



POLSAG Teknisk Review Summary

Indhold

Introduktion.....	3
Formål og afgrænsning af opgaven.....	3
Forudsætninger for læseren.....	3
Baggrund for dokumentet.....	3
Executive Summary.....	5
Teknisk review.....	7
Platform.....	7
Arkitektur.....	8
Overordnet arkitekturvurdering.....	9
Den POLSAG-specifikke SOM-model.....	9
Transaktionsstyring.....	10
Tekniske vurdering af integration til eksterne systemer.....	10
Arkitekturmæssig vurdering af integration til eksterne systemer.....	11
Office integration.....	12
Performance.....	12
CSC/Scanjour test "POLSAG Underbilag C10".....	13
APM Test 20110518.....	14
Status på performance test.....	14
Anbefalede tiltag omkring performance på kort sigt.....	15
Betragtninger omkring POLSAGs performance.....	16
Kodekvalitet.....	19
C#.....	19
Javascript.....	22
C++.....	23
PL-SQL.....	24
Test-strategi.....	25
Funktionelt testforløb.....	26
Test hos ScanJour.....	26
Test hos CSC.....	27



Test hos Rigspolitiet.....	27
Vurdering af det funktionelle test forløb	28
Anbefalinger	28
Stamdata	29
Logning	29
Udviklingsproces.....	30
Overblik	30
Status på dokumentationen	30
Risikovurdering af udviklingsprocessen.....	32
Udviklingseffektivitet.....	33
Dyretransport	33
Idræt	34
Udviklingsmiljøet	35
Referencer	37
Appendix A	38
Histogram over antallet af kodelinjer i Javascript filer	38
Javascript-filer som ikke eksplicit er inkluderet i layout-filer	38
Appendix B.....	43
C# kode-eksempel	43
Custom JSON formatter	64
F0192.cs	66
Javascript kode-eksempel.....	69
Javascript eksempel: Manglende oprydning af brug af IE-specifikt API og datanær programmering	97
Appendix C Spørgsmål omkring POLSAG ClientServices	117
Analyse	119
Udstilling af de eksterne systemer over HTTP	119
Udstilling af Business Process Handler-funktionalitet over HTTP (BPHandler.ashx)	120
Udstilling af fler-register Create/Update/Delete operationer over HTTP (ClientHandler.ashx)	121
Eksempel fra en Business Process Handler	121



Introduktion

Dette dokument indeholder en opsummering af de væsentligste detaljerede tekniske informationer, som er indsamlet under review af POLSAG, samt analyserne af samme, som danner basis for risikoanalysen og anbefalingerne.

Review'et har været afgrænset til POLSAG med tilhørende grænseflader. Randsystemerne og ESB'en er således ikke omfattet af review'et.

Formål og afgrænsning af opgaven

Opgaven har til formål at indkredse og dokumentere de største fremadrettede risici for POLSAG-projektet på det tekniske område. Der er ikke udført nogen vurderinger af projektførelsen, da det udelukkende handler om at afdække de fremadrettede risici, undtagen i de tilfælde, hvor det har været nødvendigt for at forstå processer eller selve applikationen.

Afdækningen af den tekniske side har været koncentreret omkring følgende hovedområder:

- Arkitekturen for løsningen
- Placeringen af POLSAG i Politiets systemlandskab
- Evaluering af den POLSAG-specifikke kode (baseret på stikprøver)
- Udviklingsdokumentation
- Test af POLSAG

Opgaven har ikke haft til formål at munde ud i egentlige anbefalinger. Vi har dog alligevel valgt at give nogle anbefalinger hvor vi har skønnet at det er oplagt.

Beskrivelserne i dette dokument er baseret på en gennemgang af den store mængde af POLSAG-dokumentation samt interviews med en stor del af projektets nøglepersoner. Det er i den sammenhæng nødvendigt at understrege at POLSAG er et meget stort og komplekst projekt, hvorfor der kan være information som er forbigået vores opmærksomhed af den ene eller anden årsag.

Forudsætninger for læseren

Det forudsættes at læseren er bekendt med POLSAG-løsningen, herunder den overordnede arkitektur for POLSAG løsningen og opbygningen af en løsning baseret på ScanJour CAPTIA.

Det forudsættes derudover at læseren er bekendt med Captia 4.2 samt POLSAG-projektet, herunder SOM, SOMASP og Captias extensibility punkter.

Såfremt læseren ikke opfylder ovenstående forudsætninger kan vi oplyse at den væsentligste dokumentation omkring disse emner er placeret i 'Captia Datamodel', 'Captia Guidelines and documentation', 'Overordnet POLSAG arkitektur', 'POLSAG Guidelines and documentation' mapperne på den fra ScanJour modtagne release-DVD fra Baseline-14.

Baggrund for dokumentet



En af de vigtige omdrejningspunkter for det tekniske review er interviews med nøglepersoner hos ScanJour (SJ), CSC og Rigspolitiet (RP). Andre hovedhjørneste består i inspektion og analyse af den POLSAG-specifikke kildekode samt evaluering af systemets arkitektur og platformens sammensætning.

Der har været afholdt interviews med nøglepersoner hos ScanJour, CSC og Rigspolitiet.

ScanJour: Dennis Lauritzen (løsningsarkitekt), Jan Olsen (de facto udviklings-lead), Lene Alhed Augenstenborg (tidligere ansvarlig for analyse-afdelingen).

CSC: Lars Bech Rasmussen, Tom Jørgensen (drift), Mikkel Nielsen og Dorte Krogsgaard (funktionel test), Peter Koch Jensen og John G Thomassen ??? (performance / tuning/ arkitektur)

Rigspolitiet: Michael Englev (funktionel test), Brian Nielsen (leverandøransvarlig), Tommy Rueskov Sørensen (ansvarlig for performanceprojektet) og Peter Greifenstein (Chefarkitekt).

Inspektionen og analysen af kildekoden har været udført sideløbende med dialog med primært Jan Olsen, Martin Glob (ekstern konsulent hos ScanJour), Torben Frandsen (ekstern konsulent hos ScanJour), Matthew Robert Graham (intern primær ressource på C++ og PL-SQL delen af POLSAG hos ScanJour) samt Johnny Sabinsky (QA hos ScanJour).



Executive Summary

POLSAG-løsningen hviler på følgende teknologier:

- ASMX – Ingen indbygget funktionalitet til transaktionshåndtering og garanteret meddelelsesaflevering samt afgrænset til XML (kan eksempelvis ikke håndtere binær serialisering).
- RPC over http – Er i al praksis begrænset til at fungere på LAN, eftersom RPC kræver at der er åbnet et meget stort antal netværksporte pga af dets brug af dynamiske porte.
- JavaScript – Fortolket kode, der er meget lettilgængeligt, fordi det ikke stiller nogen krav om disciplin/strukturering af koden. På Microsoft-området er der i al væsentlighed ingen værktøjsmæssig understøttelse af udviklingen af JavaScript-kode.

Ovenstående betragtes generelt som værende en del af et legacy pattern på Microsoft-området i og med, at de hænger tæt sammen med MSMQ, COM+, WSE og ultimative .Net 1.0/2.0. De førstnævnte daterer sig til slutningen af 1990'erne, mens .Net 1.0/2.0 daterer sig til henholdsvis 2002 og 2005.

Givet applikationens natur og omfang vækker det undren, at der tilsyneladende ikke er gjort nogen tiltag til at påbegynde en overgang til Microsofts nyere teknologier – .Net 3.x/4.0, WCF, AJAX/WPF osv. som daterer sig til anden halvdel af sidste tiår – på nogen fronter i udviklingsforløbet og fortsat ikke foreligger nogen planer herfor. Det samme gælder forøvrigt det forhold at koden er "spredt" over i alt fire programmeringssprog.

De anvendte teknologier har været af afgørende betydning for valget af løsning for så vidt angår integrationen til back-end systemer såvel som på klientsiden.

Teknologivalget har også dannet grundlag for det grundlæggende pattern, som gennemsyrrer hele løsningen: Kald sker med XML, der som typisk er direkte afledt af indholdet af skærbilledet, via RPC over http. Leverandøren fremhæver at denne pattern danner grundlag for et meget fleksibelt system under henvisning til at det bliver relativt nemmere at tilføje ekstra funktionalitet til XML'en. Det betyder imidlertid også at serversiden og klientsiden bliver hårdt koblet til hinanden (populært sagt vil hver eneste client-side ændring afføde en server-side ændring og vice versa) på en ikke-type-stærk måde, ligesom det meget hurtigt vil blive uoverskueligt om en tilføjelse skaber fejl andre steder, hvis der er flere der kalder samme funktion/metode/package. Det er i øvrigt vigtigt at være opmærksom på at den hårde kobling også gælder for nogle af de eksterne systemer (helt præcist Index2 og Bøde), idet løsningen i nogen udstrækning benytter kald som bliver direkte videre sendt af POLSAG servicelaget gennem ESB til de eksterne systemer, hvilket giver en meget tæt afhængighed mellem klienten og de eksterne systemer.

Vi vil betegne POLSAG som værende en traditionel monolitisk client/server-baseret applikation, der anvender XML og http. Der er altså på ingen måde tale om en SOA-baseret løsning, idet dette bl.a. ville kræve at der havde været benyttet løse koblinger, modularisering og indkapsling, metadata, SOAP etc.

På baggrund af review'et er det vores konklusion at POLSAG ikke på nuværende tidspunkt bibringer Rigspolitiet en komparativ fordel for så vidt angår den fremadrettede digitalisering af Politiet. Dvs. det er naturligvis muligt at udbygge applikationen med grænseflader mod nye klienter mv. men det er ikke vores vurdering at det vil være væsentligt mere økonomisk fordelagtigt end at nyudvikle. Hvis en eventuel videreudvikling skal være konkurrencedygtig med nyudvikling vil det ydermere forudsætte at der sker en



ændring af de nuværende udviklingsmetoder, idet vedligeholdelsesomkostningerne for den nuværende kode vurderes at ligge på et væsentligt højere niveau.

Rapporten indeholder desuden en lang række delkonklusioner med tilhørende anbefalinger. Vi vil særligt gøre opmærksom på mangelen på validering af systemets performance og skalerbarhed, fejlniveauet (såvidt vi har kunne få oplyst fra Michael Englev hos RP er der endnu ikke nogen indikation af at fejlkurverne udviser det nødvendige knæk), dokumentationens brugbarhed og indhold samt det meget lave niveau af automatiserede tests samt den generelle udfordring i relation til systemets testbarhed.



Teknisk review

Dette kapitel indeholder en overordnet analyse med tilhørende konklusioner af alle de områder, som er undersøgt ved inspektion eller afdækket under interviews. Detaljer og kodeeksempler er henlagt til de efterfølgende appendices.

Platform

De primære komponenter i POLSAG er:

- Windows XP (klient), Windows Server 2003 (applikationsserver)
- IE6 (klient), IIS 6 (applikationsserver)
- VS2008 (udviklingsværktøj)
- ASMX web services, WCF service for Sagsstyringreol
- Captia 4.2
- Javascript
- .NET 2 (application server), dog .NET 3.5 for Sagsstyringreol
- C++
- PL-SQL

Nogle af disse er betinget af den anvendte version af Captia, andre er kontraktuelt fastlagt.

Det basale call pattern er RPC¹ mellem browser og web server. Klienten anvender en proprietær scripting syntax til at tilgå og opdatere data. Dataudveksling og fortolkning er baseret på magiske strengværdier, XML-parsing af HTML-forms og omfattende brug af late binding².

Der observeres en meget høj grad af hård kobling mellem klient og server på grund af disse valg.

Den eneste mulighed, der p.t. eksisterer til at udføre impact-analyser af ændringer består af tekst-baserede filsystemsøgninger i henholdsvis kodebasen og SDD-dokumenterne.

Konklusion: POLSAG er baseret på en teknologisk platform der er inhomogen og har nogle år på bagen. Det bemærkes specielt at dette tjener til at gøre den væsentlig mindre produktiv og uattraktivt at arbejde med for udviklerne.

Mest iøjefaldende er bindingerne til .NET 2 og IE6. Særligt anvendelsen af ASMX web services er særdeles problematisk, da ASMX-teknologien ikke er pålidelig. Teknologien stiller for eksempel ingen garantier for hvor mange gange en afsendt besked når frem til modtageren – om overhovedet.

Bindingen til IE6 medfører binding til Windows XP som kun bliver patchet af Microsoft, når der findes kritiske sikkerhedshuller.

Løsningen er som nævnt karakteriseret ved mange små kald mellem browser og web server i en RPC stil, hvilket kan give performanceproblemer eller resultere i øget hardwarebehov.

¹ http://en.wikipedia.org/wiki/Remote_procedure_call

² http://en.wikipedia.org/wiki/Late_binding

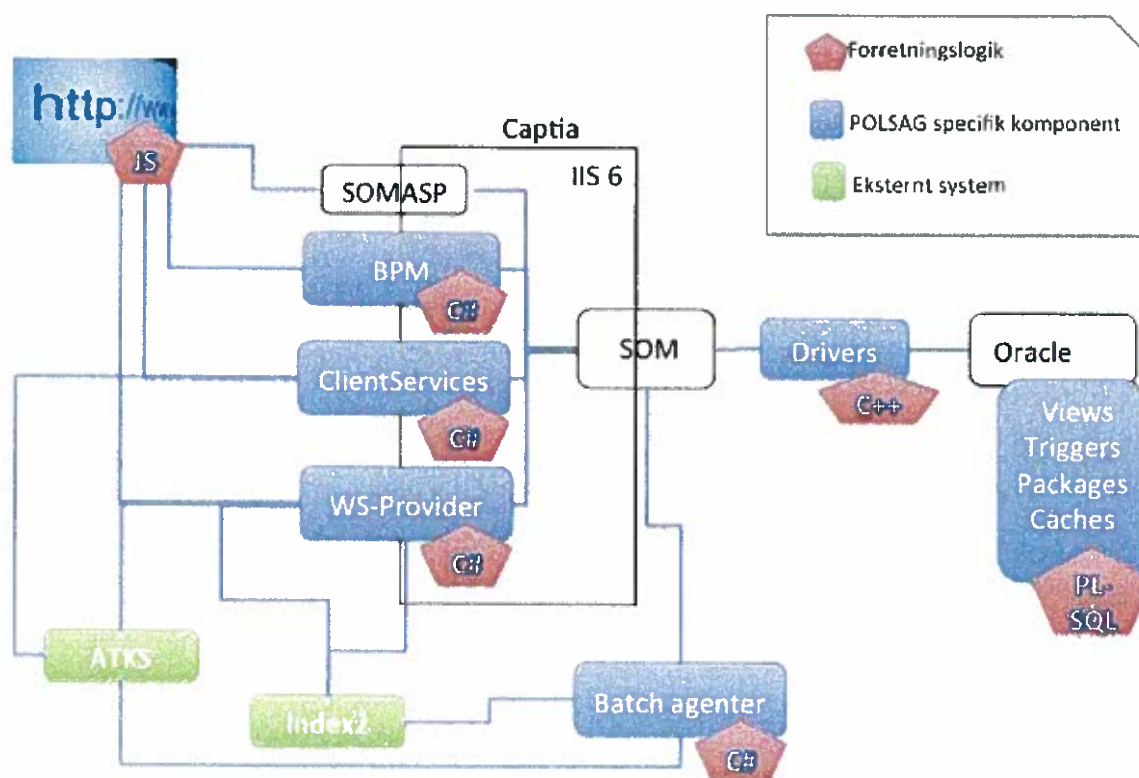


Arkitektur

Anvendelsen af Captia i POLSAG, og sammenhængen mellem de til POLSAG specialudviklede delkomponenter er illustreret i nedenstående diagram.

- **BPM** er *Business Process Manager*
- **ClientServices** er en Javascript-konsumerbar frontend til de eksterne systemer (der udstilles over HTTP-protokollen)
- **WS-Provider** er de ASMX web services som er udstillet fra POLSAG af hensyn til de eksterne systemer
- **Batch-agenter** er Windows NT services som hostes af Captia og som udfører periodiske baggrundsjobs af både opdaterende og opryddende natur
- **Drivers** er SOM-register udvidelser som anvendes til enkelt-register data-valideringer
- **'Views, Triggers, Packages, Caches'** dækker over de udvidelser til databasen som ikke nødvendigvis er tilgængelige via SOM Register-modellen, men som bl.a. finder anvendelse til performanceoptimeringer, følgeopdateringer af ændringer gennem Register-modellen, synkronisering af caches og håndtering af skema-migreringer.

De røde pentagoner angiver hvor der forefindes implementeringer af forretningslogik samt i hvilket programmeringssprog, mens de gennemsligtige bokse angiver komponenterne fra standard Captia.



Figur 1 POLSAG logisk komponent oversigt



Den øverste venstre boks angiver browser-klienten, og dennes indlejrede Javascript. Denne bør egentlig markeres som en delvis specialudviklet komponent, fordi en ikke trivial delmængde af Javascript-koden har relation til anvendelsen af hhv. BPM og ClientServices.

Overordnet arkitekturvurdering

Systemet har ingen lagdeling, når det kommer til implementering af forretningslogik. Denne implementeres i enten klienten (Javascript) eller på serveren (C#, C++ eller PL-SQL). Valget af placeringen af en given funktionalitet er generelt blevet foretaget ad-hoc af den enkelte udvikler.

I modsætning til standard Captia (som er en client/server løsning, hvor al forretningslogikken er placeret på klienten) er der her introduceret flere muligheder for placering af implementeringer. Server-side udvidelsen har været nødvendig, fordi standard Captia's transaktionsstyring-model ikke kan håndtere opdateringer til mere end et register ad gangen, og denne begrænsning ville have gjort systemet uanvendeligt til Rigspolitiets anvendelses-scenarier.

Der er ikke nogen overordnet strategi for hvordan man sikrer sig at applikationen leverer den nødvendige svartid og generelle UI-respons. Performanceproblemer forventes løst hen ad vejen når de opstår ifølge Dennis Lauritzen hos SJ, og CSC meldte i 2008, efter etableringen af performancekommisioriet, ud til RP at man ville vente med at kigge på dette område til senere (iflg Brian Nielsen). Først i foråret 2010 blev de første målinger lavet.

Konklusion: Der er ingen overordnet guidance for eller styring af, hvor og hvordan funktionalitet tilføjes til POLSAG. Givet den manglende lagdeling er funktionaliteten spredt ud over alle dele af systemet.

Strategien for hvordan man vil håndtere performance-problemer vil i nogle tilfælde vise sig at være anvendelig når det drejer sig om relativt trivielle fejl i koden. Eftersom den basale client-server kommunikation er baseret på et synkront (RPC-style) call pattern, som ikke besidder gode skaleringssegenskaber og som trækker mange ressourcer både netværksmæssigt og server-side må det forventes at det vil være forbundet med store omkostninger at udbedre denne type af performanceproblemer.

Den POLSAG-specifikke SOM-model

Den POLSAG-specifikke logiske datamodel består af 35 entiteter, som er sat sammen ovenpå den underliggende database, der består af 856 tabeller (standard Captia har 294 tabeller). Der er 542 registre i POLSAG datadictionary (standard Captia har 199 registre). Herudover er der cirka 280 specialudviklede PL-SQL views og 30 specialudviklede PL-SQL packages.

Anvendelsen af POLSAG modellen

Der er observeret en meget uensartet brug af POLSAGs SOM model spændende fra standard-Captia mønsteret med 1 skærm billede til 1 Register, over Business Process Manager-håndteringen af tvær-Register opdateringsmønstre og til views og triggers i PL-SQL som dels benyttes af performancehensyn, dels til at vedligeholde synkronisering af indblik-indstillingerne mellem personkategori- og sagsrecords.

Beslutningen om hvilket anvendelsesmønster, som skal bruges er angiveligt blevet taget på en case-by-case basis, hvilket ikke bidrager til overblikket.



Transaktionsstyring

Standard Captia har indbygget transaktionsstyring med den begrænsning at man kun kan opdatere et register ad gangen.

Den i POLSAG introducerede ASMX service applikation (Business Process Manager) løser denne restriktion ved at være i stand til at koordinere opdateringer som involverer flere registre. Der er dog ingen centraliseret styring af anvendelsen af transaktioner, hvorfor der er observeret manuel transaktionsstyring spredt rundt omkring i kodebasen (dvs. begin/commit/rollback foretages eksplicit uden nogen form for overordnet koordinering). Dette er problematisk, fordi det gør det vanskeligere (for ikke at sige umuligt) at kunne genbruge eksisterende kode i nye sammenhænge, idet man risikerer at få committet delmængder af data som i nye sammenhænge ikke er konsistente, og man bliver hårdt bundet op på den eksisterende funktions forståelse af, hvornår noget går galt.

Konklusion: Transaktionsstyringen bør centraliseres så de enkelte funktioner mv. ikke har ansvaret for at afgøre hvornår der skal foretages commit/rollback. Det er en klar svaghed i arkitekturen at der ikke er et gennemtænkt og håndhævet transaktionsstyrings-princip, særligt set i lyset af at Business Process Manager-implementeringen er introduceret netop for at håndtere transaktionsstyring.

Tekniske vurdering af integration til eksterne systemer

Al web service funktionalitet i forhold til eksterne systemer i POLSAG er bygget på ASMX-teknologien. Dette valg er taget fordi server-side løsningen er låst til .NET 2.0, og fordi det også har gjort det relativt enkelt at udstille de eksterne systemer til Javascript-klienten via ordinære HTTP-kald (å la SOMASP).

Der er ikke nogen eksplicit sikkerhedsmodel for web service grænsesnittene, og der er ikke nogen implementering af checks af data-integritet og konfidentialitet. Der anvendes ikke SSL.

Der er ikke nogen eksplicit model for håndtering af replay detection. Det er op til den enkelte udvikler at sikre mod data-korruption som følge af duplikat-beskedes (ASMX-teknologien har ikke indbygget replay detection i modsætning til WCF³).

Det har været nødvendigt at bygge et tool til at generere web service klienter ud fra WSDL⁴-beskrivelser fordi middlewarekomponenten (IBM ESB) har interoperabilitets-problemer set fra .NET for så vidt angår overførsel af datoer. ESB'en kan ikke garantere en tabsfri konvertering af UTC-datoformatet fra .NET og der er derfor blevet aftalt et custom data-format til udveksling mellem POLSAG og IBM ESB. Ovennævnte tool håndterer så denne konvertering.

Derudover bemærkes det at der er en indbygget begrænsning af længden for brugernavne. Disse kan højst bestå af 8 tegn. Rigspolitiets nuværende AD-brugere har formatet 'wrx12345' hvor 12345 er et medarbejdersnummer. Batchjobs afvikles med en SJSERVICEAGENTUSER, som er længere end 8 tegn. RP har accepteret at denne account kalder videre som SJSERVIC.

ESB'en er implementeret med en performancegaranti på et sekund. Dvs. at ESB'en højst må tilføje et overhead på et sekund til en forespørgsel.

³ http://en.wikipedia.org/wiki/Windows_Communication_Foundation

⁴ http://en.wikipedia.org/wiki/Web_Services_Description_Language



Det skal bemærkes at den mainframe-baserede ESB formentlig skal udskiftes i POLSAGs levetid, dels fordi RP ønsker et mere moderne produkt, og dels fordi driftsprisen baseret på MIPS vil være overordentlig høj, når den nuværende fastprisordning udløber.⁵

Fra et performancesynspunkt har vi ikke kunne finde noget information, som indikerer at de eksterne systemer er vurderet med i POLSAG dvs.

- Forventede svartider fra eksterne systemer (som Index2) er ikke dokumenteret.
- Der er ikke foretaget nogen analyse af POLSAGS forventede forbrug af eksterne systemer. Det tætteste vi er kommet på informationer omkring dette er, at John Thomassen fra CSC ikke regner med at POLSAG vil generere noget synderligt load. Dette er formentligt korrekt, hvis man kan antage at der ikke er en udvikler i POLSAG, der har lavet fejl og kalder de eksterne services for meget.

Det blev bemærket at POLSAG indbefatter en custom implementering af FTP-protokollen (som anvendes fra `public class POLSAG_ANM_AGENT`).

Konklusion: Integrationen til eksterne systemer er ikke pakket ind i en velafgrænset server-side komponent. De eksterne systemer er derimod eksponeret direkte til Javascript-klienten via et late-bound API, som anvender HTML-formatterede data som input og JSON-formaterede data som output. Det bliver dermed vanskeligt at sikre sig dels mod at ændringer i eksterne systemer ikke vil medføre større ændringer i POLSAG-kodebasen, dels mod et alt for heftigt ressource-træk på de eksterne systemer, fordi disse er direkte tilgængelige fra klienterne.

I forhold til data-integritet og konfidentialitet kunne man med fordel anvende SSL transport-sikkerhed for at sikre at data sendt mellem browsere og IIS samt mellem IIS og IBM ESB for at sikre mod at data kan undergå ændringer eller overvågning af uautoriserede personer/programmer.

Der er ikke noget i systemet som forhindrer identitets-forvirring overfor backend systemer, når/hvis RP overgår til en navnestandard i Active Directory, som indeholder mere end 8 tegn. Der er en indbygget 'silent failure'-funktionalitet fordi brugernavnet trunkeres, hvilket senere kan risikere at skabe ikke-sporbare logs.

Arkitekturmæssig vurdering af integration til eksterne systemer

Som nævnt ovenfor bliver al kommunikation med eksterne systemer sendt gennem ESB'en, der forestår håndteringen af format- og protokol-formatering.

Der er såvidt vi har forstået ikke udformet noget egentlig integrationsarkitektur eller strategi i POLSAG. Dette har affødt at al kommunikation med eksterne systemer foregår direkte (dog naturligvis via ESB'en) mellem de to systemer (point to point integration) på service grænsesnit, der er specielt udviklet til dette behov. Uanset hvilket systemlandskab man opererer i kan denne type integration ikke anbefales, idet den resulterer i hård kobling mellem systemerne og potentielt $n*(n-1)$ kompleksitet mellem systemerne.

Som eksempel på den hårde kobling mellem systemerne kan nævnes integrationen med bødssystemet. Forretningsbehovet er at bødssystemet skal notificeres når en POLSAG sag skifter kreds. Dette kald til bødssystemet er implementeret i syv funktioner i POLSAG, som alle vil skulle vedligeholdes. Derudover skal

⁵ Fortalt af Brian Nielsen RP, under interview



alle udviklerne være bekendt med denne forretningsregel, såfremt man implementerer anden funktionalitet i POLSAG, der ændrer kreds på en sag.

Efter vores opfattelse vil det korrekte integrationsmønster baseret på forretningshændelser give følgende flow

- Forretningsfunktion i POLSAG ændrer kreds på en sag og rejser forretningshændelse i POLSAG.
- POLSAG integrationskomponent som abonnerer på den pågældende hændelse sender en forretningshændelse til ESB'en.
- Bøde som abonnerer på forretningshændelsen på ESB'en modtager hændelsen og behandler den.

Dette integrationsmønster ville sikre afkobling mellem de forskellige komponenter og muliggøre andre abonnenter uden at der opstår behov for ændringer i det eksisterende kode.

Office integration

ScanJour har udarbejdet et antal Word-skabeloner efter Rigspolitiets ønsker. Disse installeres på alle klienter.

Integrationen med POLSAG er implementeret med nogle VSTO⁶-komponenter der kan inkluderes i skabelonerne.

Skabelonerne henter data fra Captia via SOMASP analogt til Javascript-klienten. Der er ikke nogen overordnet strategi, som sikrer at man får minimeret antallet af kald til Captia fra disse skabeloner, da hver enkelt komponent er selvstændig og foretager et servicekald. Der er dog udviklet en speciel skabelon til døgnrapporten af denne årsag.

Der er p.t. dialog omkring overlevering af opgaven med udvikling af skabeloner fra POLSAG-leverandøren til Rigspolitiet. Der er dog ikke taget nogen endelig beslutning herom endnu.

Alt aftestning af skabelonerne er manuel.

Konklusion: Der er risiko for at skabelonerne vil trække mange server-side ressourcer, helt analogt til Javascript-klienten, fordi der ikke er en fælles model for hvordan man får minimeret kommunikations-overheadet.

Der bør være særligt fokus på dette område, hvis planen om at Rigspolitiet selv skal stå for den fremadrettede udvikling af disse skabeloner bliver virkeliggjort. Dokumentskabeloner bør udvikles under skyldig hensyntagen til hvor mange server-side ressourcer de lægger beslag på, hvilket selv for udviklere af skabelon-komponenterne kan være svært at overskue.

Der bør indbygges automatiserede tests af de skabelonerne. Set i lyset af at disse muligvis skal udvikles af Rigspolitiet bør der bygges et test framework som også Rigspolitiet fremadrettet kan anvende til deres egenudviklede skabeloner.

Performance

⁶ http://en.wikipedia.org/wiki/Visual_Studio_Tools_for_Office



I forbindelse med review'et har vi fået adgang til to performancemålinger

- Den kontraktuelle test som er beskrevet i "POLSAG Underbilag C10".
- APM Test 20110518, som tester et login, søgning og døgnrapport scenarie.

CSC/Scanjour test "POLSAG Underbilag C10"

Denne test foretages i et dedikeret testmiljø hos CSC, der er baseret på 1/8 basisload. Den overordnede konklusion er at POLSAG stadigvæk ikke overholder de kontraktuelle svartider⁷. Der pågår i øjeblikket forhandlinger, da Scanjour har 3 performancepunkter de ikke kan overholde uden at udføre signifikante ændringer på løsningen.

Tabel 1 Performance tal fra 20110527 POLSAG Performance Status RP og SJ v1.0

26/5					17/5				
	SLA status	Fraktil målt mod SLA x pct (115)	Minimum SLA status	Average SLA status		SLA status	Fraktil målt mod SLA x pct (115)	Minimum SLA status	Average SLA status
Pass	22	24	35	28	Pass	20	20	28	24
Fail	19	17	6	13	Fail	21	21	13	17
Ikke målt	3	3	3	3	Ikke målt	3	3	3	3
Pass %	50%	55%	80%	64%	Pass %	45%	45%	64%	55%
Fail %	43%	39%	14%	30%	Fail %	48%	48%	30%	39%
Ikke målt %	7%	7%	7%	7%	Ikke målt %	7%	7%	7%	7%

Det skal bemærkes t CSC/Scanjour målinger er uden integration til eksterne systemer (da det er det scenarie, der er relevant i forhold til de kontraktuelle svartidskrav), hvorfor det må forudsættes at nogle af de testede operationer vil være langsommere i driftsmiljøet.

Vi vurderer at det arbejde der p.t. pågår har mere karakter af et tuningsprojekt end en performancetest. At arbejdet er koncentreret omkring de kontraktuelle test cases påvirker ligeledes prioriteringen af arbejdet. Der er dog også fundet generelle optimeringer, som tegner til at forbedre hele brugeroplevelsen af POLSAG, men der bliver også foretaget optimeringer, som udelukkende påvirker de kontraktuelle test cases.

Vi savner således en struktureret tilgang til analysen af POLSAGS performance, idet det ikke er lykket os at komme baggrunden til performanceproblemerne nærmere end følgende generelle sætninger

- Mange servicekald fra browser – Der nævnes gamle målinger på over 1000 kald for at få startsiden op). Nogle af disse kald vil formentlig forsvinde, når der omlægges til Kerberos, men der er ikke foretaget nye målinger der kan bekræfte dette endeligt.

⁷ Det skal dog bemærkes at der endnu er tvivl om det anvendte test værktøj har samme performance karakteristisk som Internet Explorer.



- Lineært voksende hukommelsesforbrug i IE6-browseren – Der nævnes 500Mb efter en times kørsel.
- Mange databasekald – Dette skyldes såvidt vi kan få oplyst primært den måde som Captia SOM fungerer på.

Det er vores anbefaling at man som absolut minimum får analyseret et repræsentativt udsnit af de mest benyttede skærbilleder med brugerinteraktion i udviklingsmiljøet helt i bund inkl. et fuldt trace af disse transaktioner kan dokumenteres.

APM Test 20110518

Denne test er gennemført i produktionsmiljøet af Rigspolitiet i samarbejde med Devoteam og dækker et login samt udvalgte søgescenarier:

- Søgning af sag på Bornholm, gerningskoder 61100-85110 og anmeld dato 02.05.2011.
- Fremsøgning af SSS (sagsstyringssag) på <journalnummer> [random 168 stk].
- Dan døgnrapport for seneste døgn i Bornholms kreds.
- Dan døgnrapport for 21/4 kl. 7:00 til 23/4 kl. 7:00 for afdeling "12vc".

Da testen udføres på produktionssystemet er det kun en delmængde af POLSAG-funktionaliteten som er blevet testes, idet der ikke må ske opdateringer af data i produktionssystemet. Testen forventes dog at give en indikation af performance og især eventuelle forbedringspunkter.

Derudover kan testen bruges til at måle systemets driftsstabilitet i form af memory forbrug i Internet Explorer, samt identificere problemer som er begrundet i brugernes forskellige geografiske placering, altså om nogle kredse har netværksmæssige problemer der kommer specielt til udtryk med POLSAG grundet den meget store brug af servicekald fra klienten.

Rapporten har følgende konklusion :

Det må forventes at svartiderne i produktion vil stige, efterhånden som der kommer flere kredse på, udover Bornholm. Hvor meget kan ikke konkluderes ud fra denne test.

og følgende anbefaling:

At der iværksættes yderligere test og tiltag for at sikre, at systemet kan klare det nødvendige load, når alle kredse er tilkoblet.

Det må ses som meget bekymrende at der fremkommer dårlige svartider i en test der kun foretager læsninger og som ikke indbefatter kald til de eksterne systemer.

Status på performance test

Det er overordentligt positivt at projektet med CSC/Scanjourns kontraktuelle performance tuning allerede er i gang med at etablere et antal egentlige automatiserede funktionelle performancetest, idet disse erfaringsmæssigt vil vise sig at have stor værdi i projektets levetid, blandt andet som regressionstest ved leverance af nye produktversioner. Her kan sammenligning af testresultater anvendes til at sikre at performance ikke forringes over tid.



Desværre kan de udførte performancetest ikke besvare spørgsmålet om, hvordan dette system vil opføre sig og opleves af brugerne i drift. De foreliggende tal tegner indtil videre et billede af at det er overvejende sandsynligt at POLSAG ikke vil blive oplevet som et hurtigt system.

Man kan heller ikke konkludere noget sikkert i relation til systemets driftsstabilitet under belastning fra de foretagne målinger, da de afviklede performancetests bl.a. ikke dækker scenarier, hvor de for POLSAG eksterne systemer bliver kaldt. Det betyder at der ikke findes nogen tal der dækker den totale brugeroplevelse.

Der er som sådan ikke nogen baggrund for at nå frem til nogen konklusioner omkring systemets ydeevne og skalerbarhed. Hverken i den ene eller anden retning. Simpelthen fordi der ikke foreligger tilnærmelsesvist tilstrækkelig information. Indtil andet er kendt må der ud fra en forsigtighedsbetragtning formodes at der eksisterer en betragtelig risiko for at det bliver nødvendigt at forlænge udrulningsprocessen, fordi det kan vise sig nødvendigt at opdatere POLSAG på vitale punkter undervejs i udrulningsforløbet.

Den eneste indikation af systemets stabilitet er den oplevelse, der har været i uddannelsesmiljøet, som dog anvender en ældre version af POLSAG. Her har der angiveligt været op til ca. 80 samtidige brugere på en installation, der består af en enkelt server (der er 20 servere i produktionsmiljøet). Indtil videre har vi ikke kunne opspore anden information omkring oplevelserne fra uddannelsesmiljøet end at dette miljø ikke har været uden vanskeligheder.

Anbefalede tiltag omkring performance på kort sigt

Baseret på projektets tidsplan og den nuværende status på performanceanalyserne har vi følgende anbefalinger, som bør effektueres uden ophold:

- Igangsæt strukturerede målinger.
- Igangsæt forebyggende og skadesbegrænsende tiltag.
- Igangsæt struktureret analyse.

Strukturerede målinger

Det anbefales, at der hurtigst muligt foretages målinger på et kørende system (enten produktionssystemet når den næste kreds kobles på eller undervisningsmiljøet).

Formålet med disse målinger skal ikke kun være at måle systemet kvalitative opførsel, men også for at etablere en fælles forståelse med udviklings- og driftsleverandøren omkring systemets performancemæssige opførsel. Målingerne skal bl.a. kunne besvare spørgsmål som

- Hvad er de meste hyppigt anvendte skærbilleder?
- Hvor mange servicekald genereres fra de mest anvendte skærbilleder?
- Hvad er de mest hyppigt kaldte services?
- Hvor mange databasekald genereres der for hver af de hyppige servicekald?
- Hvor stor er trækket på de eksterne systemer og hvad disses svartider?
- Hvis der er nogen nævneværdig belastning af databasen skal disse tunge SQL-operationer analyseres for eventuel performanceoptimering.

Skadesbegrænsende tiltag



Der er nogle brugerinitierede operationer, som potentielt er meget tidskrævende og belastende for systemet. Døgnrapporten er et eksempel som har identificerede problemer og hvor det p.t. er genstand for diskussion om den skal redesignes. Men der kan potentielt være flere andre statistikker og rapporter af en tilsvarende natur.

Døgnrapporten ses som en trussel mod POLSAG's stabilitet og derfor anbefales det at med et eller flere af nedenstående tiltag.

- Implementering af kald til rapporter og statistikker, som asynkrone operationer med et bruger interface, der sætter brugeren (eller en sysadmin) i stand til at stoppe en "løbsk" forespørgsel.
- Indsættelse af dedikerede webservere til at behandle rapporter og statistikker (dette forudsætter at antagelsen om at det ikke er databasen selv der belastes er korrekt).
- Implementering af ressourcestyring således at forespørgsler og rapporter kan begrænses i CPU-forbrug.
- Det anbefales at man som minimum i en indkørselsperiode reducerer antallet af parametre til rapporterne således at performancekonsekvenserne bliver nemmere at overskue.
- Det anbefales at der implementeres et "bestillings"-mønster, således at rapportgenereringen om nødvendigt kan scheduleres. Det bør samtidig overvejes om der tidligt på morgenen (dvs. på tidspunkter med lav belastning) kan genereres nogle standardrapporter, som for eksempel en hel afdeling kan dele fremfor at hver eneste person potentielt trækker den samme rapport.

Struktureret analyse

Der bør uden ophold påbegyndes en struktureret analyse i udviklingsmiljøet, hvor det er nemt at dokumentere al den trafik der foregår i systemet. Opgaven går ud på for udvalgte kritiske skærbilleder at optage et par brugerscenarier og så undersøge

- Trafikken mellem browser og IIS på de forskellige interaktionspunkter – Altså hvor mange kald til serveren som visningen og anvendelsen af et skærbillede genererer.
- Trafikken mellem Captia SOM og databasen.

Baseret på vores review af koden er det vores umiddelbare forventning at der vil en del overflødige kald og også en del mindre kald, der formentlig med stor fordel kan samles i færre, men større kald.

Betragtninger omkring POLSAGs performance

De performancetiltag, som hidtil er gennemført på POLSAG er af en meget tidlig og begrænset karakter. Netop performance og skalerbarhed vurderes på den baggrund at være en af de største risici for POLSAG i den helt nære fremtid.

Det er overordentligt bekymrende at man på dette tidspunkt af projektet ikke har en mere klar indikation af, hvordan systemet yder, hvordan det skalerer og hvilke komponenter i POLSAG, der er specielt ressourcekrævende og/eller belastede.

Vi er i vores arbejde med at gennemlæse koden stødt på en del kandidater for performanceforbedringer. Det er dog vores anbefaling at performanceoptimeringer altid er på baseret tal fra produktionslignende scenarier og prioriteret efter forventet anvendelsesfrekvens.



Forespørgsler gennem SOM

POLSAG er bygget oven på Captias SOM-model, som er en datamodel abstraktion oven på en database, som i POLSAGs tilfælde tager form af en Oracle-database. Ud over fordelene ved den indbyggede funktionalitet som indblik, logging og konfiguration giver det udviklerne et typestærkt C# interface til at udføre forespørgsler mod datamodellen.

Bagsiden af medaljen er (som også er en udfordring i andre O/R mapping frameworks) at udviklerne relativt hurtigt mister fornemmelsen for, hvor mange databasekald der bliver foretaget, samt at dette sjældent manifesterer sig som et problem på små datamængder på en udvikler-pc med tilhørende lokal database. Som udvikler skal man hele tiden holde sig for øje at SOM Register-API'et, ud over de oplagte database-forespørgsler i kaldene til ExecuteSearch, også foretager database-forespørgsler hver gang man skaber en instans af en Register-klasse.

Udtrykket '`new UdtrSagsdispositionRegister(this.Session)`' nedenfor giver således anledning til et (implicit) ekstra round trip til databasen.

Et eksempel:



Selve metoden er på ca. 500 linjer, hvorfor det ikke er helt ligetil at overskue, hvor mange kald der f.eks. kommer til søgning efter afgørelser.

Funktionen er også et godt eksempel på hvor svært det i det hele taget er at overskue kode, når ikke der er nogen kommentarer til at guide læseren igennem det store mængde forretningslogik, der er til stede i denne funktion.

I en klassisk client/server-arkitektur ville denne metode formentlig være klaret med et enkelt kald til en Stored Procedure i databasen. I POLSAG er noget af denne funktionalitet flyttet til applikationsserveren med flere database roundtrips til følge. Dette eksempel udgør ydermere en udmærket anskueliggørelse af udfordringen ved performanceoptimeringer af POLSAG: Hvis denne funktion bliver kaldt sjældent med små datamængder vil der næppe opstå nogen problemer og således heller ikke være et presserende behov for optimering, men hvis den bliver kaldt ofte på større datamængder vil dette give en alvorlig belastning af webserveren og måske give timeouts til brugeren.

Webkald fra browseren

Javascript-klienten foretager ganske mange kald over http til Captia-serveren. Dels for at få initialiseret skærbilleder, dels for i nogle tilfælde at trigge server-side validering af indtastede feltværdier og dels for at kunne kommunikere med de eksterne systemer. CSC betegner Captia-klienten som en 'very chatty application', hvilket dels skyldes det underliggende RPC-style call pattern, dels at der i flere tilfælde er stor forskel på hvilke data som et skærbillede skal vise og den underliggende Register-model, så man for at



viser et skærbillede er nødt til at forespørge flere registre (og dermed have flere round trips til webserveren) efter data.

Det er ikke lykkedes os at fremskaffe konkret information omkring antallet af kald fra web-browseren. En gammel test viste mere end 1000 kald for at vise første skærbillede.

Der anvendes Windows authentication mellem browser og webserveren, men der foreligger ikke os bekendt noget dokumentation for, hvorvidt Active Directory-opsætningen i Rigspolitiets domæne er af en sådan karakter at Kerberos fungerer korrekt i forhold til kommunikationen mellem IE6 og IIS6. Der er dermed en risiko for at hvert kald til webserveren trigger flere roundtrips på grund af det indbyggede automatiske fallback til NTLM authentication mellem browser og webserver.

Kodekvalitet

Gennemgangen og vurderingen af kodebasen har været foretaget med hovedfokus på vedligeholdbarhedsegenskaberne og hvor enkel den vil være at udvide og ændre fremover. De anvendte kriterier for vurderingen har været

- Læsbarhed
- Overskuelighed
- Anvendelse af objekt-orienterede best practices (undtagen for PL-SQL koden)
- Automatiseret test coverage

Der har for de tre objekt-orienterede programmeringsplatforme i høj grad været fokus på anvendelsen af best-practice principperne:

S.O.L.I.D [SOLID acronym: [http://en.wikipedia.org/wiki/Solid_\(object-oriented_design\)](http://en.wikipedia.org/wiki/Solid_(object-oriented_design))]

- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

DRY [DRY acronym: http://en.wikipedia.org/wiki/Don%27t_repeat_yourself]

- Don't Repeat Yourself

POLA [POLA acronym: http://en.wikipedia.org/wiki/Principle_of_least_astonishment]

- Principle Of Least Astonishment

C#

Der er ikke anvendt CodeAnalysis eller periodiske statiske analyser af kodebasen i udviklingsforløbet.



FxCop 1.35 rapporterer knap 5000 issues. Nogle af disse skyldes at FxCop ikke kan detektere late-binding kald. Men der er stadig mange konkrete anbefalinger fra FxCop, som udviklere vil kunne drage stor nytte af (se Code Analysis rapport for C# delen af POLSAG [vedlagt dokument POLServicesCodeAnalysis.FxCop]). Der er tale om mange violations af navngivningsstandarder og almindelige policies for robust API-design, og temmelig mange deciderede fejl (bl.a. IDisposable implementeringer, general-exceptions-catching og string-operations-problemer som ikke håndterer Culture settings). FxCop har fundet over 200 kritiske fejl, og i alt knap 3.000 fejl. Der er 29 kritiske warnings.

NB: Den autogenerated DataAccess kode er ikke omfattet af FxCop-kørslen.

De kritiske violations:

- DoNotCatchGeneralExceptionTypes – 130 violations
([http://msdn2.microsoft.com/library/ms182137\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182137(VS.90).aspx))
- DoNotRaiseReservedExceptionTypes – 30 violations
([http://msdn2.microsoft.com/library/ms182338\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182338(VS.90).aspx))
- CallGC.SuppressFinalizeCorrectly – 2 violations
([http://msdn2.microsoft.com/library/ms182269\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182269(VS.90).aspx))
- TypesThatOwnDisposableFieldsShouldBeDisposable – 1 violation
([http://msdn2.microsoft.com/library/ms182172\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182172(VS.90).aspx))
- OperationsShouldNotOverflow – 1 violation
([http://msdn2.microsoft.com/library/ms182354\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182354(VS.90).aspx))
- SpecifyCultureInfo, SpecifyFormatProvider, SpecifyCultureInfo – ialt 483 violations
([http://msdn2.microsoft.com/library/ms182189\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182189(VS.90).aspx),
[http://msdn2.microsoft.com/library/ms182190\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182190(VS.90).aspx),
[http://msdn2.microsoft.com/library/bb386080\(VS.90\).aspx](http://msdn2.microsoft.com/library/bb386080(VS.90).aspx))

FxCop warnings, som påpeger manglende oprydning:

- RemoveUnusedLocals – 81 warnings
([http://msdn2.microsoft.com/library/ms182278\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182278(VS.90).aspx))
- ReviewUnusedParameters – 89 warnings
([http://msdn2.microsoft.com/library/ms182268\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182268(VS.90).aspx))
- AvoidUnusedPrivateFields – 211 warnings
([http://msdn2.microsoft.com/library/ms245042\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms245042(VS.90).aspx))
- AvoidUncalledPrivateCode – 18 warnings
([http://msdn2.microsoft.com/library/ms182264\(VS.90\).aspx](http://msdn2.microsoft.com/library/ms182264(VS.90).aspx))

Der er 148 TODO's og 12 steder kastes en *NotImplementedException* aktivt fra levende kode. Mange TODO's ser ud til ikke at være blevet fulgt op, men det er ikke tydeligt om det skyldes manglende oprydning af TODO's eller manglende implementering.

Det bemærkes at selvom applikationen i meget høj grad anvender late binding og magiske strengværdier til at binde det hele sammen er der næsten ingen centraliserede definitioner af disse magiske strenges værdier. Der optræder mange hardkodede kopier af disse ellers helt centrale (og dermed stort set



uforanderlige) værdier, og man må dermed forvente at det bliver næsten umuligt at ændre i den logiske datamodells navne.

Derudover er der omfattende brug af hardkodede værdier i den logik som er implementeret og det vil dermed være ganske risikabelt at ændre værdier i feks stamdata.

Den statiske analyse af C#-kodebasen angiver følgende nøgletal: Der er cirka 45.000 linjers kode (auto-genereret Register-model ovenpå datadictionary er ikke medtaget), og cirka 6.300 metoder. Heraf er knap halvdelen (ca 21.500 linjer) koncentreret i 313 metoder. Med andre ord ligger knap 50% af funktionaliteten samlet i 5% af kodebasen.

Knap 20.000 linjers kode ligger i metoder som har en cyklomatisk kompleksitet på 10 eller derover (den højeste fundne cyklomatiske kompleksitet for en metode er 328, hvilket i praksis betyder at man, for at danne sig et overblik over denne metode, skal kunne overskue 328 samtidige mulige uafhængige eksekverings-veje gennem metoden).

Optimalt set bør alle metoder have en cyklomatisk kompleksitet tæt på 1, da det giver den optimale vedligeholdbarhed.

Der forekommer i størrelsesordenen 50 eksplicitte checks for null-argumenter, men der er ingen konsekvent gardering mod null-inputs i public metoder for at undgå 'Object reference not set to an instance of an object' runtime exceptions. Eksempler:

- K0033.ValidateControl
- K0432.Validate
- K5042.ValidateControl
- F0589.HentPOLSASSag

Ligeledes ses der heller ikke en konsekvent strategi for håndtering af null-returværdier fra metoder. Eksempelvis vil følgende konstruktion (fra F0589.HentPOLSASSag):

```
POLSAS.POLSASSagSagHentOutput output = POLSAS.POLSAS.SagHent(Session, input);  
if (output.FejlStruktur.Kode != "0")
```

være en kandidat til at terminere webserver-processen, såfremt kommunikationen med det eksterne POLSAS system fejler, eftersom den kaldte POLSAS.POLSAS.SagHent er implementeret som følger:

```
public static POLSASSagSagHentOutput SagHent(sjSession Session, POLSASSagSagHentInput Input)  
{  
    POLSASSag ws = CreatePOLSASSagWS(Session, "SagHent");  
    try  
    {  
        return ws.SagHent(Input);  
    }  
    catch (Exception ex)  
    {  
        ConLib.HandleConsumerException("POLSAS.POLSASSag.SagHent", ex);  
        return null;  
    }  
}
```



Et andet eksempel på dette (dvs. manglende check af null retur-værdi fra metode) ses i det vedlagte dokument F0192.cs, metoden `public string findKredsnavn(string input)`.

Det bemærkes at der er lavet en custom implementering til JSON-formattering af svar fra ClientServices til Javascript-klienten. Denne implementering er ikke unit testet, og fungerer via late-binding API'et til .NET Type-systemet. Kommentarerne i implementeringen indikerer at den ikke er helt gennemarbejdet endnu (se 'Custom JSON formatter').

Javascript

Der findes ikke gode tools på linje med FxCop og CodeMetrics fra Visual Studio til at lave statiske analyser af Javascript-kode. Nogle udviklere på POLSAG anvender JSLint til at fange de mest almindelige fejl, men der er ingen automatisering af dette.

Programmeringsparadigmet er næsten udelukkende proceduralt og der er identificeret meget forretningslogik i brugergrænsefladen. Dette er forventeligt givet Captias client/server arkitektur, men strider mod nyere arkitekturprincipper, hvor man af hensyn til vedligeholdbarhed og fleksibilitet tilstræber helt at undgå at placere forretningslogik i præsentationslaget.

Der er cirka 100.000 linjers Javascript-kode, hvoraf cirka 30.000 linjer er SDD-relaterede kommentarer og decideret udkommenteret kode.

Mønsteret med få meget store filer/funktioner fra C#-sporet gentager sig for Javascript-koden (se Histogram over antallet af kodelinjer i Javascript filer), og der observeres en del duplikering af funktioner. For eksempel afslører en manuel analyse af den client-side funktionalitet, som er lavet for at implementere 'SDD Skærm billede D0019 Fængslet V0027', hele fire duplikerede funktioner (`convertDatoTidtilDato`, `FejlKoderSomAsp`, `getPolErrMsg`, `MergeDynamicMsg`, `onInitUdforendeAfd`) med funktionelt identiske implementeringer, men med subtile forskelle som indikerer at kun nogle af duplikaterne er blevet vedligeholdt i forbindelse med oprydninger. Samme manuelle analyse afslører at der er mindst fem roundtrips til serveren via SOMASP alene for at få initialiseret viewet. Og så er der tilsyneladende en inkonsistens mellem SDD og implementering, eftersom SDD kræver at 'Id-nr D0019-017 skal filtreres via F0259', men F0259 findes ikke i, hverken Javascript- eller C#-kodebaserne.

Et godt eksempel på den manglende indkapsling af forretningslogik bag passende abstraktioner kan observeres ved at værdien af feltet "GK_HOVEDKATEGORI" indgår i if-else og switch konstruktioner omtrent 60 steder i Javascript koden.

Det er bemærkelsesværdigt at der ikke er noget egentligt udviklingsmiljø til at vedligeholde Javascript-koden. Dette betyder at når man arbejder med denne del af koden skal man rundt om Team Exploreren og filsystemet for at udføre opgaven.

Javascript-funktionalitet inkluderes i runtime-miljøet ved at placere source-filerne i et specifikt bibliotek under Captia-installationen, som Captia kigger i by default. Den Javascript-funktionalitet som skal anvendes i et view inkluderes i viewets layoutfil (standard Captia style).



Der er ikke support af at Javascript filer kan inkludere andre Javascript filer, hvilket betyder at man er nødt til at lægge alle includes i den yderste layoutfil.

Der er ingen support i POLSAG for at få kontrolleret om alle skærbilleder er konfigureret korrekt i forhold til Javascript-funktionalitet. Det kan kun detekteres manuelt ved at navigere rundt i UI og holde øje med dels browser-rapporterede Javascript-fejl, dels manglende funktionalitet når der trykkes på knapper mv.

Der er 178 layout-filer i POLSAG. Der er 1380 includes af javascript-filer i disse. Altså i gennemsnit 7 includes per layout.

Der er 418 Javascript-filer ialt. Af disse er der 84 som ikke eksplicit er inkluderet i nogen layoutfil (se 'Javascript-filer som ikke eksplicit er inkluderet i layout-filer' for en komplet liste).

Det bemærkes at der er steder i Javascript-koden, hvor der eksekveres flere på hinanden følgende opdateringer, og dermed potentiel risiko for data-inkonsistens. Eksempler:

- function rekvir() i customControl_Sigtelse.js
- function SletFrihedsberovelseF0092 i D0019_D0021.js

som begge først kalder BusinessProcessManager og dernæst kalder SOMSAP's UpdateFromUrl endpoint. Hvis det er meningen at de data som opdateres ikke behøver at være konsistente er dette selvfølgelig ikke det store problem. I modsat fald skal opdateringerne samles server-side, hvis man skal sikre mod inkonsistens. De ovennævnte eksempler er rapporteret til ScanJour, og Jan Olsen melder at det er en fejl at der ikke er blevet ryddet ordentligt op i JavaScript-koden efter introduktionen af Business Process Manager.

Der er omtrent 60 anvendelser af url'en UpdateFromUrl.asp spredt rundt omkring i Javascript-koden. Det har ikke været muligt indenfor rammerne af dette review at gennemgå alle disse, samt deres anvendelse fra andre dele af Javascript-koden for at kontrollere om der findes tilsvarende opdateringsmønstre andre steder i kodebasen. Det er vores opfattelse af den store Javascript-kodebase udgør en stor risiko for slutbrugeroplevelsen, eftersom Javascript fortolkes af klienten (dvs. en meget stor del af kodefejlene vil først blive fundet, når man anvender løsningen) og fejlhåndtering er svær i Javascript. Udvikling af Javascript er heldigvis et område hvor der er sket rigtigt meget på den tekniske front og vi anbefaler at der kigges på en teknologi som Script#⁸ og JSON⁹. Script# ville bidrage til at bringe Javascript-kodebasen ind i et statisk analyserbart sprog (C#) og dermed få en bedre værktøjsunderstøttelse til fx at kunne lave impact-analyser. Vi vurderer i øvrigt også at en standard JSON-implementering vil kunne erstatte de proprietære tiltag (JSON formatter), som ScanJour har udviklet og derved tjene til at reducere kompleksiteten.

C++

Det er ganske omstændeligt at komme til at kunne kompilere C++-kildekode. For at gøre dette skal man følge en 42-punkts installations-guide. Såfremt man har brug for at udvikle drivere er der et ekstra dokument med rettelser til 42 punkts guiden samt ekstra opsætning. Der er endvidere en del manuelle

⁸ <http://projects.nikhilk.net/ScriptSharp>

⁹ <http://en.wikipedia.org/wiki/JSON>



skridt forbundet med at skubbe en ny release ind i source control (man skal lægge binaries manuelt ind i TFS).

Der er knap 13.700 linjer C++-kode (og knap 8.300 kodelinjer excl. kommentarer).

Koden er, helt som for C#- & JavaScript-delen, procedural og langt mere data-orienteret end objekt-orienteret. Der er ingen indpakning af feltnavne i datamodellen.

Der er 348 unit tests, fordelt på 14 fixtures. Der observeres mange DRY violation i test setup koden, men anti-pattern AssertionRoulette observeres dog kun få steder.

Navngivningskonventionen ser ud til at følge SDD-mønsteret, hvorfor det kræver krydslæsning af forstå tests.

PL-SQL

POLSAG består af omkring 35.000 linjers PL-SQL kode. Der er ingen test coverage af denne del af koden.

ScanJour rapporterer at man primært anvender PL-SQL i forbindelse med performanceoptimeringer af views. De anvendes også når SOM's register-model kommer til kort, hvilket f.eks. er tilfældet, når en use case kræver at et register skal hente data fra grandchild-tabeller.

Triggers anvendes primært til at håndtere skema-opdateringer og til at holde eventuelle cache-tabeller i sync. Eksempel på en cache-tabel er 'nuværende sagsplacering', der er meget tung at beregne og som ikke var en del af det oprindelige SDD-design.

Vi fik ikke forelagt nogen dokumentation af alle de efterhånden mange tabeller, triggers og views, som ikke er en del af SOM-modellen. Den eneste oplagte vej til at danne sig et overblik er således en kombination af Oracle's dependency analyse tools og data dictionary-definitionen.

Navngivningstandarden for SQL scripts er ikke dokumenteret.

Da det er muligt at omgå SOM's indbliksstyring fra denne del af kodebasen er det udelukkende op til den enkelte udvikler at sørge for altid at repetere indblik-filtreringen fra SOM i SQL-baserede views. Skulle man få brug for at tilføje indblikfiltrering på eksisterende tabeller, enten direkte eller via nedarvning af indblik, skal man således sørge for at gennemgå PL-SQL koden for at sikre at denne ændring slår igennem alle steder, hvor disse tabeller måtte være anvendt. Denne mulighed for at omgå indbliksstyringen i SOM anvendes i øvrigt aktivt til oversendelser til anden kreds.

Konklusion

Der er ingen konsistent navnestandard for kildekoden. Konsekvensen af dette er at selv rutinerede ressourcer kan have vanskeligt ved at navigere rundt i kodebasen og dermed overskue hvor en given funktionalitet aktiveres/reageres på.

Den nuværende konvention om at anvende fortløbende nummererede navne uden domæne-betydning rapporteres at være lettere at navigere i end en tidligere konvention som forsøgte at anvende domæne-relaterede navne til objekter og metoder.



Men da der er ikke foretaget nogen oprydning af den del af kodebasen, som anvender andet end den nyeste navngivning, er navngivningen ikke konsistent. Det betyder at ramp-up tiden for nye folk vil være længere end hvis der var anvendt forretningsnære navne. Se for eksempel 'Domain-Driven Design: Tackling Complexity in the Heart of Software' (af Eric Evans)¹⁰ for en glimrende udredning af, hvordan et klart, utvetydigt og konsekvent sprogbrug bidrager meget positivt til udviklingen af software.

Det er vanskeligt at danne sig et overblik over den samlede kodebase og der er ingen tooling support til at afhjælpe dette. Nye ressourcer skal således memorere hundredevis (hvis ikke tusindvis) af ikke-intuitive navne fra SDD'er, hvis de skal være i stand navigere rundt i kodebasen på nogenlunde effektiv vis

De tre af programmeringsplatforme (C#, C++ og JavaScript¹⁰) er baseret på objekt-orienterede programmeringsparadigmer, men næsten al den gennemsete kode er af procedural karakter, og store mængder funktionalitet er samlet i ganske få 'God classes/methods', som er vanskelige at vedligeholde/videreudvikle. Der henvises til 'Kode Metrikker for C# delen af POLSAG (vedlagt dokument POLServicesCodeMetrics.xlsx)' for yderligere detaljer.

Der er ikke i forbindelse med reviewet fundet eksempler på konsekvent anvendelse af veletablerede OO-designprincipper fra f.eks. 'Gang of Four' [Design Patterns: Elements of Reusable Object-Oriented Software (af Gamma, Helm, Johnson, Vlissides)], S.O.L.I.D., DRY eller POLA.

Det forhold at der ikke er blevet udført en konsekvent oprydning af kodebasen, såvel i forhold til 2009 reviewets anbefalinger, som en del af en fortløbende proces, bidrager til at reducere læsbarheden og overskueligheden af kodebasen ganske væsentligt. Der henvises bl.a. til 'Clean Code: A Handbook of Agile Software Craftsmanship' (af Robert C. Martin)¹⁰ for en udredning af, hvorfor oprydninger (og konsekvent minimering af antallet af kodelinjer i en kodebase) bidrager meget væsentligt til en kodebases vedligeholdbarhed og udvidbarhed.

Test-strategi

Hovedparten af test-arbejdet foregår manuelt.

Der er dog opsat en system-test suite i QTP, som gennemgår de mest simple scenarier. Denne test suite bliver dog angiveligt kun sporadisk vedligeholdt, eftersom der ikke er afsat dedikerede timer til dette.

Derudover er der nogle få automatiserede tests i C#- og C++-sporene. Disse tests afvikles dog kun manuelt, når udviklerne finder det nødvendigt. Det bemærkes at C#-test suiten ikke kunne kompileres under review'et og at der dermed de facto ikke er nogen test coverage af den del af kodebasen.

Der er ingen automatiserede tests til PL-SQL koden, og Javascript-delen er kun dækket via QTP og manuelle tests.

Det kan således konstateres at det nuværende test coverage langt fra er tilstrækkeligt til at forhindre fejl i forbindelse med rettelser. Mens review'et blev udført on-site observeredes et eksempel på at introduktionen af et nyt HTML element i et view triggede en fejl i den logik, som processerede opdateringer for et andet view.

¹⁰ JavaScript er ikke et klassebaseret sprog, som C++ og C#, men understøtter objektorienterede koncepter som tillader bedre strukturering af kode end hvad der p.t. er situationen i POLSAG.



Dette betyder samlet set at enhver ændring eller ny feature i POLSAG kun kan verificeres manuelt ved hjælp af en debugger. Udviklerne rapporterer således at de debugger virkeligt meget, og indikerer at det ville være praktisk umuligt at foretage modifikationer uden en sådan. Samtidig er der dog intet som forhindrer at andre code paths end de som trigges manuelt går i stykker i forbindelse med udførelse af ændringer. Og der er ingen automatiserede tools til at bistå med at få trigget alle relevante code paths.

Konklusion

POLSAG lider i udpræget grad under manglen på automatiserede tests til kodebasen. De meget høje tilbageløbsprocenter er efter vores vurdering en klar indikator for at det er vanskeligt at vedligeholde kodebasen som nu.

Givet systemets omfang og indhold mener vi ligeledes at det er nødvendigt at Rigspolitiet foretager komplette systemtests før hver produktionssætning af nye releases for at undgå regressionsfejl. Disse system-tests må nødvendigvis også inkludere scenarier, hvor batch-agenterne har været aktiveret for at kunne verificere at de ændringer som disse foretager i data er korrekte. Det er tvingende nødvendigt at en væsentlig andel af disse tests er automatiserede, da det ikke er økonomisk og tidsmæssigt holdbart at denne del af testen tager mindst 75¹¹ manddage.

Vi anser den manglende automatiserede test coverage som værende stærkt problematisk i forhold til udviklings- og vedligeholdelsesprocessen for POLSAG, eftersom udviklerne ikke automatisk og hurtigt bliver gjort opmærksom på at de har introduceret fejl. Dette forhold sammenholdt med at det er praktisk taget umuligt at lave statiske impact-analyser af ændringer betyder at man ikke umiddelbart kan forvente at se et fald i antallet af tilbageløb, hvis ikke der sker en væsentlig ændring på dette område. Hvis ikke udvikleren som foretager en ændring har næsten komplet overblik over samtlige funktionaliteter og deres indbyrdes implicite afhængigheder i systemet er der jo så at sige næsten ingen chance for at man kan modificere den eksisterende kode uden at løbe en høj risiko for at introducere fejl.

Funktionelt testforløb

Der udføres funktionel test i alle led i leveranceprocessen.

Test hos ScanJour

Johnny Sabinski fra ScanJours interne QA-afdeling rapporterer at de mener at dække omkring 75% af al funktionalitet ved hver test cycle. QTP test suiten indgår i dette. Det meldes fra ScanJour QA at der er tilbageløb fra Rigspolitiet på omkring 20% af alle fejlrettelser, samt på 10-15% af nyudviklingen. ScanJours minimal-test (som foretages ifm hotfix-releases) tager 10-12 timer at gennemføre. Der er 3-4 personer involveret i denne test.

Vi er i tvivl om Scanjour QA's fornemmelse af testdækningen er korrekt, når vi vurderer op imod at Rigspolitiet kun mener at være i stand til at dække 2/3 af løsningen med en funktionel test, der strækker sig over 80 manddage. Begge testafdelinger medregner i øvrigt ikke andet end slutbruger-rettet funktionalitet. Batch jobs og integrationer er således ikke med i ovenstående tal. Der er dog aldrig foretaget egentlige code coverage målinger af test suiteerne, så den faktiske dækningsgrad af kildekoden er ukendt.

¹¹ Baseret på information fra telefoninterview med Michael Englev, RP



Jan Olsen (de facto dev lead fra ScanJour) rapporterer at mange af de oplevede fejl kan henføres til forskelle i opsætningen af stamdata hos hhv. Rigspolitiet og ScanJour (f.eks. organisationsopsætningen). Det skal bemærkes at denne holdning deles ikke af Rigspolitiet og ScanJour. Der hersker dog enighed om at forskelle i stamdata i de forskellige miljøer gør det svært at vide om fejl er i systemet eller i opsætningen.

Test hos CSC

CSC's tester kun de rettelser som er indeholdt per release. Der har indenfor de seneste cirka 8 måneder været 30-40 releases iflg Mikkel Nielsen.

Der testes for defects, men ikke for regressionsfejl. Det er ifølge Dorte Krogsgaard ikke altid at releases indeholder de forventede defect-rettelser, og det er heller ikke altid tydeligt at de modtagne rettelser er testet af SJ før de leveres eftersom rettelserne ikke altid virker korrekt.

Nogle batch-agenter testes først i produktion iflg Mikkel Nielsen. Lars Bech Rasmussen oplyser at det drejer sig om den fysiske flytning af data via OPC-scheduled FTP-jobs.

Mikkel Nielsen rapporterer en tilbageløbsprocent på 20-30%. CSC mener ikke der er tid nok i planen fra RP til at de kan nå at teste ordentligt, og dette vanskeliggøres yderligere af de meget hyppige releases.

Afslutningsvis foretages en testbarheds-test af CSC før en release frigives til RP. Varigheden er cirka en halv dag.

Test hos Rigspolitiet

Efter test hos CSC testes nye releases hos Rigspolitiet.

Her foretages en separat testbarheds-test som tager en dag til halvanden. Denne test indeholder et par hundrede test cases.

Herefter starter den egentlige funktionelle acceptance-test. Der har været brugt cirka 110 manddage på denne indtil videre i UD6. Et fuldt test-forløb tager cirka 75 manddage, og man mener at dække cirka to tredjedele af kernefunktionaliteten i det typiske daglige politiarbejde.

Sideløbende foretages der regressionstests, men kun hvis der er overskydende tid i kalenderen. Defect-rettelser gentestes og der rapporteres en tilbageløbsprocent på 20%. Forskelle i de involverede it-miljøer, uklare fejlrapporter og misforståelser på udviklingssiden bidrager alle til tilbageløb.

Ud af de cirka 650 defects som har været rettet i år har der været opdaget følgefejl i et sted mellem 40 og 80 tilfælde, spredt ud over alle fejkategorier. Cirka 10 fejl er sluppet helt ud til brugerne. Brugernes fejlbeskrivelser er ofte vanskelige at kommunikere videre, så der forekommer roundtrips på rettelser af bruger-opdagede fejl.

Der vurderes ikke at være nogen udsigt til at mængden af fejl er for nedadgående p.t. Der findes ligeså mange nye fejl som der rettes - men primært i kategori 2 og 3. Mindst betydende kategori 4 fejl logges ikke længere. Der ses dog kun meget sjældent kategori 1-fejl på nuværende tidspunkt.



Fremadrettet er der ikke en målsætning om at lave fuld regressionstests. Det er planen at fordele testindsatsen med 50% til forretningskritiske testdrejebøger, 25% til eksplorative tests, og 25% til at teste funktionalitet der ikke har været testet længe.

Der eksisterer et sted mellem 2.000 og 4.000 testdrejebøger. Cirka 10% af disse menes at være forældede.

Udfordringer ved nuværende testforløb

Rigspolitiet ser følgende udfordringer ved det nuværende testforløb:

- Configuration management er ikke optimal – Dette gælder ikke mindst for stamdata. Stamdataopsætningen er ikke den samme i test og produktion, hvilket både giver spildtid i testen og en risiko for at testen ikke er dækkende for systemets opførsel i produktion. Disse problematikker er især rodfæstet i den begrænsede understøttelse for change management af stamdata.
- Mere stabilt leverance-forløb – Der er mange rettelser op til skæringsdatoerne, og der låses ikke for feature-udvikling op til releases. Der ønskes en stabiliseringsfase før releases, hvilket imidlertid konflikter med at der er mange kritiske rettelser som skal med hver gang.
- Det har vist sig at forbedre processen, når Rigspolitiet hjælper CSC med deres test, idet det letter videnoverdragelsen på tværs af parterne.

Vurdering af det funktionelle test forløb

Den enorme testbyrde som påhviler Rigspolitiet sammenholdt med observationerne fra kodereview'et giver anledning til bekymring.

Sagt med andre ord er kombinationen af at fejlretning ofte giver anledning til følgefejl og det forhold at en fuldt dækkende test ikke er realistisk nærmest dømt til at resultere i at der er en del fejl, som nødvendigvis først vil blive opdaget når de rammer brugerne i produktionen.

Anbefalinger

Såvidt vi kan vurdere arbejder Rigspolitiet målrettet og professionelt med testopgaven. Det mest oplagte sted at sætte ind er at reducere arbejdsbyrden ved den funktionelle test samt øge dækningen af samme.

Rigspolitiet arbejde selv på at udvikle en strategi for udvælgelse af den test-delmængde, som giver den største reelle testdækning. Vores anbefaling er dog stadig at man bør reducere testbyrden ved at automatisere så store dele af disse tests det er økonomisk muligt.

Det må desværre forudses at være en meget stor opgave at automatisere den funktionelle test, hvorfor det anbefales først og fremmest at øge testkvaliteten under udviklingen, hvor effektiviteten er langt større og byrden med at holde testene opdateret er mindre. Det vil især være følgende typer funktionalitet der med fordel kan unit testes:

- Batch operationer
- Kontroller
- At data kan gemmes og hentes igen



Som nævnt andre steder i dette dokument er etableringen af et passende niveau af unit tests i Scanjourns kode en udfordring, idet koden ikke er designet med automatisk test for øje. Men givet omkostningerne ved den nuværende test samt udfordringerne med at vedligeholde koden ser vi ikke noget alternativ til at investere massivt i automatiske tests.

Stamdata

En stor del af POLSAG's funktionalitet er styret af konfiguration af stamdata, f.eks. ændrer brugergrænsefladen opførsel baseret på stamdata.

Det er Rigspolitiet som ejer og vedligeholder det komplette sæt af stamdata. Hidtil har produktionssætninger baseret sig på enten en komplet load af stamdata eller ingen ændringer til stamdata.

Der er nu efter første produktionssætning ved at blive fastlagt en strategi for, hvordan man skal håndtere change management. Eksisterende stamdata kan i praksis ikke slettes, og vil derfor blive de-aktiveret i stedet. For at give en indikation om omfanget af POLSAG stamdata så tager 4-5 timer at installere et komplet sæt af stamdata i en 'frisk' database.

Konklusion

Baseret på den høje kompleksitet i POLSAG stamdata vurderes det at udgøre en risiko at der ikke er fastlagt en procedure for håndtering af stamdata ændringer. Med de nuværende værktøjer i POLSAG er stamdata ændringer en manuel proces, som udføres gennem CAPTIA Configuration manager. Stamdata ændringer skal derfor manuelt flyttes gennem de forskellige testmiljøer med den deraf følgende risiko for inkonsistens til følge.

Ydermere gør mængden af stamdata det kombinatorisk umuligt at teste alle kombinationer af opsætning, hvilket kun øger risikoen omkring stamdata opdateringer.

Logning

Captia indeholder en standard use-log.

Derudover er der bygget en POLSAG-specifik såkaldt SOAPLogger, som logger alle kald til og fra ASMX web services i POLSAG. Logningen foregår asynkront (hvilket formodes at ske af hensyn til performance). Der er ikke specificeret hvilke web service-operationer som skal logges, eller hvilke delmængder af beskederne som skal logges, hvorfor alt bliver logget. Hvis logning til databasen fejler skrives der en kort meddelelse i Windows Application Event Log, som imidlertid ikke indeholder request/responses.

POLSAG-applikationen logger rigtigt meget data til databasen. Der blev logget ca. 4 millioner rækker i timen under CSC/Scanjourns performancetest. Det syntes på den baggrund at være relevant at der bliver lagt en strategi for, hvordan det forskellige logs skal håndteres inden POLSAG går i drift.

Konklusion

Der er risiko for at ikke alle ASMX-adgange til data logges, fordi logningen foregår asynkront. Der er således ingen garantier for at logningen af web service-operationer kan anvendes til at rekonstruere et fuldt hændelsesforløb.



Der eksisterer desuden ingen procedurer for, hvordan der sker backup af Windows Event Loggen, eller nogen beskrivelser af, hvordan man undgår at Windows Event Log'ens almindelige rollover-indstillinger fører til overskrivning af fejlbeskeder.

Udviklingsproces

Eftersom dette review opgave har haft til formål at afdække risici og som sådan vurdere tilstanden på POLSAG fra et kodekvalitets- og et arkitektur-synspunkt har vi ikke foretaget en egentligt review af udviklingsprocessen. For at kunne vurdere mulighederne for videreudvikling og vedligeholdelse af POLSAG har vi til gengæld udført et overordnet review af den tilgængelige udviklerdokumentation.

Overblik

Den metode der er anvendt i POLSAG-projektet er i store træk som følger:

- Scanjourns analytikere arbejder sammen med Rigspolitiet omkring skærbilleder/funktionelle områder i et workshop-lignende forløb. Resultatet bliver dokumenteret i SDD'ere. Nogen gange er workshops dokumenteret som billeder af whiteboards. Disse billeder befinder sig på udviklernes pc'er og kan i nogen grad genfindes dér.
- Scanjourns udviklere implementerer funktionaliteten på baggrund af SDD'erne.
- CSC og RP testere udvikler testdrejebøger baseret på SDD'erne.

Denne udviklingsproces er pænt sagt mystisk, idet der faktisk bliver udført en funktionel analyse, som imidlertid ikke bliver dokumenteret. Det er derefter op til både udviklere og testere at udlede den funktionelle anvendelse ud fra SDD'erne, som mange af udviklerne selv betegner som svært forståelige.

At denne proces overhovedet virker skyldes at de medarbejdere der deltager i processen har været med i projektet længe og derfor i vid udstrækning er i stand til at udlede den nødvendige information baseret på deres hidtidige erfaring.

Status på dokumentationen

POLSAG har genereret en stor mængde udviklingsdokumentation. Der er følgende nøgletal for dokumentationen:

- 154 SDD'er for skærbilleder.
- 770 SDD'er for funktioner.
- 207 SDD'er for lister.

Vi har ved stikprøver fundet at der hersker en god overensstemmelse mellem dokumenterne og den implementerede kode baseret på den overordnede struktur og de indbyrdes afhængigheder.

Vi har dog også (som forventet givet projektets omfang og type) fundet mindre inkonsistenser som f.eks. referencer i dokumentationen til funktioner, der ikke findes. Dette betyder at dokumentationen kan anvendes til støtte i forståelsen af kodens organisering, men at udviklerne tillige er nødt til at anvende fritekstsøgning for at etablere en tilbundsående forståelse for afhængighederne.

Dokumentationen (og koden) er et resultat af det skærbillede-fokuserede udviklingsforløb, som udviklingsmodellen dikterer. Dette forhold kombineret med en proces, som har meget høje krav til



detaljeringsniveau og godkendelser resulterer i en dokumentation, der først og fremmest rummer "kode nær" information.

Dette er i øvrigt en udfordring som Scanjour er bevidst om, hvorfor man i de seneste iterationer har gjort forskellige forsøg på at gøre specifikationen mindre kode-orienteret. Dette har dog vist sig at give nogen udfordringer i relation til at noget viden ikke er blevet kommunikeret til udviklerne.

Det er vores opfattelse at den foreliggende dokumentation har kontraktuel værdi, når det skal afgøres om der er udviklet det som er aftalt. Til gengæld har den meget lav værdi for nye udviklere, der skal vedligeholde løsningen, da 80-90% af indholdet i SDD'erne er informationen som normalt vil finde ved at læse kildeteksterne. Set fra et vedligeholdelsesperspektiv er værdien ved dokumentationen således meget lav, idet tvivlstilfælde alligevel vil kræve en større udredning for at afgøre om kildeteksterne eller SDD'erne har den rigtige tolkning af løsningen.

Vores anbefaling til forbedringer på dokumentationsområdet er:

Forbedret designdokumentation

Det er ikke muligt at komme med løsningen på projektets dokumentationsudfordringer i denne sammenhæng, da det forudsætter en indgående analyse af processen mellem Rigspolitiet og Scanjourns analytikere og udviklere.

Med det sagt, så har vi dog observeret at det umiddelbart virker til at dokumentationen ville blive betydeligt forbedret, hvis der blev arbejdet på flere niveauer

- Anvendelsesniveau (use case/scenarios) – Altså hvordan funktionaliteten forventes anvendt. Dette vil relatere sig direkte til de testcases som anvendes til at verificere implementeringen.
- Felt definitioner og regler – De nuværende tabeller er meget komprimerede og henviser typisk til koder eller noget indforstået "hentes/beregnes ...". Det anbefales at justere lidt i tabellerne, så pladsen bliver benyttet bedre og teksten bliver mere selvforklarende. Det drejer sig generelt om små justeringer, som virkelig bedrer overblikket (e.g. KOXYZ Check CPR nummer i stedet for blot KOXYZ)
- Logikken i funktionen – Selve logikken i funktionerne er alt for kompleks til at være indeholdt i en enkelt funktion. I nogen tilfælde får man reelt en fornemmelse af, at de store funktioner blandt andet skyldes at hvis man foretager en opsplitting kræver det endnu flere dokumenter. Et eksempel ses i Figur 2 Eksempel fra SDD. For at reducere kompleksiteten så det bliver muligt at vedligeholde er det nødvendigt at splitte funktionen op i mindre klumper og anvende UML-diagrammeringsværktøjer som sekvens- og aktivitetsdiagrammer for at give et overblik over funktionaliteten.
- Der bruges meget plads på at beskrive hvordan felter hentes og flyttes til andre objekter – Dette kunne med fordel enten flytte til et Appendiks eller pakke i C#/Javascript-klasser som dokumenteres separat, hvis der er dokumentationskrav til dette.



C0035 Output Frakendelse FrakendelseFraDato Frakendelse.FrakendelseTilDato = C0035 Output Frakendelse.FrakendelseTilDato
Hvis der for frakendelsen i Frakendelse listen findes forekomster i KlipJournalnumre listen, rekvireres de på følgende måde: Der oprettes en ny KlipSag forekomst under frakendelsen for hver forekomst i KlipJournalnumre listen ved at sætte KlipSag.Journalnummer = Sag.Journalnummer fra KlipJournalnumre listen.
Hvis der efter gennemløb af de rekvirerede frakendelser er en frakendelsesforekomst, som ikke er blevet rekvireret, fjernes forekomsten.
Alle UdloesteKlip forekomster Afgørelse → UdloesteKlip fjernes. Herefter oprettes der en UdloesteKlip forekomst for hver C0035 Output.UdloesteKlip forekomst. På forekomsten sættes: UdloesteKlip → KlipType.Kode = C0035.Output.UdloesteKlip.KlipType UdloesteKlip.AntalKlip = C0035.Output.UdloesteKlip.AntalKlip
Alle konfiskations forekomster Afgørelse → Konfiskation fjernes. Herefter oprettes der en konfiskations forekomst for hver C0035 Output.Konfiskation forekomst. På forekomsten sættes: Konfiskation → (Konfiskation).Gemingskode Kode = C0035 Output.Konfiskation.Konfiskation
Hvis Afgørelse.Indfordring=<tom>, kaldes funktion F0155 Boede.BoedeSag_Hent med Ident.IdNummer og Sag.Journalnummer.
Hvis F0155 Output.Returkode ≠ 0 og BODE-00001, Så Vis F0155 Output.Fejlmeddelelse til brugeren og afbryd rekvirering af afgørelsen.
Hvis F0155.Output.Returkode = 0, sættes Afgørelse.Indfordring=J og Boede.Boede.Journalnummer=Sag.Journalnummer.
Til sidst ajourføres KR_oversendelse. Hvis der findes en KR_oversendelse forekomst under afgørelsen, hvor KR_oversendelse → KRStatus.Kode = O eller F sættes KR_oversendelse → KRStatus.Kode = A - Afsluttet Der oprettes en ny KR_oversendelse forekomst under afgørelsen. På den nye KR_oversendelse, sættes: KR_oversendelse.KrKvitteringDatoTid = Dags dato + tidspunkt KR_oversendelse.KrOversendDatoTid = Dags dato + tidspunkt KR_oversendelse → Medarbejder.IdNummer = <aktuel brugers Medarbejder.IdNummer>

Figur 2 Eksempel fra SDD

Etablering af vedligeholdelsesdokumentation

Givet situationen anbefales det ligeledes at der udarbejdes dokumentation, som kan støtte den fremtidige vedligeholdelsesproces. Denne dokumentation skal bygge ovenpå den eksisterende kodebase og ikke være en konkurrerende beskrivelse af det samme. Vedligeholdelsesdokumentationen skal også indbefatte information om sammenhængen i systemet.

Det er en omfangsrig opgave at vende dokumentationsfokus fra instruktioner om hvordan koden skal implementeres til at give information om, hvordan systemet forventes at opføre sig. Det er ikke muligt at definere vedligeholdelsesdokumentationen her, men vi har dog følgende input:

- Det skal overvejes om Rigspolitiets brugermanual kan bruges som hjælp til at navigere i kodebasen, hvis den annoteres med numre på skærbilleder og funktioner.
- Det er vigtigt hurtigst muligt at få nedfældet den viden, der befinder sig hos Scanjourns medarbejdere. Denne analyse kan ses som første runde af den sådan aktivitet.
- Det bedste ville være, hvis man kunne organisere kodebasen på en måde, der gør det muligt at følge afhængighederne mellem modulerne direkte

Risikovurdering af udviklingsprocessen

Vi har identificeret en del risici i relation til udviklingsprocessen.



Manglende overblik over den udviklede løsning

POLSAG-systemlandskabet er komplekst. Der er mange teknologier og programmeringsplatforme i spil, der er ikke nogen lagdeling i arkitekturen, der er ikke nogen domæne-model til at definere de centrale abstraktioner i Rigspolitiets forretningsmodel og man skal krydslæse mange SDD-dokumenter og forskellige kildetekster for at finde ud af dels, hvordan en given feature hænger sammen, dels hvor mange steder den er implementeret.

En af de største risici ved en sådan situation er at løsningen udvikler sig til at blive mere og mere låst, fordi ingen kan overskue konsekvenserne af rettelser. Vi er allerede stødt på følgende eksempler på dette

- Det vurderes for en stor risiko at opdatere Captia platformen til 4.5.
- Kode kopieres, hvilket fører til dublering af stort set samme kode, fordi man ikke kender de andre anvendelser samt ikke kan overskue hvordan det testes.
- Vi har bemærket at der kun meget nødtigt foretages ændringer til POLSAG data dictionary.

Omkostningstungt specifikationsforløb

Den meget omstændelige udviklingsproces (dvs. selve processen såvel som SDD'erne), som finder anvendelse ved specificering af ny funktionalitet eller ændringer til det eksisterende, vil komme til at påvirke POLSAG fremadrettet på flere punkter.

- Udviklingen af POLSAG vil blive meget langstrakt og alt andet lige gøre business casen svagere.
- Brugerne vil opleve at der går meget lang tid før der kan reageres på deres feedback.
- Afhængigheden af leverandørens nøgleressourcer vil kun blive større henover tid.
- Det vil let kunne opstå flaskehalse i Rigspolitiets organisation.

Ingen vedligeholdelsesegnet dokumentation

Denne risiko vokser med tiden. Vi har flere gange under review'et forespurgt på information, som har vist sig at være forsvundet med medarbejdere, der har forladt Scanjour. Det vurderes på nuværende tidspunkt ikke at være muligt at vedligeholde og videreudvikle POLSAG uden de medarbejdere hos Scanjour og CSC, som har fulgt projektet gennem de sidste par år.

Udviklingseffektivitet

For at understøtte vores påstande om at det er vanskeligt og kostbart at udvikle videre på POLSAG har vi taget udgangspunkt i to eksempler fra SDD5/Polsas efterslæb

Dyretransport

Formålet med dyretransport er beskrevet som følger "Der skal kunne registreres klip for dyretransportforseelser efter de samme principper som for klip ved færdselsforseelser. Dog må de to typer forseelser ikke blandes sammen."

For at implementere denne funktionalitet er der opbygget en pakke der indeholder følgende ændringer

- Fem skærbilleder.
- 24 funktioner (hvor de 23 allerede eksisterer).
- Har resulteret i produktionen af i alt 46 opdaterede SDD-dokumenter-

Dette eksempel påviser flere udfordringer med POLSAG



- At logik og elementer er spredt over løsningen.
- At processen med at vedligeholde SDD'er ikke fungerer særlig godt med ændringer. De SDD'er vi har kigget på ved stikprøve har godt nok en ændringshistorik der nævner Dyretransport. Men det er ikke til at se, hvor meget der er ændret. Så i praksis bliver det meget svært at have mere end en ændring kørende pr. SDD.
- Det er overordentlig svært at dokumentere og vedligeholde de nødvendige funktionelle tests (både til godkendelsen af ændringen og de fremadrettede regression tests), da der ikke findes en samlet funktionel beskrivelse.

Det har ikke umiddelbart været muligt at fremskaffe informationer om estimer og det faktiske ressourceforbrug for denne opgave hos ScanJour.

Idræt

En noget mindre ændring omkring "Idrætsbegivenheder" som har følgende formål

" At kunne meddele en person generel karantæne (Forbud) mod at opholde sig ved bestemte idrætsbegivenheder indenfor en bestemt afstand. Evt. forbud gældende i bestemt tidsrum. Der kan tillige gives påbud om den sikkerhedsmæssige indretning. Efterkommes dette påbud ikke kan Politiet meddele forbud mod, at idrætsbegivenheden afholdes. Løsningen falder ind i det der er designet, derfor er det en udvidelse til det designede."

Dette eksempel viser et par aspekter ved udfordringerne ved projektet

- Ved 1-1 konverteringen har man valgt at "bunke" idrætsfunktionalitet, selvom ovenstående i virkeligheden udgør to vidt forskellige processer.
- Politiets arbejde er meget processtyret og ovenstående mangler en masse detaljer og afhængigheder, som formentlig har været genstand for debat i workshops med Politiet, men aldrig er blevet dokumenteret. SDD'en indeholder kun de kodenære konklusioner, hvor der nedenfor er



vist et simpelt uddrag

	Der oprettes en ny kontrol til afgørelsesbilledet for at udføre den nævnte kontrol.
	Når IND har fået meddelt forbuddet ændres koden for "Afgørelsestype" til 40. Dette er allerede håndteret i kontrol 1383 for afgørelse.
	I "Frakendelse" sættes koden "FK" og længden på forbuddet angives med en "Fra dato". FrakendelseType.Længdekod skal være = 3 for den nye kode, hvilket betyder, at frakendelse fradato skal udfyldes.
	I anmærkningsfeltet skrives den tilhørende tekst, eksempelvis, "Må ikke tage ophold på og ved arealer, hvor der afholdes fodboldlandskampe". Der skal ikke foretages nogen kontrol af om feltet udfyldes.
	Afgørelse skal manuel indberettes til KR. Dette håndteres af funktionen F0143 BeregnAfgørelseKR. Funktionen skal ikke rettes.
	Ved overtrædelse af forbuddet oprettes en ny sag med gemingskode 84377 – Karantæne ved idrætsbegivenheder, overtrædelse af forbud.
	På afgørelsesudskriften skal udløbsdatoen for påbuddet være anført, som i Polsas. Det antages, at udløbsdatoen er lig med Frakendelse tildato, hvorfor afgørelsesudskriften ikke skal rettes.
	Aktualitetsmarkeringer skal vises i relevante skærbilleder. Hvis der er ændringer til aktualitetsbilledet i POLSAG forudsættes det løst ifm. ændringer til Aktualiteter i ændringsanmodningen AE098.

- Det er ikke muligt for udenforstående ved at læse SDD'en at kunne verificere at reglerne faktisk dækker det nævnte formål. Der er f.eks. ingen referencer til påbud om sikkerhedsmæssige indretninger, hvilket formentlig skyldes at man opfatter det som en selvfølge at alle læsere ved at det allerede var implementeret i POLSAG.

Dette er en såkaldt lille ændring som kun dækker over 5 SDD'er. Her oplever man dog igen, hvor meget viden, der står "mellem linjerne". I ændringen næves "Ændring til frakendelsestype, SDD5 - datamodelændringer" som ikke findes i baseline-dokumentationen, hvilket gør det meget svært at vide, hvad der er ændret. Det er formodentlig relateret til den med gult overstregede tekst.

Det har ikke umiddelbart været muligt at fremskaffe informationer om estimater og det faktiske ressourceforbrug for denne opgave hos ScanJour.

Udviklingsmiljøet

I et udviklingstungt projekt som POLSAG er det vigtigt at fokusere på udviklingsmiljøet og den produktivitet det bidrager med.

Der er mange manuelle processer involveret i at udvikle funktionalitet i POLSAG:

- Captia-plattformen tilskynder til at placere forretningslogikken i JavaScript. Der findes imidlertid ikke nogen værktøjer til at lave statiske analyser af sammenhængene i den del af kodebasen.
- Captia-plattformens proprietære syntaks for, hvordan man laver views og data binding i klienten er ikke understøttet af et tilsvarende værktøjssæt til at arbejde med den proprietære syntaks.
- Den lave test coverage og det manglende fokus på dels vigtigheden af, dels produktivitetens forøgelsen man får fra dækkende unit test suite, betyder at al udvikling i dag foregår ved hjælp af debuggeren og at al funktionalitet kun kan verificeres ved hjælp af brugergrænsefladen.



- De omfattende POLSAG-specifikke udvidelser, som er lavet til standard Captia, udvider antallet af steder, hvor der er implementeret forretningslogik, hvilket øger antallet af kodelinjer og teknologier, man skal overskue for at ændre/udvide funktionaliteter.
- Den omfattende brug af late binding og u-typed data input/output formater betyder mange af de fejl, som ellers kunne være blevet fanget af compileren, i stedet vil manifestere sig som runtime fejl, når systemet er i brug.
- Den omfattende brug af magiske strenge som fundament i logikken og dataudveklingskontrakterne gør det nødvendigt at falde tilbage på fil-baserede søgninger i hele kodebasen, når man f.eks. skal vurdere hvilket navn man kan tillade sig at anvende til et nyt felt på et view.
- Det manglende udviklingsmiljø til Javascript-kodebasen betyder, at der ikke er nogen automatisk integration til Team Foundation Serveren, hvilket igen medfører at mange små opgaver som i vores optik burde blive udført automatisk nu udføres som manuelle arbejdsprocesser.
- Den kryptiske navngivningskonvention og de meget skærm billede/data-fokuserede SDD-dokumenter betyder at man er nødt til manuelt at krydsnavigere mellem kodebaserne og SDD-dokumentationen hver gang man skal arbejde med kodebasen

Det samlede indtryk af udviklingsmiljøet står i stærk kontrast til de sidste mange års trend indenfor integrerede udviklingsmiljøer, hvor der netop har været fokus på at samle så meget funktionalitet som muligt i automatiserbare processer og hvor navigation i kildekode og anvendelse af frameworks er blevet understøttet med stadig bedre værktøjsunderstøttelse.

En anden faktor som bidrager væsentligt til at forlænge/fordyre udviklingsprocessen er den ret lave grad af automatisk test coverage: Desto hurtigere en udvikler kan få feedback på om rettelser har ødelagt anden funktionalitet i systemet, desto hurtigere kan fejlen blive rettet. Dermed vil man dels undgå at andre skal bruge tid på at opdage, undersøge og rapportere fejlen, dels kan man minimere den tid som udvikleren når at bruge på at implementere den fejlbehæftede funktionalitet.

Udviklerne betegner selv sproget i SDD-beskrivelserne som grænsende til 'kryptografi'. De rapporterer som sådan at de bruger meget tid på at afkode betydningen af SDD'erne, samt at det i nogle tilfælde er nødvendigt at opspore tidligere ressourcer på projektet for at få afdækket deres tvivlsspørgsmål omkring de vitale detaljer i en SDD.



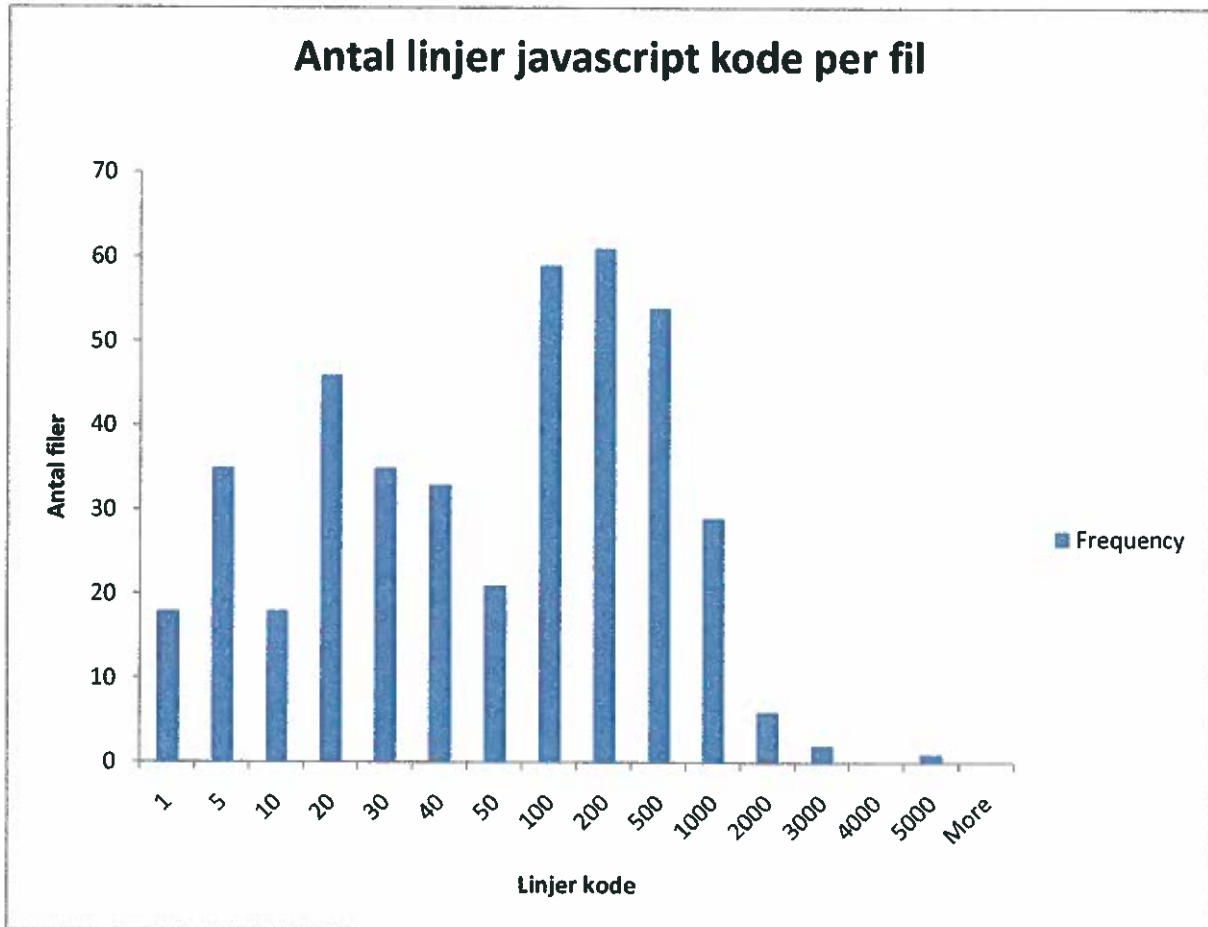
Referencer

1. Kodemetriker for C#-delen af POLSAG. Se det vedlagte dokument POLServicesCodeMetrics.xlsx.
2. Code Analysis-rapport for C#-delen af POLSAG. Se det vedlagte dokument POLServicesCodeAnalysis.FxCop.
3. Design Patterns: Elements of Reusable Object-Oriented Software (af Gamma, Helm, Johnson, Vlissides).
4. SOLID acronym: [http://en.wikipedia.org/wiki/Solid_\(object-oriented_design\)](http://en.wikipedia.org/wiki/Solid_(object-oriented_design))
5. DRY acronym: http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
6. POLA acronym: http://en.wikipedia.org/wiki/Principle_of_least_astonishment
7. Clean Code: A Handbook of Agile Software Craftsmanship (af Robert C. Martin)
8. Domain-Driven Design: Tackling Complexity in the Heart of Software (af Eric Evans)



Appendix A

Histogram over antallet af kodelinjer i Javascript filer



Javascript-filer som ikke eksplicit er inkluderet i layout-filer

A0004.js

address_etage_domain.js

afgoerelse_afgoerende_domain.js

afgoerelse_dom.moedendeanklager_domain.js

anholdelse_ansvarlig_kreds_domain.js

anholdelse_ext.anbringelses_myndighed_domain.js

anholdelse_ext.indbragt_til_domain.js



boede_client.js
contact_myndighed.kode_domain.js
contact_party.name_ref_key_domain.js
customCaptia.js
customControl_overfoer.js
customControl_sagsdisposition.js
customFunctions_sigtsag.js
customMenus_genstand.js
customReport.js
customServerSjlink.js
custom_subscription.js
D0045_F.js
D0045_M.js
D0077_F.js
D0091_C.js
D0091_F.js
D0091_M.js
D0145_F.js
D0154_F.js
D0187_M.js
D0200_F.js
D0208_F.js
D0218_F.js
dcs_client.js
eftersoegning_ext.udfoerende_myndighed_domain.js
F0072.js
F0536.js
F0539.js



F0659_VisDatafoelgeseddel.js

faengsling_ext.anbringelses_myndighed_domain.js

file_action_ou_domain.js

file_officer_domain.js

forholdsafgoerelse_afgjortfor_domain.js

genstandskategorier_elabval_domain.js

genstand_koeretoerj.nationalitettekst_domain.js

hierarchicalMenu.js

kontantdelsum.js

L0003.js

L0034_F.js

L0035_F.js

L0040_F.js

L0063_F.js

L0067_L0068.js

L0069_F.js

L0084_F.js

L0111_L0112_L0113_F.js

L0138_F.js

L0139_F.js

L0146_F.js

L0147_F.js

L0148_F.js

L0163_F.js

L0164_F.js

L0166_F.js

L0167_L0176_f.js

L0169_L0170_L0171_F.js



L0179_F.js
L0181_F.js
L0187_M.js
L0215.js
L0236_F.js
L0238_F.js
L0245_F.js
L0253_L0269_L0270_F.js
l0256.js
l0263_l0264_f.js
l0266_M.js
l0268_M.js
Lxxx_Afg_UdlKlip_F.js
patruljeresource_radioid_domain.js
personkategori_ident.nationalitet_landenavn_domain.js
record_action_ou_domain.js
record_dokument.journalnummer_domain.js
record_file_key_domain.js
S0007_D0124_F.js
S0018_M.js
S0023_F.js
S0025_F.js
S0026_M.js
sagsdisposition_patruljeenhed_key_domain.js
sagsdisposition_sagsdispositiontype_domain.js
signalement_beklaedning.sig_beklaedningfarve_domain.js
sigtelse_ext.gerningskode_domain.js
sigtelse_ext.udfoerende_myndighed_domain.js

soege_domains_foedested_domain.js

soege_domains_nationalitet_kode_domain.js

2



Appendix C Spørgsmål omkring POLSAG ClientServices

Dette appendix indeholder uddybende spørgsmål til ScanJour som opfølgning på afklaringsmødet 22/6. Besvarelsen af spørgsmålene er udført af Dennis Lauritzen (DPE), som ligeledes har indsat en række yderligere kommentarer. Al Dennis' tekst er angivet i kursiv.

Q: Kan det bekræftes at C++-drivers ikke implementerer logik som er specifik for web applikationen?

DPE: Ja det kan bekræftes. Drivers har kun adgang til data der gemmes i felterne på et register. Der implementeres forretningslogik der sikrer at data indeholder de korrekte data (både et minimum set, samt dataafhængigheder som defineret af RP). Jeg kan finde kommentarer der indeholder D0XXX – men dette er taget direkte fra SDD dokumenterne, og gør det derfor nemmere at finde SDD dokumentationen fra koden og omvendt.

Q: Kan det bekræftes at PL-SQL laget ikke implementerer logik som er specifik for web applikationen?

DPE: Jeg går ud fra at I ser bort fra PL-SQL der blot indlæser data (stamdata, fejlbeskeder, navne på skærmbilledernes labels).

Jeg har fundet 1 trigger som desværre ikke aflæser typen af genstand på andet end layout navnet. Dette er triggeren "SJTK_HIST_GENSTANDE_ROW"... denne har dog en default (if... elseif... else...) som blot sætter genstanden til en anden genstand. Det er dog muligt for en hvilken som helst applikation at udfylde layout med det nødvendige – selvom dette er ikke er så gennemskueligt i vedligeholdelsesøjemed, så dette vil kræve en kommentar (eller at vi retter op på triggeren – hvilket vil være det naturligt rigtige at gøre ☺)

Jeg har fundet i oprettelsen af nødidenter, at der i en kommentar nævnes "D0014" i triggeren "noedident_forslag_before_row", men dette er blot et felt til statistik, og kan ligeså godt udfyldes med teksten "iPhoneSmartyApp" såvel som teksten "D0014".

Jeg har ikke kunnet finde andet?

Q: Kan det bekræftes at nyudvikling skal anvende SOMASP til datalæsning og enten SOMASP eller ClientHandler.ashx til enkelt-register-opdateringer samt ClientHandler.ashx til fler-register opdateringer?

DPE: Alle skærmbilleder henter data via SOMASP, så det ville der ikke være noget mærkeligt i.

ClientHandler.ashx vil være en mulighed for de udstillede registre der er her, men der er stadig mulighed for at benytte de eksisterende BPM facader, eller at lave nye BPM facader mere tilegnet klienten eller klienttypen.

Q: Er det korrekt forstået at Javascript regerer på forretningskonfiguration i stamdata modulet.

DPE: Ja det er korrekt – det er jo kodet ud fra stamdatatyper og de tilhørende stamdataattributter og relationer hvordan de skal reagere. Det samme sker på serversiden, ligesom beskrevet i SDD'erne?

Q: Er det korrekt forstået (baseret på de vedhæftede eksempler) at BPM er tæt koblet til skærmbillederne og at den kode ikke kan genbruges.

DPE: Det er korrekt at BPM i første omgang er udviklet med henblik på at understøtte POLSAG web-



applikationen, men som jeg forsøgte at forklare i går, er der netop lavet en XML abstraktion så det er muligt for andre applikationer at tilgå de samme facader (som BPM jo egentlig er/skal være).

Q: Er det korrekt forstået at der er faktisk er forretningslogik(baseret på vedhæftede) i BPM koden

DPE: Det er korrekt at der er faktisk forretningslogik i BPM laget – dette skal selvfølgelig refaktoreres, da det ikke følger guidelines. Dvs. at BPM laget udelukkende skal bestå af facader der kalder ned i funktioner der kan genbruges på tværs af applikationen og andre eksterne systemer... ligesom vi gør med f.eks. webservices og agenter.

Vi har ledt efter egentlige arkitektur guidelines der sikrer den opsplitning som Bo præsenterede i går men har ikke kunne finde dette. Kan I pege os i den rigtige retning?

DPE: Omkring BPM, så fremgår det af vejledningen til Business Process Handlers API'et. Dokumentet ligger på DVD'en under "POLSAG Guidelines and documentation".

Omkring integrationen til eksterne systemer er det beskrevet i "Arkitekturbeskrivelse af POLSAG integration til randsystemer V0005", der ligger på DVD'en i mappen "\Overordnet POLSAG arkitektur\Integrationer", samt ren tilføjende beskrivelse i dokumentet "Generelle kravsbesvarelser for POLSAG løsningsarkitekt V0005", der ligger på DVD'en i mappen "\Overordnet POLSAG arkitektur".

Bare for at vi er sikre på at vi forstår hvordan ændringer implementeres. Kan I ikke på kort bullet form angive hvordan følgende udvidelser implementeres, men angivelse af hvad der er direkte genbrug fra POLSAG.

- Sagsoprettelse uden for POLSAG, e.g. fra smart device

DPE: Kald Sag.Anmeldelse.Opret Webservice (den findes jo allerede?)

- Indgivelse af anmeldelse fra en Borger

DPE: Kald Sag.Anmeldelse.Opret Webservice (den findes jo allerede?)

- Outsourcing af delopgaver e.g. alarmcentralen.

DPE: Der vil formentlig være tale om en anderledes arbejdsgang og som vi sagde i går, så vil det formentlig give mening at implementere nye facader der dog stadig kalder ned i eksisterende funktioner – med mindre RP mener at der skal laves nye funktioner der understøtter forretningsgangen på en anden måde – dog uden at bryde med dataintegriteten for RP's begrebsmodel.

- Fuld digitalisering af kommunikationen med domstolene (dvs. de processer hvor der i dag modtages papir materiale modtages fuldt digitalt, inklusiv kendelser, retsmøder etc)

DPE: Her ville jeg genbruge datafølgeløsningens funktioner, udvide facaden (eller lave en ny facade hvis det ønskes at begge versioner skal kunne sameksistere) med et ekstra funktionskald der tager XML'en som input, og der udvider indholdet i XML'en med det ønskede, hvide de ekstra dokumenter ud og ændre i distributionen af disse (det er utopi at tro at man kan sende flere 100 MB på e-mail med den eksisterende infrastruktur ☺)... distributionen skal selvfølgelig aftales sammen med RP og CSC.



Nedenfor følger en analyse af hvordan kode-basen i client services er struktureret. Hvis der er nogle misforståelser i analysen bedes I også kommentere disse.

Analyse

ClientServices består teknisk set af en udstilling af følgende funktionalitet i et browser-konsumerbart format:

- Udstilling af de eksterne systemer over HTTP, så disse kan kaldes fra webapplikationens Javascript-libraries
- Udstilling af Business Process Handler-funktionalitet¹² over HTTP (BPHandler.ashx)
- Udstilling af fler-register Create/Update/Delete operationer over HTTP (ClientHandler.ashx)

Ingen af de ovennævnte interfaces er typestærke.

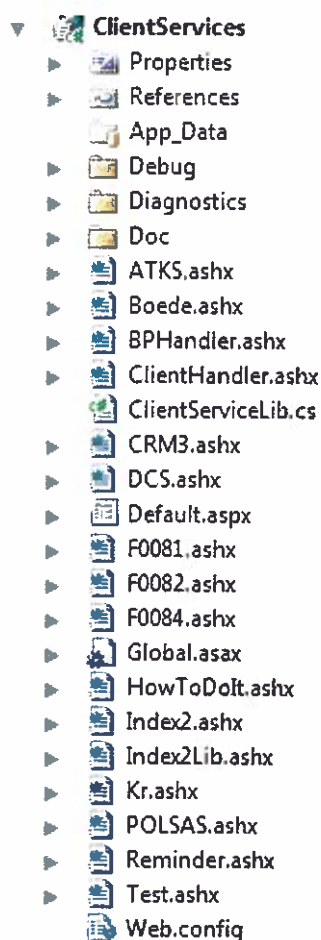
Alle er bygget til at fungere på en HTTP-klient som post'er en HTML FORM med felter, som skal have navne der passer med de skærbilleder som er i POLSAG-webapplikationen.

DPE: Som jeg forsøgte at forklare i går er det korrekt at der fra klienten mange steder blot sender HTML FORM objektet – dvs. det "serialiseres" til blot at være en XML repræsentation af HTML FORM objektet. Denne XML skal selvfølgelig overholde en struktur og det er korrekt at navnene er dem der passer i skærbillederne, men det er en ret simpel struktur (<Element id="navn på felt">Tekstbaseret dataindhold</Element>) og XML er netop den valgte abstraktion for at gøre det muligt for nye klienter at tilgå samme funktionalitet. I Jeres konklusion var det udtrykt at dette ikke er egnet til digitalisering ved f.eks. mobile applikationer, men da det blot er XML er det absolut ikke uegnet til at bruge for f.eks. mobile applikationer. Jeg vil medgive at hvis det skal udvikles af eksterne parter vil det kræve noget specifik dokumentation, der nemt kan autogenereres ud fra skærbillederne – netop fordi skærbillederne også blot er XML.

Udstilling af de eksterne systemer over HTTP

ClientServices indeholder p.t. følgende HTTP-protokol endpoints (BPHandler og ClientHandler er behandlet i de efterfølgende afsnit):

¹² Business Process Handlers er det som "populært" benævnes Business Process Manager (BPM).



Udstilling af Business Process Handler-funktionalitet over HTTP (BPHandler.ashx)

Implementeringerne af Business Process Handlers anvender primært det typestærke billede af registermodellen til at manipulere data.

Der er 531 steder i Business Process Handler frameworket, hvor man er direkte afhængig af at HTTP POST (DPE: Se tidligere kommentar omkring "serialiseringen til XML format") fra klienten respekterer POLSAG webapplikationens HTML-struktur. Dette er fordelt på ca. 110 Business Process Handler-implementeringer (ud af i alt ca 130 business process handlers). Disse tal er fremkommet ved at undersøge, hvor der anvendes en XPathNavigator (DPE: Som netop blot er et XML API fra Microsoft til at navigere i XML?) til at trække data ud af den indkommende HTML FORM (DPE: Nu XML) og anvende de ekstraherede data. De eksisterende Business Process Handlers kan af denne grund ikke forventes genanvendt til andre klienter end POLSAG web applikationen.

DPE: Her er jeg ikke enig jfr. tidligere kommentar omkring "serialiseringen" til XML format og muligheden for at autogenerere dokumentation.

Den generelle tilgang til BPM er at man som klient kalder en facade (ikke typestærk), der kalder videre ned i funktioner (typestærke). Med denne opdeling har man også muligheden for nemt at lave flere facader (også



typestærke til klienter der understøtter dette), men genbruge hele den underliggende funktionalitet, blot man følger signaturen til funktionerne.

Og som vi sagde i går, så vil der være eksempler på at det ikke er udført korrekt og stadig skal refaktoreres visse steder (oprydning).

Udstilling af fler-register Create/Update/Delete operationer over HTTP (ClientHandler.ashx)

Implementeringen af ClientHandler.ashx anvender standard-Captias streng-baserede API til at manipulere data, samt en POLSAG-specifik formatterings-syntaks for, hvordan man skal adressere de registre og felter i disse som skal manipuleres.

Når man anvender ClientHandler.ashx API'et er der ingen anvendelse af de server-side C#-valideringer, som man får med gennem Business Process Manager interfacet (DPE: Der er de C# valideringer der er implementeret på de udstillede klasser igennem dette API). C++-driver valideringer kører stadig fordi databasen tilgås via standard SOM API, men man skal implementere al forretnings-proces- og data-validering client side, hvis ikke funktionaliteten i C++-driveren er dækkende for den konkrete use case (Eller benytte BPM som den meste funktionalitet netop blev implementeret i som reaktion på egen erkendelse og dette berkræftet i Triforks review).

Data skal stadig hentes via SOMASP, eftersom ClientHandler.ashx ikke udstiller Read-access til SOM registre. Det vil sige at nye klienter på POLSAG de facto er bundet til en browser (fordi dataadgang er over HTTP) (DPE: Det er da ikke nødvendigt at være browser for at kommunikere over HTTP? Langt hovedparten af webserviceløsninger benytter da SOAP over HTTP? Jeg er godt klar over at det ikke er SOAP der benyttes her, men HTTP er jo blot en transportprotokol som benyttes af flere overliggende standard dataprotokoller – XML (som vi benytter, om end lidt proprietært i formatet) kan også transporteres via HTTP.) og man skal direkte anvende standard Captias SOMASP ESDH-rettede API til udviklingen (Som nævnt tidligere vil dette være naturligt – alle skærm billeder benytter SOMASP til at hente data til især renderingen – endvidere giver det muligheden for at angive tekstbaserede skabeloner (enten simple skabeloner deklareret i URL'en som en streng, hvilket er yderst fleksibelt for klienten, eller både simple og komplicerede skabeloner gemt som en tekst-/HTML-/XML fil, hvilket selvfølgelig er bundet til det der er implementeret i filen). Skabeloner igennem SOMASP benytter mergehandler til at hente data, som vi snakkede om den aller første dag for reviewet.

POLSAG bidrager således med en teknisk løsning på opdatering af flere registre samtidig, samt de C++-drivers som knytter sig til de enkelte registre (alle registre der kommer i spil vil aktivere drivers i alle tilgædelser af registrene).

Der er 13 funktioner i Common Funktions-bibliotek, hvor HTML (XML?)-strukturen af input data er i sving. (DPE: Jeg har ikke kigget nærmere på dette, men det lyder som om at der også her er behov for lidt oprydning!)

Eksempel fra en Business Process Handler

Nedenfor er oplistet en del af den ca 1250 linjer lange Business Process Handler

F0015Omjournaliser_BusinessProcessHandler implementering. (DPE: Jeg er enig i at dette er et grelt eksempel der trænger til refaktoring – både som opdeling i flere funktioner (1250 linjer er bare for mange), samt adskillelse mellem BPM facade (ikke typestærk) og funktion (typestærk). Så ja der er stadig behov for oprydning, men dette eksempel er heller ikke i tråd med hverken god kodeskik eller guidelines. Det



lagde vi heller ikke skjul på i går at der stadig er behov for).

Implementeringen er helt afhængig af at HTML input-strukturen/feltnavne/attributnavne er identisk med POLSAG webapplikationens skærbilledet/skærbillederne, som kalder denne Business Process Handler.
(DPE: Jeg håber det er nok at referere til det jeg har skrevet tidligere omkring "serialisering" til XML og kommentarer omkring dette).

Bemærk i øvrigt at uddraget herunder kun udgør en mindre del af den omtalte Business Process Handler.