F30%2F2002&_alid=78412444&_rdoc=50&_fmt=full&_orig=search&_cdi=7329&_sort=d&_st=4&_docanchor=&_acct=C000034699&_version=1&_urlVersion=0&_userid=646099&md5=be10795918852f6e9ec41c14757b597b

Gudgin et al.(2001)
http://www.w3.org/TR/2002/CR-soap12-part1-20021219/

Trolltech. http://www.trollthech.com

UserLand Software, Inc. (2001) http://www.xmlrpc.com/

W3C http://www.w3.org/TR/wsdl

Appendix 1

Laborationsspecifikation epostklient

En enkel mailläsare
Mailläsare har vi väl alla åsikter om vad de ska kunna göra och ingen av de som finns i idag är perfekt som allas tycke och smak.

Mailprogram har idag blivit allt mer komplexa med alltifrån adressböcker, kalenderfunktioner, hantering av olika email protokoll som pop och imap, mailfiltrering och till hantering av html, jpgs osv. osv. osv.

Det allra mesta om hur mailhantering ska ske styrs idag via så kallade RFC:er (Request for comments) som beskriver en de facto standard. Det finns beskrivningar för exakt hur alla möjliga mailheaders ska se ut, hur email kan skickas, hur man via olika protokoll kan läsa email, hur man hanterar attachments mm mm Att följa dessa är viktigt så att vi både kan ta emot email skickat från vilket annat mailprogram som helst och sedan vill vi ju att våra mail ska vara läsliga för andra. Problemet är dock många gånger att alla mailprogram inte följer standard.

Uppgift:
Att skriva en tämligen enkel mailläsare som kan följande:

För G gäller att mailläsaren kan följande:
Skicka mail via smtp
Läsa mail från en pop-server
Kunna svara på mail
Sortera inkomna mail på Avsändare, Subject och datum
Kunna spara mail i en godtycklig fil
Preferencer, dvs. defaultinställningar, som vilken host vi ska läsa email ifrån, vilken är vår användaridentitet etc.
För VG gäller att dessutom implementera följande:
En enkel adressbok
Det ska vara möjligt att spara mail i mailboxar, dvs. man ska kunna skapa 1 eller flera mailboxar dit man kan flytta mail.
Alla sända mail ska automatiskt sparas i en Sent-box.
Man ska kunna göra forward på ett inkommet mail, dvs. hela mailet (exklusive headrar) skickas till en annan mailaddress.
När man gör reply på ett mail ska mailet man svarar på automatiskt kopieras in i mailkroppen.
Möjlighet att lägga till en signatur, dvs. en text sist i mailkroppen som automatiskt läggs till alla sända mail.
Ni behöver inte kunna hantera saker som attachments, MIME, html-kod osv. Däremot måste ni följa RFC:erna noggrannt, så att headerformat för t.ex. datum följs. De mail ni skickar måste vara fullt läsliga i t.ex. Netscape, Eudora eller Pine och där ha rätt datum osv. Ni ska också kunna ta emot mail från andra mailprogram som följer RFC:erna.

Bakgrundsinformation:
Smtp finns beskrivet i rfc2821, ersätter rfc821
Utseendet för mailheaders definieras i rfc2822, ersätter rfc822
Pop3 definieras i rfc1939
Alla dessa rfcer kan t.ex. hittas på The Internet Engineering Task Force site eller på Sunets ftp-site.

Inspiration:
För att få lite inspiration över hur en mailläsare kan tänkas se ut finns nu ett antal grafiska alternativ att titta på. Några av dessa är:

Några Open Source alternativ med grafisk gränssnitt
Mahogany,
Kmail,
Balsa,
Tkrat, göteborgsalternativet, mycket av koden skriven i TCL.
Mer eller mindre kommersiella:
Netscape Mail
Eudora
Mulberry
Generellt gäller
Allt arbete ska dokumenteras via tillämpliga delar av Mathiassen et. al. bok. Generellt gäller att laborationsuppgiften är ska implementeras m.h.a C++ och QT, ta CommonC++ biblioteket till hjälp för att sköta alla nätverkskontakter.

Appendix 2

Kodlista

```
/*
 * Copyright (c) 2001 Ethan Cerami. All rights reserved.
 * This code is from the book XML Web Services Essentials.
 * It is provided AS-IS, WITHOUT ANY WARRANTY either expressed or implied.
 * You may study, use, and modify it for any non-commercial purpose.
 * You may distribute it non-commercially as long as you retain this notice.
*/
/* Servicen tar emot en sökförfrågan från klienten och kopplar upp mot
 * Gula sidorna och utför sökningen. Returnerar sökresultat i forma av egendefinierad
 * XML.
 * Koden är ändrad och ombyggd av Jonas Henriksson och Hans Strandberg hösten 2003
 * under arbetet med vår C uppsats om Web services på Informatik, Handels Göteborg.
 * ex102-2@student.informatik.gu.se
 */
package com.ex102_2.service;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import org.apache.soap.util.xml.XMLParserUtils;
import org.apache.soap.util.xml.DOM2Writer;
import java.io.*;
import java.util.StringTokenizer;
public class PhoneService{

public Element getProduct (Element request)throws IOException{
        String fName="";
        String eName="";
        String zip="";
        //String fil="debug.txt";
        //PrintWriter fut = new PrintWriter(new BufferedWriter(new FileWriter(fil)));
        Gul gul=new Gul();

        /* Söker ut attributen från clientanropet med hjälp av
          en egen parserklass. */
        DOM2Writer domWriter = new DOM2Writer();
        OurXMLParser ourParser = new
        OurXMLParser(domWriter.nodeToString(request));
        fName=ourParser.getTagValue("fName");
        eName=ourParser.getTagValue("eName");
        zip=ourParser.getTagValue("zip");
        if(zip==null)// Kan vara fel ev. zip.equals(null) men null är ingen text?
           zip="";
```

```
        gul.setFname(fName);
        gul.setEname(eName);
        gul.setZip(zip);
        /* Söker på gulasidorna och sparar sökresultatet i en stringarray*/
        Browser brow=new Browser(gul.geturl());
        brow.postPage("kod.txt",gul.getpostString());
        String arr[] = gul.parseResult("kod.txt");

        //  Skapar ett XML-dokument som sparar sökresultat
        DocumentBuilder docBuilder = XMLParserUtils.getXMLDocBuilder();
        Document doc = docBuilder.newDocument();
        // Root elementet i dokumentet.
        Element resultSet=doc.createElement("buddylist");

        StringTokenizer sT;
        for(int i=0;i< arr.length;i++){
           //         fut.println("Arr: "+arr[i]);
           // Dela upp svaret från Gul objektet.
           sT = new StringTokenizer(arr[i],"¤");
           String addr=sT.nextToken();
           String zippa=sT.nextToken();
           String town=sT.nextToken();
           String telefon=sT.nextToken();

           // Skapa buddy noden där informationen om varje enskild person ligger.
           Element buddyNode = doc.createElement("buddy");
           buddyNode.setAttribute("phone",telefon);

           // Bygger en adress nod.
           Element adressNode = doc.createElement("adress");
           Text adressText = doc.createTextNode(addr);
           adressNode.appendChild(adressText);
           buddyNode.appendChild(adressNode);

           // Bygger en zip nod.
           Text zipText =doc.createTextNode(zippa);
           Element zipNode = doc.createElement("zip");
           zipNode.appendChild(zipText);
           buddyNode.appendChild(zipNode);

           // Lägger in buddynoden i huvud dokumentet.
           resultSet.appendChild(buddyNode);
        }
//      fut.close();

        return resultSet;
```

```
    }
}
```

Gul

```
/* En special klass för att hantera gulasidorna för att
 * kunna söka och hämta information.
 */
package com.ex102_2.service;
import java.io.*;
import java.util.regex.*;
import java.util.Vector;
public class Gul{
    private String url="privatpersoner.gulasidorna.se";
    private String fname="";
    private String ename="";
    private String zipcode="";
    //region:  0 är för hela Sverige,9=Göteborgsområdet
    private String region="0";
    private String packed[];

    public Gul(){
    }
    /*
     * En metod som plockar ur adress, telenr och zip från en textfil,
     * plockad från privatpersoner.gulasidorna.se.
     * Returnerar sökresultat packat i en String array avskild med ¤.
     */
    public String[] parseResult(String fil){
        try{
            boolean result=false;
            BufferedReader fin = new BufferedReader(new FileReader(fil));
            String rad=fin.readLine();
            Vector vec = new Vector();
            while(rad != null){
                // Skapa ett mönster objekt som matchar recip_address ,
                // det är i denna raden informationen ligger.
                Pattern pa = Pattern.compile("recip_address1=");
                Matcher ma = pa.matcher(rad);
                result = ma.find();
                if(result){
                    // Sök fram värden på adress,postnr ocg telefonnummer.
                    rad=rad.substring(rad.indexOf("recip_address1=")+15);
                    String adress=rad.substring(0,rad.indexOf("&amp"));
                    rad=rad.substring(rad.indexOf("recip_zip=")+10);
                    String zip=rad.substring(0,rad.indexOf("&amp"));
                    //431%2042 är samma som 431 42
```

```
                    // Byter ut %20 mot " ", för att få korrekt postnr.
                    Pattern p2 = Pattern.compile("%20");
                    Matcher m2 = p2.matcher(zip);
                    StringBuffer sb2 = new StringBuffer();
                    boolean result2 = m2.find();
                    if(result2) {
                          m2.appendReplacement(sb2, " ");
                    }
                    m2.appendTail(sb2);
                    zip=sb2.toString();
                    // Sök fram stad.
                    rad=rad.substring(rad.indexOf("recip_city=")+11);
                    String city=rad.substring(0,rad.indexOf("&amp"));
                    //Sök fram telefonnr recip_phone=031-872012
                    rad=rad.substring(rad.indexOf("recip_phone=")+12);
                    String phone=rad.substring(0,rad.indexOf("\">"));
                    // Packa till en sträng per sökresultat.
                    // Och lägg in i vectorn.
                    String answer=adress+"¤"+zip+"¤"+city+"¤"+phone;
                    vec.add(answer);
                }
                rad=fin.readLine();
            }
            fin.close();
            // Packa om vectorn till en Sträng array.
            packed =new String[vec.size()];
            for(int i=0;i<vec.size();i++)
                  packed[i]=(String)vec.elementAt(i);
      }
      catch (Exception e){
            System.out.println("Error in Gul.parseResult(), Exception: "+e);
      }
      return packed;
}
/* Metod som bygger en sträng för utsökning på gulasidorna.*/
/* Ändring: 021024 ny:last=250 gammal:last=25 */
public String getpostString(){
      String ps="search/hits.asp?"+
            "firstname="+fname+"&firstnameType=exact&lastname="+ename+
            "&lastnameType=exact&region="+region+"+&adress=&adressType=exact"+
            "&zipcode="+zipcode+"&title=&titleType=exact&first=1&last=250&y=13";
      return ps;
}
public String geturl(){return url;}
public void setFname(String n){
      fname=n;}
```

```
   public void setEname(String n){
         ename=n;}
   public void setZip(String n){
         zipcode=n;}
}
```

Browser

```
/**
 * En klass som sköter kommunikation med olika webbsidor.
 */
package com.ex102_2.service;
import java.io.*;
import java.net.*;
public class Browser{
   private String line="bapp";
   private String directory="";
   private String fil="temp.txt";
   private BufferedReader stdin;
   private Socket s;
   private BufferedReader input;
   private BufferedWriter output;

   public Browser(String url){
   try{
            stdin= new BufferedReader(new InputStreamReader(System.in));
            s = new Socket(url, 80);
            input = new BufferedReader(new InputStreamReader(s.getInputStream()));
            output = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
            output.flush();
         }
         catch (Exception e){
            System.out.println("Exception creating Browser object: "+e);
         }

   }
   public Browser(String url,String dir){
         try{
            directory=dir;
            stdin= new BufferedReader(new InputStreamReader(System.in));
            s = new Socket(url, 80);
            input = new BufferedReader(new InputStreamReader(s.getInputStream()));
            output = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
            output.flush();
         }
         catch (Exception e){
            System.out.println("Exception creating Browser object: "+e);
```

```
        }
    }
    /**
     * Metod för att anropa och skriva resultatet till en fil.
     */
    public String postPage(String filname,String postString){
        fil=filname;
        System.out.println("Metod postPage");
        try{
            output.write("GET /"+postString+"\r\n\r\n");
            output.flush();
            PrintWriter fut = new PrintWriter(new BufferedWriter(new FileWriter(fil)));
            while(line!=null){
                line=input.readLine();
                fut.println(line);
            }
            fut.close();
            System.out.println("Skrivning till fil: "+filname+" avslutad!");
        }
        catch (Exception e){
            System.out.println("Error in Browser.postPage(), Exception: "+e);
        }
        return "bapp";
    }
    /**
     * Metod som hämtar hem vald webbsida.
     * Under konstruktion. Sidan som hämtas sparas ej eller returneras inte på ett trevligt
sätt.
     */
    public String getPage(){
        try{
            if(directory.equals(""))
                output.write("GET / HTTP/1.0"+"\r\n\r\n");
            else
                output.write("GET /"+directory+"/"+"\r\n\r\n");
            output.flush();
            while(line!=null){
                line=input.readLine();
                // Här får det hända något kul som att skriva till en fil eller
                // concatenera en sträng.
            }
        }
        catch (Exception e){
            System.out.println("Error in Browser.getPage(), Exception: "+e);
        }
        return "bapp";
```

```
    }
}
```