UiO : Universitetet i Oslo

KANDIDAT

12

PRØVE

ITEVU4330 1 Robusthet i store og komplekse software systemer

Emnekode	ITEVU4330
Vurderingsform	Hjemmeeksamen
Starttid	07.12.2022 09:00
Sluttid	16.01.2023 22:59
Sensurfrist	--
PDF opprettet	20.02.2023 14:11

Seksjon 1

Oppgave	Tittel	Oppgavetype
1	Innlevering	Filopplasting

1 Innlevering

Erstatt med oppgavetekst.

Din fil ble lastet opp og lagret i besvarelsen din.

Last ned | Fjern | Erstatt

Filnavn:	ITEVU4330 prosjektbesvarelse - kandidat 11 og 12.pdf
Filtype:	application/pdf
Filstørrelse:	1.05 MB
Opplastingstidspunkt:	12.01.2023 15:08
Status:	**Lagret**

Robusthet i UiOs
Tjenester for Sensitiv Data

Kandidatnumre: 11, 12

Prosjektoppgave for
ITEVU 4330 Robusthet i store komplekse software systemer

UNIVERSITETET I OSLO

12.01.2023

Introduksjon

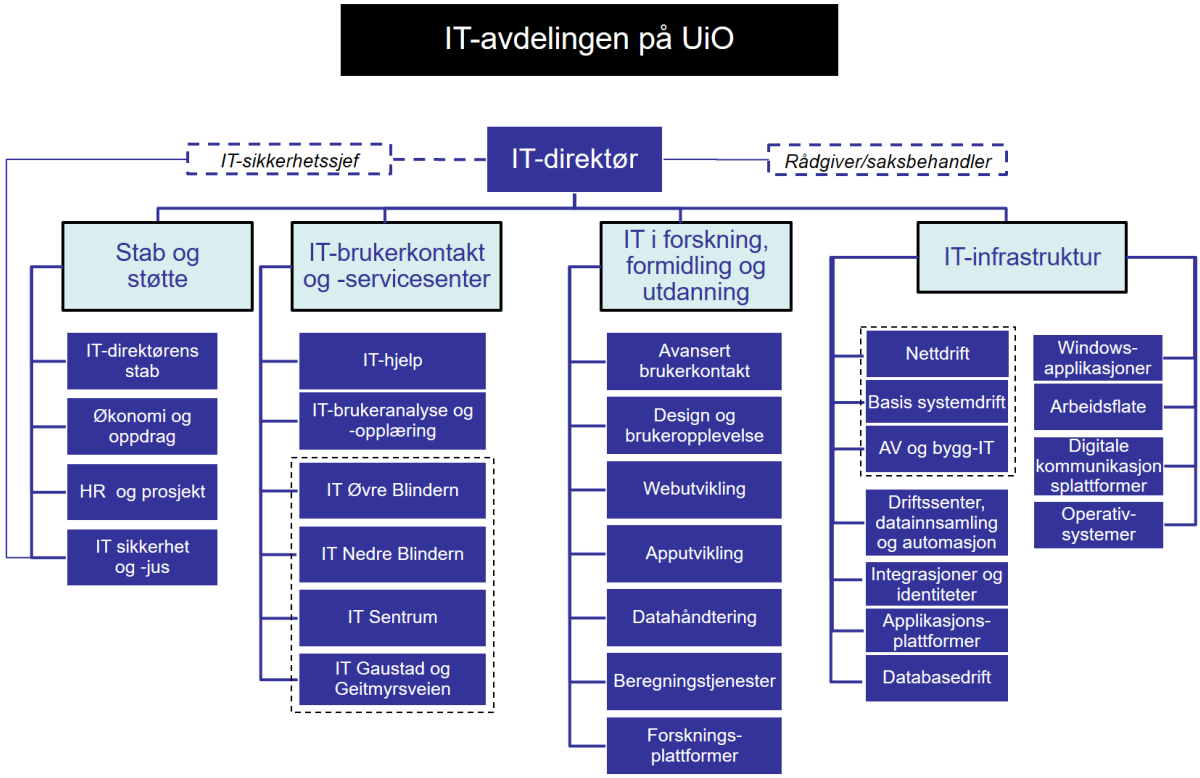
I denne oppgaven tar vi for oss forhold som påvirker robustheten i software-systemet bak Tjenester for Sensitiv Data (TSD). Spesifikt ønsker vi å undersøke følgende problemstilling:

På hvilken måte jobber UiOs IT-avdeling, spesifikt underavdeling for IT i Forskning, Formidling og Utdanning for å levere robuste IT-løsninger som dekker brukernes behov, og i hvilken grad er arkitekturen til Forskerplattformene UiOs IT-avdeling leverer fleksibel nok til å integrere ny funksjonalitet uten å påføre den underliggende arkitekturen signifikante endringer?

Før vi går videre er det på sin plass å diskutere hva vi mener med robusthet i software-systemer. Det kan være en rent praktisk evne til å tolerere feil som oppstår i systemet, eller hvordan systemet takler feil input f.eks. via APIene, til mer høynivå definisjoner rundt systemets evne til å takle eksterne endringer. For denne oppgaven ser vi på robusthet fra flere vinkler. Vi velger å definere robusthet i et software-system som systemets evne til å kunne integrere ny funksjonalitet og implementasjon på ny infrastruktur uten å påvirke systemets integritet. Især er det relevant å se på endringer som kan tenkes å påvirke systemets arkitektur og i så fall hvordan arbeidet relatert til slike endringer har foregått. I denne oppgaven vil vi også benytte robusthet i software systemet som en forutsetning og delvis definisjon på kodens kvalitet, i tillegg til robusthet i et software team som teamets evne til å operere agilt og produsere robust kode.

Vi gjør dette ved først å se på hvilket bakteppe TSD oppstod i, før vi tar for oss problemstillingen mer i detalj, avgrenser oppgaven, og går gjennom valg av metode og relevant litteratur. I seksjon for funn og diskusjon ser vi på hvilke endringer som er gjort i TSD for å øke graden av robusthet. Vi ser videre på eksempler på ny implementasjon og funksjonalitet som kan benyttes som et mål for hvor robust systemet er blitt. Til slutt konkluderer vi med noen avsluttende bemerkninger.

Bakgrunn



Figur 1 Organisasjonskart til IT-avdelingen på UiO

Universitetets IT-avdeling (tidligere USIT[1]) har 350 årsverk fordelt på 4 underavdelinger og 28 seksjoner, se Figur 1. IT-avdelingen leverer og støtter IT-tjenester og -løsninger for Universitetet i Oslo, i tillegg til universiteter, høyskoler og forskningsmiljøer ellers i Norge. Dette gjelder IT som støtter de to hovedområdene utdanning og forskning, i tillegg til løsninger for administrative oppgaver (lønn, innkjøp osv).

Denne oppgaven tar for seg software-systemer som i hovedsak leveres av 3 forskjellige seksjoner som jobber tett sammen i IT-avdelingen: Seksjon for Beregningstjenester, Seksjon for Forskningsplattformer og seksjon for Webutvikling. Alle seksjoner er del av Underavdelingen IT i Forskning, Formidling og Utdanning, og alle er involvert i tjenestene og infrastrukturen rundt Tjenester for Sensitiv Data (TSD)[2], Educloud[3] og Nettskjema[4] som denne oppgaven omhandler.

Hovedoppgavene til seksjon for Beregningstjenester er å levere og drifte lokal og nasjonal e-infrastruktur for beregning og lagring, i form av High Performance Computing (HPC) infrastruktur, i tillegg til mindre clustere og "stand-alone" servere for forskjellige forskningsgrupper. Dette innebærer blant innkjøp og installasjon av hardware, installasjon av

[1] 01.01.2023 trådte omorganiseringen av UiOs IT organisasjon i kraft. USIT er slått sammen med deler av lokal IT på UiO og hele organisasjonen heter nå IT-avdelingen.
[2] https://www.uio.no/tjenester/it/forskning/sensitiv/
[3] https://www.uio.no/tjenester/it/forskning/plattformer/edu-research/
[4] https://www.uio.no/tjenester/it/adm-app/nettskjema/

OS og konfigurasjon og installasjon av vitenskapelig software, samt brukerstøtte relatert til bruk av både infrastruktur og software.

Seksjon for Forskningsplattformer har ansvar for to plattformer for leveranse av tjenester. Begge plattformene leveres ut fra infrastruktur levert og forvaltet av seksjonene for Nettdrift (nettverk), Basis systemdrift (virtualiseringsplattform, virtuelle desktoper, osv.), Beregningstjenester (HPC-anlegg, spesialservere med akselleratorer som GPU, FPGA, osv), samt en rekke andre seksjoner i IT-avdelingen. Den viktigste plattformen seksjon for Forskningsplattformer leverer i dag, er den egenutviklede Tjenester for Sensitive Data (TSD) som er integrert med et eget sikkert HPC-anlegg, Colossus. TSD startet i 2014 og har per dags dato over 2000 aktive prosjekter og over 8000 brukere fra til sammen 80 norske institusjoner, alt styrt av 14 fulltidsansatte. TSD tilbyr en løsning som oppfyller lovens strenge krav til behandling og lagring av sensitiv data og er tungt i bruk av blant annet Helsesektoren. Nylig har (deler av) TSD sine løsninger blitt integrert i et annet system: Educloud med det tilhørende HPC-anlegget Fox. Hvordan denne integrasjonen har fungert, og hvordan den opprinnelige kodebasen er blitt skrevet om til å bli mer robust, er to av hovedpunktene vi ønsker å undersøke i denne oppgaven.

IT-avdelingens egenutviklede Nettskjema vil også bli nevnt. Nettskjema er en webbasert tjeneste for å bygge svarskjema i hovedsak for spørreundersøkelser, men tjenesten benyttes også for å levere lyd- og bildeopptak. Løsningen er utviklet for å kunne håndtere sensitiv data; da knyttes skjemaet opp til et prosjekt i TSD eller Educloud, og svarene sendes direkte inn til sikkert (prosjekt-) område, der de kan behandles videre.

Det er viktig å nevne at utvikler-miljøene ved Universitetets IT-avdeling er en sammensatt gruppe med medlemmer som består av folk fra tradisjonell IT-bakgrunn, men også med et ikke ubetydelig innslag av medlemmer uten en typisk IT-bakgrunn. Dette er blant annet folk fra en akademisk naturvitenskapelig bakgrunn fra f.eks. kjemi eller fysikk, eller fra andre områder innen akademia. Denne sammensatte gruppen folk fører til mye innovasjon og ganske frie tøyler sammenlignet med mer tradisjonelle IT-selskaper. Det kan til tider skape et noe kaotisk utviklingsmiljø og implementasjon av løsninger som i første omgang har noe uklare arkitekturvalg, som igjen kan skape komplikasjoner senere. Vår hypotese er at et slikt miljø også er meget agilt og kan respondere raskt på nye brukerkrav og ny teknologi man ønsker å prøve ut. Kan en slik sammensetning av ansatte tilføre ekstra verdi til IT-organisasjonen - og kanskje spesielt i underavdeling for IT i Forskning, Formidling og Utdanning som selv utvikler, tilpasser og drifter løsninger som ikke er mulig å kjøpe "off the shelf"?

Problemstilling og avgrensninger

I denne oppgaven ser vi på følgende problemstilling:

På hvilken måte jobber UiOs IT-avdeling, spesifikt underavdeling for IT i Forskning, Formidling og Utdanning for å levere robuste IT-løsninger som dekker brukernes behov, og i hvilken grad er arkitekturen til Forskerplattformene UiOs IT-avdeling leverer fleksibel nok til å integrere ny funksjonalitet uten å påføre den underliggende arkitekturen signifikante endringer?

Spesifikt ser vi på tre forhold som eksempler på endringer og undersøker hvordan disse påvirker arkitekturen:

- fra en Minimum Viable Product (MVP) - emergent arkitektur refaktorert til en mer robust Service Orientedet Architecture (SOA) basert arkitektur
- duplikasjon av arkitekturen til en ny infrastruktur
- ny funksjonalitet på toppen av infrastrukturen

Med tanke på tittelen til dette erfaringsbaserte kurset: "Robusthet i store, komplekse software-systemer" og eksamenen denne oppgaven skal besvare, er det naturlig å se på software-systemer levert eller konsumert av UiOs IT-avdeling som begge besvarelsens forfattere er ansatt i. Man kan diskutere hvorvidt et system som TSD, systemet vi undersøker i denne oppgaven, kan regnes som stort og komplekst om man sammenlikner med andre systemer som skal håndtere sensitive data innen norsk helsesektor. Vi vil påstå at tjenestene som leveres ved hjelp av TSD klart er på nivå med tjenester levert fra svært store, komplekse systemer i helsesektoren og UH-sektoren i Norge, særlig når det gjelder kompleksitet. Det er også det største og mest komplekse systemet utviklet og levert av IT-avdelingen. Slik sett kan vi anse TSD som stort og komplekst nok til at oppgavens problemstilling kan være av interesse også utenfor IT-avdelingens fire vegger. I tillegg er litteraturen, metodene og teorien som omhandler software-arkitektur ikke bare relevant for de største systemene, men også for noe mindre systemer som TSD.

Metode

For å undersøke problemstillingen har vi intervjuet eller samtalt med 5 personer fra i alt 2 seksjoner ved IT-avdelingen: den opprinnelige sjefsarkitekten til TSD som nå har tilknytning til både seksjon for Beregningstjenester (BT) og Forskningsplattformer (FP), to utviklere i dagens TSD-team tilknyttet FP, et uformelt intervju med nåværende sjefsarkitekten i TSD[5] tilknyttet FP samt et uformelt intervju med en senior HPC infrastruktur-arkitekt i BT[6]. Vi har gjennomført 3 av de 5 intervjuene inspirert av K.R.E.A.T.I.V metode utviklet av Asbjørn Rachlew med andre (Bergestuen et al., 2020), en norsk variant av den opprinnelige P.E.A.C.E[7] metoden (McGurk et al., 1993; Clarke & Milne, 2001) utviklet av UK Home Office for operasjonelt politi i England og Wales. K.R.E.A.T.I.V teknikken er basert på 7 hovedområder (Kommunikasjon, Rettssikkerhet, Etikk og empati, Aktiv bevisstgjøring, Tillit gjennom åpenhet, Informasjon og Vitenskapelig forankring) og ble utviklet for å forbedre politiets avhørsteknikk slik at intervjuer unngår å lede intervjuobjektet i den ene eller andre retningen. Intervjuprosessen skal ifølge KREATIV metode gjennomføres ved hjelp av 6 steg: 1) Planlegging 2) Kontaktetablering 3) Fri forklaring 4) Sondering 5) Avslutning 6) Evaluering. I vårt tilfelle var det ikke nødvendig å følge metoden stringent, men det viktigste hovedmomentet har vi fulgt, nemlig i størst mulig grad la intervjuobjektet fritt redegjøre rundt saken vi undersøker. På forhånd hadde vi planlagt hva vi ønsket å belyse under intervjuet (planlegging) - dette formidlet vi på en overordnet måte til intervjuobjektene uten å gå for mye i detaljer, ved stort sett å holde oss til vår problemstilling, og forklare at vi ønsket at intervjuobjektet så fritt som mulig forteller om forhold hen anser som relevant og interessant. Vårt hovedmotiv for å gjennomføre intervjuene på denne måten var for i så liten

[5] Grunnet sykdom kunne ikke det planlagte formelle intervjuet gjennomføres som planlagt

[6] Rakk ikke å delta i formelt intervju på grunn av reisevirksomhet

[7] P.E.A.C.E: Preparation and planning; Engage and explain; Account, clarification and challenge [Account]; Closure; and Evaluation).

grad som mulig å styre samtalen inn mot forhåndsantakelser vi måtte ha, og gi intervjuobjektet mulighet til å belyse saker og forhold vi i utgangspunktet kanskje ikke hadde tenkt på. Der det var nødvendig fulgte vi opp med oppfølgingsspørsmål eller spørsmål for å undersøke om vi hadde forstått forklaringen på et korrekt vis (sondering). Etterarbeidet med intervjuene har bestått av transkribering, og re-etablering av kontakt for oppfølgingsspørsmål.

I tillegg til intervjuer har vi satt oss inn i relevant litteratur rundt temaet agil og robust arkitektur. De viktigste funnene fra litteraturen er oppsummert under.

Teoretiske perspektiver

Problemstillingen, som gitt i avsnitt Problemstilling og avgrensning, søker å finne svar på om arkitekturen og infrastrukturen som leveres av IT-avdelingen er fleksibel nok til å håndtere endringer, altså om den er agil og robust nok. For å kunne besvare problemstillingen må vi (i) forstå hva som defineres som IT-arkitektur, (ii) forstå hva som ligger i fleksibel og robust IT-arkitektur og (iii) forstå hvordan et software team bør operere for å bidra til (ii).

Når det gjelder definisjon av software-arkitektur velger vi å benytte (Waterman et al., 2015, 348): *We define software architecture as "the set of significant decisions about the high level structure and the behaviour of a system" [..], where 'significant' is measured by the cost of change [..*]". Dette betyr at mindre endringer som kan ha implikasjoner på arkitekturen ikke er så viktig når vi diskuterer IT-arkitektur, dette for å avgrense begrepet og ikke sammenblande det med de mer daglige, regulære endringene av softwaren.

For forståelsen av fleksibel IT-arkitektur er (Bloomberg, 2013) mest interessant, men også (Fielding, 2000) kan gi et interessant innblikk i hvordan en arkitektur som kanskje er den største og mest fleksible til dags dato er bygget opp. Når det gjelder hvordan et software team bør operere, benytter vi spesielt (Martini & Bosch, 2016) og (Waterman, M., & al, 2015).

I det følgende oppsummerer vi teori og litteratur som vi har funnet relevante for oppgavens problemstilling.

"***Design Patterns***" (Gamma, E., & al, 1995), skrevet av "the gang of four" innen objektorientert programmering, er en katalog av 23 designmønstre forfatterne har funnet gjentatte ganger i forskjellige objektorienterte systemer. Disse mønstrene er "beskrivelser av kommuniserende objekter og klasser som er skreddersydd for å løse et generelt design-problem i en spesiell kontekst". Mønstrene i katalogen er delt opp etter tre formål:

- *Creational patterns* omhandler prosessen rundt objekt-kreering,
- *Structural patterns* tar seg av komposisjonen av klasser eller objekter,
- *Behavioral patterns* karakteriserer hvilke måter klasser eller objekter interagerer og fordeler ansvar.

Hvert mønster beskriver hvilke omstendigheter der mønsteret er relevant, når det kan benyttes i forhold til andre design-begrensninger samt konsekvenser og avveininger når mønstret implementeres i et større design. Det er viktig å merke seg at ingen av designene i boka er eller var nyskapende, mange av mønstrene hadde allerede lenge vært brukt av utviklere individuelt før boka. Det nye var at (Gamma, E., & al, 1995) satte navn på

mønstrene og systematiserte dem på en slik måte at utviklere kunne plukke opp mønstre og teams av utviklere kunne lettere forklare og forstå designvalg som ble tatt.

Boka beskriver blant annet hvordan man bør designe systemet slik at det er robust i forhold til å forutse nye krav og endringer i eksisterende krav og lister også opp åtte vanlige grunner for å foreta redesign samt hvilke design-mønstre de anbefaler for å løse dem (Gamma, E., & al, 1995, side 24). Selv om boka nærmer seg tretti år gammel er fortsatt svært mange av disse som er høyst aktuelle, så som

- *Avhengighet av spesifikke operasjoner* - fører til hardkoding av forespørsler som gir mindre portabilitet (designmønstre: Chain of Responsibility, Command),
- *Avhengighet av hardware og software plattform* - eksterne APIer og OS interfaces er forskjellige på forskjellige plattformer, så software som er bygget på en plattform vil være vanskelig å portere til en annen plattform (designmønstre: Abstract Factory, Bridge)
- *Algoritme-avhengigheter* - kanskje særlig et problem i vitenskapelig programvare der optimering av algoritmer er i sentrum (designmønstre: Builder, Iterator, Strategy, Template Method, Visitor)
- *Tette koblinger* - tette koblinger fører til monolittiske systemer, kanskje det viktigste anti-mønsteret å løse for en agil arkitektur (designmønstre: Abstract Factory, Bridge, Chain of Responsibility, Command, Facade, Mediator, Observer)

Et viktig og gjentagende poeng i boka er at man bør identifisere hva i implementasjonen som kommer til å variere og enkapsulere dette[8].

Selv om vi ikke har valgt å se på spesifikk bruk av design patterns i TSD, er dette en såpass grunnleggende del av IT-arkitektur og -utdanning at det kan sies å ligge til grunn for det meste av teorien bak TSD sin arkitektur. Vi har derfor valgt å ta den med her, og vi vil også se at den er relevant for mye av den andre litteraturen vi benytter i denne oppgaven.

*"**Agile IT revolution**"* (Bloomberg, 2013) er skrevet som et slags manifest der forfatteren maler en visjon om at dagens Enterprise IT raskt kommer til å bli akterutseilt i en pågående IT-revolusjon identifisert ved fem supertrender:

- *Location Independence* - rent konkret et SOA-prinsipp der den underliggende fysiske implementasjonen av en tjeneste er abstrahert vekk fra selve tjenesten. Trenden er fundamental for skyteknologi, men omfatter også mobilitet, der smarttelefoner og tilgjengelighet av nettverk gjør en tilgjengelig, uavhengig av fysisk tilstedeværelse.

[8] For eksempel om man har en liste man ser for seg at kan traverseres og aksesseres på mange forskjellige måter, så kan man introdusere en klasse av iterator-objekter som kun definerer forskjellige sett av disse mekanismene.

- *Global Cubicle* - som en forlengelse av lokasjons-uavhengighet kan man sette opp kontoret sitt hvor som helst fra[9]. Sett fra arbeidsgivers side betyr det også at ekspertise kan hentes fra hvor som helst, hvilket bidrar til in- og outsourcing, om arbeidsgiver tillater mangel på fysisk tilstedeværelse.
- *Democratization of Technology* - store enterprise-wide systemer, tunge anbudsprosesser, høy-risiko produksjonssetting er på vei ut, små systemer, fri software, SaaS og abonnementsbaserte tjenester er på vei inn.
- *Deep Interoperability* - teknologier og åpne standarder som web services og REST gir dyp interoperabilitet, men eneste måte å få leverandører til å gå for åpne standarder er å ikke kjøpe fra leverandører som ikke gjør det.
- *Complex Systems Engineering* - for å få ekte agilitet må organisasjoner tenke nytt om integrasjoner. Governance tar over for integrasjon, Chief Information Officer blir Chief Governance Officer. EA-rammeverk blir erstattet av best practices for kontinuerlig forretningstransformasjon.

Ny IT-arkitektur må være robust for kontinuerlige endringer. SOA bygget på riktig implementert Representational State Transfer (REST) og sky-teknologi kommer fullstendig til å erstatte dagens Enterprise IT når syv identifiserte brytningspunkter har manifestert seg:

1. Kollaps av Enterprise IT
2. tomt for IPv4-adresser;
3. rammeverkenes fall (Zachman, TOGAF og des like)
4. Cyberkrig
5. Generation Z (nå mer kjent som Millenials) entrer arbeidsplassen[10]
6. eksplosjon av Big Data
7. Enterprise applikasjons-kræsj.

Kanskje mest interessant for denne oppgaven er bokens del tre der et forslag på hvordan smidig arkitektur kan implementeres. Om man skjærer gjennom alle forsøkene fra leverandører på å spinne enhver ny teknologi til noe man kan tjene penger på, går tilbake til roten av SOA, REST og Cloud og unngår de vanligste misforståelsene ved REST[11] så kan man implementere ekstremt skalerbare løsninger i stand til å støtte enhver forretning i kontinuerlig endring.

"**Architectural Styles and the Design of Network-based Software Architectures**" (Fielding, 2000) oppsummerer Fieldings arbeid fra 1993 til 2000 som arkitekten bak Hypertext Transfer

[9] Dette har da også blitt enda mer relevant som følge av de pandemi-relaterte nedstengingene i 2020-2022, men spiller i dette tilfellet mer på at unge arbeidstakere mer og mer vil ha stadig mindre forståelse for at jobb er avhengig av fysisk tilstedeværelse.

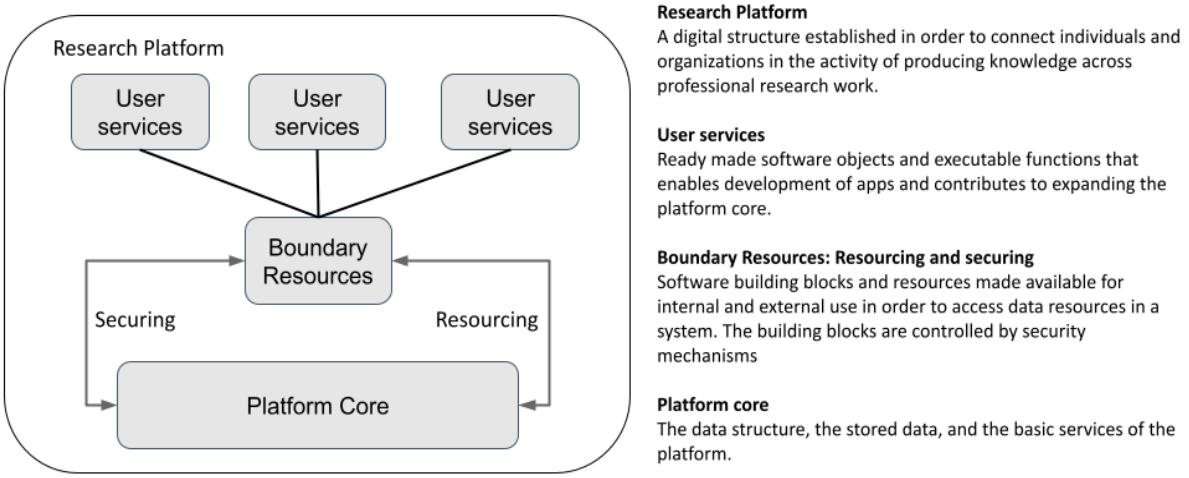
[10] Merk at dette var i 2013, millenials har kommet godt inn i arbeidslivet nå

[11] For eksempel at man ikke tar inn over seg prinsippet om "Hypermedia as the Engine of Application State" (HATEOAS), altså at man tror man kan gjøre REST *uten* å ha som mål om å bygge en distribuert hypermedia-applikasjon. Dersom hypermedia-applikasjoner *ikke* ivaretar forretningskravene, og arkitekturerfaringen din forteller deg at det å bygge en distribuert hypermedia-applikasjon ikke er veien å gå, så er *ikke* REST veien å gå.

Protocol (HTTP) og Uniform Resource Identifiers (URI), to kjernekomponenter i World Wide Web (WWW), og er som sådan en case-studie av WWW. Avhandlingen definerer REST, arkitekturen bak WWW, og viser hvordan den er bygget opp av forskjellige nettverksbaserte arkitekturer basert på 7 arkitektoniske nøkkelegenskaper: ytelse; skalerbarhet; enkelhet; modifiserbarhet; synlighet; portabilitet; og pålitelighet. Ved å se på hvilke begrensninger hver arkitektur legger kan man sette sammen flere arkitekturer og optimere på de nøkkelegenskapene som er ønsket.

En interessant observasjon her er at føringene i REST er lagt på kommunikasjonslaget, alt applikasjonslaget trenger å forholde seg til er en connector. Dette sørger i seg selv for lav kobling mellom komponenter og er godt eksempel på budskapet i (Gamma, E., & al, 1995) om å enkapsulere det som skal endres.

I "**A Research Platform for Sensitive Data**" (Øvrelid et al., 2021) brukes TSD som en use case på hvordan man kan sette opp en forsker-plattform som kan understøtte håndtering av forskningsdata (research data management) på en forsvarlig måte der prinsippene for Findable, Accessible, Interoperable, Reusable (FAIR) data samt sikkerhetspolicyer som General Data Protection Regulation (GDPR) (EU, 2016) kan ivaretas, samtidig som plattformen kan tilby de beregningsressursene og tjenestene forskerne trenger. (Øvrelid et al., 2021) definerer *forskerplattform* som en digital struktur etablert for å koble sammen individer og organisasjoner for å produsere kunnskap gjennom profesjonelt forskningsarbeid. Figur 2 er en gjengivelse av Fig. 1 i (Øvrelid et al., 2021, 130) og viser en skjematisk oversikt over hvordan en forskningsplattform bør bygges opp ifølge forfatterne. En forskerplattform består av en *plattform-kjerne* i bunnen, *kant-ressurser* som gir tilgang til underliggende *ressurser* fra plattformkjernen og sørger for *sikring* av ressursene, samt *brukertjenester* på toppen som tilrettelegger for utvikling av applikasjoner og bidrar til å ekspandere plattform-kjernen.



Figur 2 Skjematisk oversikt over oppbyggingen av en forskningsplattform

Foruten en generell beskrivelse av arkitekturen til TSD, gir (Øvrelid et al., 2021) en beskrivelse av en mulig prosess for å bygge en forskningsplattform samt hvordan man kan vedlikeholde en forskningsplattform gjennom overordnet styring av datalivssyklusen (data life cycle governance).

I "**A Multiple Case Study of Continuous Architecting in Large Agile Companies: current gaps and the CAFFEA Framework**" (Martini & Bosch, 2016) undersøker forfatterne hvordan arkitektur ledes i fem større software-selskaper og identifiserer ved hjelp av metoder hentet fra psykologi (Grounded Theory) tre arkitekt-roller; chief architect, governance architect og team architect. samt mangler i arkitektur-praksis i de studerte selskapene. Utfra dette kommer de fram til organsisasjonsrammeverket CAFFEA (Continuous Architecting Framework For Embedded software and Agile).

I tillegg til å identifisere arkitektroller kan (Martini & Bosch, 2016) sin beskrivelse av tre typer teams; Governance Teams, Architecture Teams og Runway Teams, der spesielt Runway Teams kan være med å kaste lys over noe av den agile arbeidsmetodikken i TSD.

"**Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles**" (Mårtensson, T., & al., 2019) identifiserer, ved hjelp av litteratursøk og intervjuer av personell i tre store multinasjonale selskaper, tre forbedringsområder i forhold til system-design og -arkitektur:

- Produktets arkitektur
- Måter å jobbe med system-design og -arkitektur
- Arkitektens rolle

I tillegg foreslår (Mårtensson, T., & al., 2019) tre korresponderende strategier for forbedring:

- Systemer med modulær og løst koblet arkitektur
- En balansert tilnærming der system-design og -arkitektur fokuserer på systemets kjerne-karakteristikker
- Arkitektene skifter perspektiv fra kontrollerende til fasiliterende

Et interessant funn her er at de tre selskapene, der to av selskapene i utgangspunktet ligger nær fossefallsmetodikk og ett selskap anser seg som svært agilt, ønsket seg en mer balansert tilnærming et sted mellom fossefall og 100% agilt. Med fossefall menes her at design og arkitektur defineres i en separat designfase før implementasjonen starter, mens 100% agil tilnærming er å starte kodingen umiddelbart. Det er også interessant at man også her finner empiri på at man bør tilstrebe modulær og løst koblet arkitektur, i dette tilfellet for å understøtte kontinuerlig integrasjon. Dette passer godt med (Gamma, E., & al, 1995) sine design patterns,

så vel som (Fielding, 2000) sine arkitekturbetraktninger og (Bloomberg, 2013) sine ideer om ekstremt skalerbare arkitekturer.

"**How Much Up-Front? A Grounded Theory of Agile Architecture**" (Waterman, M., & al, 2015) tar for seg spenningen mellom software arkitektur og agilitet. Med software arkitektur menes her høynivå-strukturen og organiseringen av et software-system. Dersom et agilt software team bruker for lite tid på arkitektur-design øker risiko og sannsynlighet for feiling og man risikerer å ende opp med "accidental architecture". På den annen side, dersom teamet bruker for mye tid forsinkes leveransene og verdiskapningen og evnen til å respondere på eksterne endringer forringes. Ved hjelp av intervjuer med 44 deltakere kommer (Waterman, M., & al, 2015) fram til en "grounded theory of agile architecture" med 6 *krefter* (F1-F6) som påvirker den agile arkitekturen samt 5 *strategier* for å motvirke disse:

- F1: ustabilitet i kravene
- F2: teknisk risiko
- F3: tidlig verdiskapning
- F4: team-kultur
- F5: agilitet hos kunden
- F6: erfaring

- S1: respondere på endringer
- S2: adressere risikoer
- S3: emergent (fremvoksende) arkitektur
- S4: hoved-design i forkant
- S5: Bruke rammeverk og template-arkitekturer

Merk at noen av strategiene utelukker hverandre helt eller delvis, slik at man må se an hvilke krefter som er mest fremtredende for å velge hvilke strategier som skal benyttes, ikke ulikt hvordan man kan gå fram for å velge design patterns i (Gamma, E., & al, 1995). For eksempel benyttes emergent arkitektur (S3) for å fremme kundens behov for tidlig verdiskapning (F3), men S3 utelukker hoved-design i forkant (S4). På den annen side svarer S4 ut manglende agilitet hos kunden (F5), altså at kunden har behov for å vite hele designet på forhånd (fossefallsmetodikk). Dette kan igjen tolkes som at en kunde som mangler agilitet vil kunne streve med å få oppfylt ønske om tidlig verdiskapning, og viser at for å oppnå full agilitet er man avhengig av at hele verdikjeden fra utviklere til kunder innehar en viss grad av agilitet.

En strategi som kan være av spesiell interesse for denne oppgaven er S1 "evnen til å respondere på endringer", og de kreftene som påvirker S1; "team-kultur" (F4), "agilitet hos kunden" (F5) og "arkitektens erfaring" (F6).

Med dette har vi oppsummert et utvalg av relevante artikler og relevant teori som vi vil benytte oss av i diskusjonen rundt oppgavens problemstilling. Dette er selvsagt kun et lite utvalg fra en enorm mengde mulige bøker og artikler som omhandler programvarearkitektur og programvaredesign, og er i stor grad basert på kursets anbefalte litteratur. Materialet er dog relevant, og med tanke på kursets avgrensning, anser vi utvalget som representativt nok for i tilstrekkelig grad å støtte vår problemstilling og diskusjon rundt den.

Funn og diskusjon

I denne seksjonen går vi gjennom funn fra intervjuene og samtalene vi har gjennomført i forbindelse med denne oppgaven. Vi tar også for oss noe relevant bakgrunn og historie som allerede er kjent for oss, da vi begge tilhører IT-avdelingen på UiO. Vi har valgt å inkludere diskusjonen som en fortsettelse av funnene, i stedet for en separat del, og i diskusjonen henviser vi fra litteraturen der det er relevant.

Våre hovedfunn og diskusjon rundt disse dreier seg om (i) hvilken utvikling i kodebasen til TSD har bidratt til å gjøre TSD mer robust, (ii) arbeidsmetodikken våre informanter formidlet at de benytter, (iii) de formelle og uformelle kommunikasjonsmåtene innad i teamet og hvordan dette er viktige bidrag til et agilt team og en robust kodebase. Vi tar også for oss to use-cases: (iv) implementasjonen av Educloud, eller "TSD-light" som var arbeidstittelen informantene benyttet, og (v) Open On Demand (OOD) plattformen som nylig kom på plass inn mot Fox, Educlouds HPC ressurs.

TSDs opprinnelse og utvikling

Opprettelsen av TSD sprang ut fra behovet forskere hadde til trygt å kunne oppbevare og kjøre analyser på sine sensitive data, trygt kunne dele dem med sine med-forskere, og trygt hente dem ut igjen ved behov, fra "hvor som helst i verden". IT-kompetansen til mange av disse forskerne var relativt lav, og de benyttet stort sett Windows, ikke Linux-maskiner som er mer gjengs innen tradisjonell vitenskapelig beregning. Fokuset var å levere et "Minimal Viable Product" raskt og godt, derfor ble eksisterende infrastruktur og teknologiske løsninger som allerede var i bruk på IT-avdelingen benyttet. Siden opprinnelsen av TSD har tjenesten vært i kontinuerlig utvikling, og teamet i TSD har endret seg fra å ha ren infrastruktur kompetanse til å bli mer DevOps med utviklingskompetanse. Tjenestene på toppen av infrastrukturlaget er idag helt refaktorert og er basert på tjenesteorientert arkitektur (SOA) og Application Programming Interface (API).

Ifølge den opprinnelige sjefsarkitekten, ble TSD bygget opp av 3 viktige grunnelementer: (i) identitets- og tilgangsforvaltningssystem. (ii) virtualiseringsteknologi, og (iii) virtuelt nettverk. For (i) benyttet (og benytter fortsatt) UiOs IT-avdeling Cerebrum[12]. TSD lot derfor i første omgang Cerebrum helt ta seg av identitets- og tilgangsforvaltningen, selv om nivået av sikkerhet som krevdes i TSD resulterte i behov som lå noe på siden av hva Cerebrum leverte. Cerebrum er bygget opp rundt prinsippet at en person, i dette tilfelle en bruker, kun har ett brukernavn og en unik personlig brukergruppe[13]. Denne brukeren benyttes til identifikasjon og tilgang til alle tjenester som benytter Cerebrum. I TSD er det derimot slik at en person kan ha mange brukernavn og personlige brukergrupper. Det er fordi man som medlem av et prosjekt i TSD har et unikt brukernavn gjeldende kun for det prosjektet. En bruker har derfor minst like mange brukernavn som antall prosjekter personen er deltaker i. På denne måten kan tilgang og rettigheter styres med nødvendig granularitet.

De to andre essensielle komponentene var virtualisering (ii) og virtuelle subnett (iii). For å etablere et sikkert "vanntett skott" mellom prosjektområdene, ble hvert prosjekt tildelt en virtuell maskin eller en fysisk server eid av og dedikert til prosjektet, i et privat virtuelt

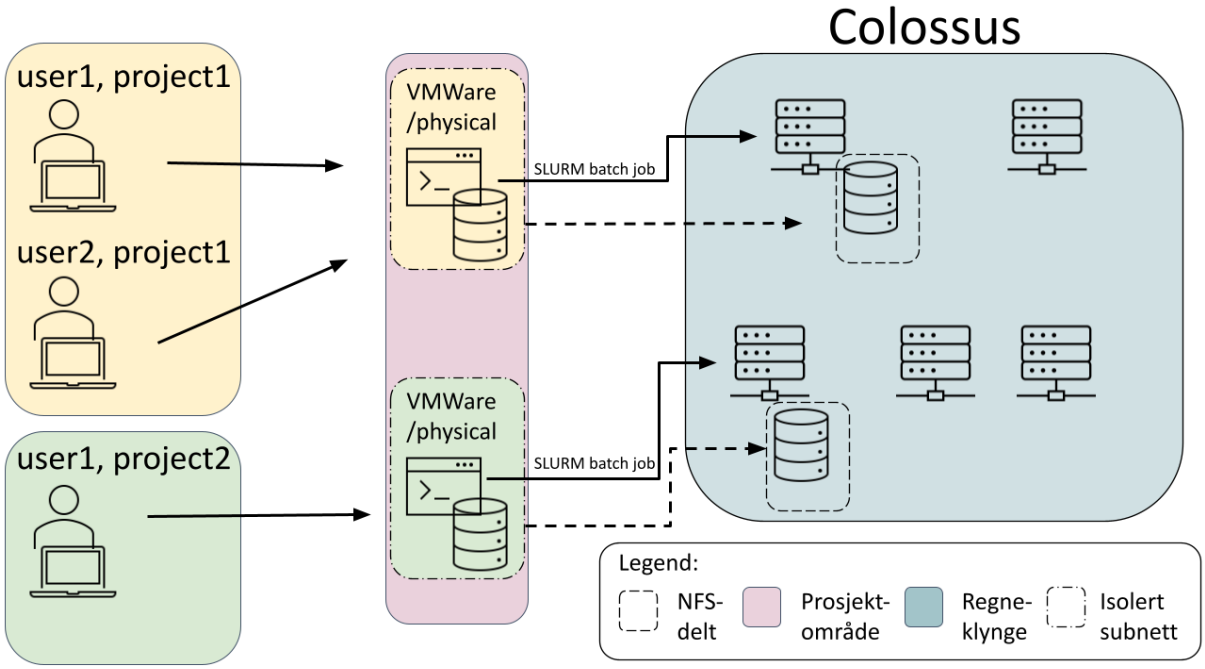
[12] https://www.usit.uio.no/om/tjenestegrupper/cerebrum/

[13] Ansatte på IT-avdelingen har i tillegg en drift-bruker. En bruker kan selvsagt også være medlem av mange upersonlige felles-grupper.

subnett. For det virtuelle nettverket benyttet man opprinnelig Cisco sin virtuelle nettverksløsning da det var/er det IT-avdelingen bruker. Innlogging til prosjektområdet foregikk (og foregår) enten fra en nettleser, der man klikket seg inn på en virtuell (Windows eller Linux) desktop, eller via en lokalt installert VMWare klient.

Etter at infrastrukturen til en første implementasjon av TSD var etablert (brukeropprettelse og tilgangskontroll, virtuelle servere og virtuelle private subnett for ytterligere sikring av prosjektområdet), var tilgang til en kraftig regneklynge den neste viktige kapabiliteten som kom på plass. Colossus regneklynge dedikert til TSD sørget for dette. Regneklyngen hadde (og har fortsatt) ingen direkte brukertilgang via login: kjøring av beregningsjobber foregår ved å submitte jobber via skeduleringssystemet Slurm[14] inn til Colossus via det sikre prosjektområdet i TSD. Tilgang til prosjektets lagrede data var (og er) via on-demand NFS-montert filsystem, sikret ved hjelp av Kerberos[15]-protokollen.

Figur 3 viser oppsett av prosjektområdet i TSD og tilknytningen til regneklyngen Colossus. Den viser hvordan brukere som er medlemmer i et prosjekt får et dedikert prosjektområde de deler. Dette er separert fra andre prosjekter. Videre vises hvordan deres data blir NFS-montert opp mot en node i regneklyngen Colossus. Monteringen skjer "on-demand" - dvs kun når en jobb (analyse av data) sendes til Colossus for kjøring. Filområdet blir demontert når jobben er ferdig. Sikkerheten er ivaretatt i Colossus ved hjelp av POSIX filrettigheter, og det er ingen mulighet for å kjøre såkalte interaktive jobber, heller ikke å logge seg inn på noen av maskinene i Colossus.



Figur 3 Skjematisk oversikt over oppsett av prosjektområde og tilgang til beregningsressurs

Med disse grunnelementene på plass kunne daværende USIT levere den første sikre tjenesten for lagring og bearbeiding av sensitiv data til forskere og klinikere i Norge, der sensitiv data

[14] https://slurm.schedmd.com/

[15] https://web.mit.edu/kerberos/

er data klassifisert[16] helt opp til sort nivå - data som er strengt fortrolig og krever den høyeste grad av beskyttelse.

TSD har fra starten av vært i en kontinuerlig utvikling for å holde tritt med behovene til det stadig økende antall prosjekter og brukere. I det følgende tar vi for oss de viktigste grepene som er gjort i forbindelse med dette arbeidet.

Grep som har blitt gjort for robusthet i TSD

Ettersom TSD vokste fra en håndfull prosjekter og brukere i 2014 til nesten 300 prosjekter og 3000 brukere i 2018, frem til dagens 2000 prosjekter og mer enn 8000 brukere ble behovene for skalerbare og robuste tjenester håndtert komponent for komponent. Fremgangsmåten kan vi finne igjen litteraturen, blant annet i (Waterman et al., 2015), der strategiene S1: respondere på endringer, S2: adressere risikoer og S3: fremvoksende arkitektur, kan sies å regjere i TSD. Fokuset har vært å få på plass løsninger relativt raskt med den kompetansen og teknologien man i øyeblikket innehar, og deretter gjøre endringer gradvis og i prioritet basert på brukerens behov. Kreftene som påvirker denne strategien er ifølge (Waterman et al., 2015) F3: tidlig verdiskapning (en viktig verdi for brukeren), F4: team-kultur (nødvendig for et agilt og løsningsorientert team) og F6: arkitektens erfaring (nødvendig med solid erfaring for å kunne lede den fremvoksende arkitekturen i riktig retning). Kreftene kan i dette tilfelle også sees på som kvaliteter, som det fra intervjuene og samtalene har kommet frem at teamet i TSD bevisst er fokusert på.

Dagens kodebase er helt refaktorert fra den opprinnelige implementasjonen. I dette arbeidet har det vært fokusert på å benytte eksisterende og veletablerte standarder der dette har vært relevant, f.eks. Kerberos-protokollen, REST interfacet (Fielding, 2000) osv. Komponentene er modularisert og er idag implementert som microservices, med alle tjenestene API-basert og kjørt i software-containere. Litteraturen, blant annet (Bloomberg, 2013) (Mikkelsen et al., 2019) (Mikkelsen et al., 2020), er i stor grad enig om at Service Oriented Architecture (SOA) eller den enda mer modulariserte mikro-service arkitekturen er en forutsetning for en robust arkitektur, og det er derfor naturlig at dette er retningen refaktoreringen har gått i. Dette har løst mange av problemene den opprinnelige implementasjonen hadde. Den første implementasjonen av TSD var preget av objekt-basert kode, der objekter deles ved hjelp av delt minne. Ifølge (Bloomberg, 2013, 151) er dette problematisk fordi: "*The challenge, therefore, with functional/RPC [Remote Procedure Call - red.anm.] programming is how to deal with tight coupling*". Problemet med slik kode er altså at den høye graden av tette koblinger gjør den lite fleksibel og dermed lite robust for endringer. I refaktoreringen av TSD er kodebasen blitt modularisert og transformert til å være enten web-basert via APIer eller koordinasjonsbasert ved hjelp av meldingskø (RabbitMQ). Koden er skrevet helt om til å bli løst koblet, hvilket ikke bare forbedrer kodebasens robusthet i seg selv, men også utviklingsteamets mulighet for å være agile. Med løse koblinger kan enkelt-utviklere jobbe med komponenter uten å måtte ta hensyn til objekt-deling i andre deler av koden de kanskje ikke har detaljkunnskap om. Det er et veldefinert kommunikasjonslag mellom komponentene i form av APIer, og i tillegg benytter alle komponentene meldingskøen som sørger for en koherent og sikker kommunikasjon mellom komponentene.

En veldig viktig komponent som det tidlig ble klart at det måtte tas kontroll over var Cerebrum som ble benyttet for identitets- og tilgangsstyring (Identity and Access

[16] grønne data: åpent eller fritt tilgjengelig, gule data: begrenset, røde data: fortrolig, sorte data: strengt fortrolig

Management - IAM). Behovene i TSD var som nevnt noe på siden av det Cerebrum ble utviklet for. Men den viktigste flaskehalsen var ifølge informantene allikevel at utviklingsteamene opererte i siloer og at utviklingsprosessen av et monolittisk Cerebrum ikke var fleksibel og agil nok til å imøtekomme behovene i TSD. Utviklingsteamet i TSD tok derfor gradvis kontroll over utviklingen ved først å kopiere funksjonaliteten i Cerebrum, for deretter å gradvis utvide den etter behov. Med denne fremgangsmåten sikret de seg at allerede eksisterende funksjonalitet var ivaretatt, men at de samtidig trygt kunne utvikle koden videre for ny funksjonalitet.

Innføring av APIene ble foretatt komponent for komponent i tråd med en agil arbeidsmetodikk. Første komponent ut var fil-APIen. Sikker overføring og lagring av data (filer) har alltid vært en av hoved-tjenestene til TSD. I den opprinnelige versjonen fungerte selvsagt løsningen, men det var en tungvint affære. Brukeren måtte først selv kryptere sin egen data. For å unngå å sikkerhetsrisikoen med åpne porter inn til prosjektområdene i TSD ble dataene først lastet opp til en dedikert felles fil-server via sftp, den såkalte "filslusa". Deretter ble dataene automatisk kopiert inn til brukerens prosjektområde i TSD. Kopieringen kunne ta lang tid, og det skapte mye forvirring hos brukerne at dataene ikke var på plass i prosjektområdet med en gang. Denne løsningen hadde flere potensielle risikofaktorer, f.eks. at brukeren ikke tok seg tid eller ikke hadde kompetanse til å kryptere dataene, krypteringen var for svak, filer kunne bli gjort synlige for brukere utenfor prosjektet, brukeren kunne gjøre feil slik at data gikk tapt eller ble korrupt, osv. Sett fra et kost-nytte prinsipp og fokus på tidlig verdiskapning (Waterman et al., 2015) var det klart at en fil-API var en god første tjeneste å levere etter "ny lest" for omtrent 6 år siden. Import ble gjort klart først, deretter eksport omtrent et år senere. Med Fil-APIen er det ikke lenger noen potensiell risiko for å dele data, og kommunikasjonen mellom klient og server er "end-to-end" kryptert. Dataene blir streamet direkte inn til endelig TSD prosjekt-område. Dette gjør tjenesten både veldig mye sikrere og samtidig veldig mye mer brukervennlig. En heldig bivirkning var ifølge våre informanter at antall support-tickets gikk drastisk ned, løsningen fungerte nå slik brukerne forventet.

Neste tjeneste som ble levert var den såkalte data-loaderen. Dette var en ny tjeneste, og slik sett ikke en refaktorering av en eldre løsning, men implementasjonen ble utført i tråd med den modulariserte kodebasen til TSD. Data-loaderen er en funksjonalitet i TSD som aggregerer filer som er lastet opp og lagret i prosjektområdet til en kombinert fil - den "merger" data. Data er i dette tilfellet svarskjema fra Nettskjema. For enklere analyse er det essensielt å merge data fra potensielt tusen små filer til en fil som inneholder alle svar. Tidligere måtte brukerne gjøre dette manuelt etter eget initiativ. Det innebar først å eventuelt måtte dekryptere filene fra Nettskjema i prosjekt-området, merge de, og eventuelt kryptere de på nytt. Med data-loaderen blir dette gjort automatisk. Data-loaderen sjekker jevnlig om nye filer (svar-skjema) har kommet inn, og setter igang en merge-jobb om det er tilfellet. Alt skjer i bakgrunnen og automatisk, uten behov for manuelle steg fra brukerens side. Dette reduserer behovet for teknisk ekspertise hos brukeren, som igjen reduserer sjansen for feil i prosessen og som igjen fører til økt sikkerhet og et robust system.

En neste stor og viktig utvikling for nye TSD var implementasjonen av selvbetjeningsportalen. Tidligere måtte alt av brukerforespørsler, det være seg søknad om nytt prosjekt, registrering av et nytt prosjekt-medlem, behov for nytt passord eller lignende, behandles via e-post, med manuell behandling av alle saker. Dette nevner alle informantene fra TSD at fungerte fint i starten av tjenesten, da det var relativt få brukere og prosjekter. I dagens situasjon med over 8000 brukere er det klart at denne løsningen ikke er skalerbar.

Automatisering måtte på plass, og det kom i form av en web-basert selvbetjeningsportal. Oppgaver relatert til brukeradministrasjon, prosjektsøknad, prosjektadministrasjon, og tilgang til interaktiv innlogging til prosjektområdet i TSD er nå tilgjengelig via selvbetjeningsportalen.

Arbeidet med moduliseringen av kodebasen i TSD diskutert over, ble satt igang for å imøtekomme skalering av tjenesten. Både i form av at systemet effektivt og raskt måtte kunne håndtere et stadig økende volum av innkomne kall, men også i forhold til å organisere og modularisere kodebasen for å minimere risiko for feil, og øke effektivitet i vedlikehold og feilretting - dette diskuterer vi i mer detalj i neste seksjon. Når det gjelder IT-arkitekturen og dets struktur basert på krav for sikkerhet og granularitet har dette påvirket valg av teknologi i infrastrukturlaget. Cisco ble som nevnt valgt i første instans fordi dette var teknologien de nettverksansvarlige hadde erfaring med og kompetanse i. Behovet for det veldig høye antall private virtuelle subnett (ett per prosjekt) kunne til slutt ikke støttes av Cisco, og man så seg nødt til å bytte teknologi. Valget falt på VMWare sin løsning NSX-T[17]. Denne endringen var en mye tyngre og smertefull prosess enn endringen i kodebasen, som sier noe om at infrastrukturen i bunn er tyngre å endre på enn software-laget på toppen. I etterpåklokskap kunne man hevde at det ville vært riktigere å gå for NSX-T fra starten av. Det ville dog hindret den smidige og raske opprinnelige realisasjonen TSD, kanskje til og med ført til at tjenesten ikke ble realisert i det hele tatt. Så selv om bytte av teknologi var smertefullt kan vi i TSD sitt tilfelle hevde at det var en smertefull endring som måtte komme på et tidspunkt. Og kost-nytte verdien av å først velge Cisco var helt klart positiv, fordi TSD faktisk kom på plass og ble benyttet i stadig økende grad.

Vi har hittil beskrevet tekniske grep som er gjennomført for å gjøre TSD mer robust, men kanskje vel så viktig for robusthet i et større software-system er kulturen og arbeidsmetodikken i utviklerteamet. Dette diskuterer vi i neste seksjon.

Utviklingsteamets roller og arbeidsmetodikk

Rundt 2017 ble det foretatt store endringer i organiseringen innad i teamet bak TSD. I dag er det 14 personer som utvikler og vedlikeholder kodebasen, og teamet gikk fra i hovedsak å være de opprinnelige utviklerne med bakgrunn fra infrastruktur, til folk med utviklerbakgrunn og DevOps erfaring. DevOps (Humble & Molesky, 2011) (Bloomberg, 2013) “development and operations“- er en arbeidsmåte med tilhørende teknologi som sikter på å forene utviklingsprosessen (development) med implementasjonsprosessen (deployment) som viktige integrerte prosesser i en vellykket utviklingssyklus, og må sies å ha blitt en arbeidsmetode de fleste større utviklingsteam benytter idag. I DevOps er ideen kontinuerlig kode-utvikling, kontinuerlig leveranse, kontinuerlig implementasjon og ideelt sett kode basert på microservices. Alt dette sørger for en agil kodeutviklings-livssyklus. Med DevOps oppsto mye teknologi for å understøtte den kontinuerlige utviklings- og implementasjonsprosessen. I dagens realisasjon er dette for eksempel GitHub og GitLab med sine web-verktøy for kode-revisjon gjennom “pull” eller “merge-requests” der det er en integrert mulighet for kode-gjennomgang av andre før endringen legges til i kodebasen. Man kan sette opp koden til automatisk å bygge på det antall plattformer man måtte ønske for å teste at koden ikke har introdusert feil, og sette opp andre automatiske tester i tillegg til automatisk deployment - installasjon i produksjon - om man måtte ønske det. DevOps prosessen og verktøyene som

[17] https://docs.vmware.com/en/VMware-NSX

følger med har, mener både vi som intervjuere og våre informanter, vært avgjørende for at TSD har kunnet levere robust kode av høy kvalitet.

Før vi diskuterer hvordan arbeidet rundt software-arkitektur foregår i TSD er det på sin plass å definere rollene nøkkelpersonene i TSD innehar. Vi kan her benytte arkitekt-rollene beskrevet i (Martini & Bosch, 2016, 2). Her kan en person i TSD teamet kalles "sjefsarkitekt", mens det er totalt 3 personer som har såpass stor oversikt over hele arkitekturen at de kan kalles "governance architects". De 3 fungerer som mediatorer videre inn mot utviklingsteamet i forhold til de større arkitektoniske utviklingslinjene og implikasjonene dette har for de forskjellige komponentene. I og med at de selv er del av utviklingsteamet kan de også sees på som en slags "team-arkitekter", selv om hvert "feature-team" slik (Martini & Bosch, 2016) definerer de, er større enn teamene innad i TSD. Team-arkitektene er ifølge (Martini & Bosch, 2016,5) utviklere som er ansvarlige for arkitekturen i "feature-teamet" og fungerer som en teknisk leder eller erfaren utvikler. De 3 vi kaller governance arkitekter har en slik team-arkitekt-rolle og i TSD sitt tilfelle smelter derfor disse rollene sammen.
Endringer i arkitekturen diskuteres i utviklerteamet før de blir implementert. På denne måten får teamet god generell oversikt over strukturelle implikasjoner og de kan samtidig bidra med diskusjoner om spesielle hensyn eller behov i tilknytning til komponentene den enkelte utvikler arbeider med. Kun etter diskusjon i teamet starter koding, og da er arkitektoniske implikasjoner allerede diskutert og forstått. Dette gir mindre sjanse for at løsninger som implementeres senere vil gi arkitektonisk gjeld, og er viktig for systemets robusthet. Ifølge sjefsarkitekten i TSD selv, kan arkitektonisk gjeld defineres på følgende måte (oversatt fra engelsk): *"Hvis du har et sluttbruker-problem som ikke passer inn i dagens arkitektur, da har du ATD (Architectural Technical Debt)."*

Sjefsarkitekten og de 3 "governance arkitektene" er i regelmessig kontakt og har både formelle og uformelle diskusjoner når det gjelder implikasjoner på arkitekturen i TSD. Dette bidrar til viktig deling av ideer og forståelse for problemene og mulige løsninger, i tråd med undersøkelsene foretatt av (Mårtensson et al., n.d., 134): "*Another perspective is architecture communication: [...] it is important to pay attention to the fact that the architects and the development teams communicate iteratively (in both directions). In particular, architects need to communicate clearly and continuously the importance of the main architectural significant requirements, while they need to have feedback (from the team or from analysis tools) on the status of the system, to understand if the architectural requirements are at risk.*"
I TSD fungerer denne iterative kommunikasjonen både som et resultat av DevOps prosessen, men også de uformelle møtepunktene og diskusjonsarenaene.

Teamet er såpass lite at de fleste til en viss grad kan være involvert i det meste av prosessene og det er enkelt med både formelle og uformelle diskusjoner rundt valg av løsninger. Blant annet møtes de i teamet som ønsker og har anledning for en daglig sosial morgenkaffe. Utviklerne vi intervjuet i TSD poengterte også at det er stor takhøyde for å innrømme og diskutere feil i teamet. Dette er en veldig viktig ressurs som sørger for at feil får minimal negativ effekt siden feilene raskt blir oppdaget og løsninger blir diskutert i teamet uten at det får noen negativ personlig konsekvens for vedkommende som introduserte feilen. Som (Waterman et al., 2015, 351) poengterer: "*A culture that is people-focused and collaborative is a very important factor in a team's ability to communicate*". I TSD er det stort og bevisst fokus på nettopp dette. Utviklingsteamet i TSD har fortsatt fordelen med å være relativt lite. I større team er en mer stringent organisasjon viktigere, og det er viktig at rollene er mer formalisert. Allikevel ser vi de viktige arkitekt-nøkkelrollene er ivaretatt, noe som er

essensielt for at utviklerne er inneforstått med de grunnleggende arkitektoniske hensyn og planer og strategier fremover. Dette sørger for å redusere risikoen for å velge løsninger som i det senere løp fører til arkitektonisk teknisk gjeld.

I arbeidet med refaktoreringen, kan måten teamene har arbeidet på til en viss grad ha likheter med "Runway Teams" beskrevet i (Martini & Bosch, 2016, 6). Et Runway Team arbeider med å legge til rette for endringer i arkitekturen som er initiert etter behov for refaktorering heller enn direkte brukerbehov (som utvikles av Feature Teams). Arbeid med arkitekturen krever en annen måte å arbeide på enn i et typisk agilt Feature Team. "*Unlike the functionality of a system, design cannot easily be decomposed into small chunks of work, user stories, or "technical stories." Most of the difficult aspects of architectural design are driven by nonfunctional requirements, or quality attributes: security, high availability, fault tolerance, interoperability, scalability, and so on.*"(Bellomo et al., 2014, 10). Refaktoreringsarbeidet har nettopp vært et slikt type arbeid. De fleste utviklerne har vært del av dette arbeidet, og teamene har måtte operere "mer globalt" sammenlignet med vanlig utvikling som ikke påvirker arkitekturen. Dette vil vi påstå har kommet hele TSD prosjektet og utviklerne til gode i form av en verdifull oversikt over den helhetlige arkitekturen og "filosofien" i TSD som igjen kan sies å bidra til robusthet.

Ifølge informantene foregår utviklingsprosessen i TSD stort sett etter følgende lest: når endringer eller nye moduler skal implementeres er det først en planleggings-sesjon med brainstorming. Her er sjefsarkitekten involvert ettersom han har den helhetlige oversikten og også har best bilde av brukerbehovene, dersom det er relevant. Dersom endringen er initiert av et spesifikt brukerbehov, generaliseres dette slik at løsningen tilfredsstiller det aktuelle behovet, i tillegg til å være fleksibel nok til å romme kommende behov. Teamet diskuterer hva som er mulig å implementere, og bestemmer deretter hvem i teamet som skal utføre oppgaven. Kodingen foregår vanligvis uten moderasjon først, men med samarbeid med andre i teamet der det er behov, f.eks. i forhold til komponenter som trenger tilknytning til bruker-autorisasjon. Etter at første kodeforslag er på plass tas en kode-gjennomgang i teamet, modul for modul, der utviklings- og implementasjonsvalg blir forklart. Kode-gjennomgangen foregår stort sett ukentlig. I denne kodegjennomgangen kan andre komme med forslag og bemerkninger som bidrar til å forbedre løsningen. Dersom mange kommentarer kommer inn, kan prosessen gjentas etter at de nye forslagene er implementert - i en iterativ prosess.

Jevnlig går teamet også gjennom såkalte "epics". Dette er større, overordnede saker, og gir utviklerne innblikk i hva de andre utviklerne jobber med for tiden. Dette sier en av informantene gir en fin mulighet til å få overblikk og få andre perspektiver i form av løsninger og teknologi, på samme måte som den jevnlige kode-gjennomgangen gir, men på et større og mer sammensatt plan.

En annen viktig strategi har vært at ny funksjonalitet og refaktorering har blitt utviklet parallelt med den eksisterende løsningen. I en overgangsperiode opererer den gamle og den nye løsningen side om side, og først når den nye løsningen er blitt tilstrekkelig stresstestet og det er blitt vist at den fungerer som tiltenkt, har den gamle blitt faset ut. Dette har også gitt brukerne god tid til å bli kjent med den nye løsningen, noe som hjelper til å redusere trykket fra support-henvendelser. Alle utviklerne er involvert i rullerende support-skift, og det er naturligvis viktig at endringer som påvirker brukerne skaper minst mulig forvirring og feil av opplagte årsaker, men også for å frigjøre tid til utviklingsarbeid.

I tillegg til planleggingsmøtene for større endringer, og de ukentlige kode-gjennomgangene, arrangeres det halvårlige retrospekt-sesjoner. Disse tar for seg hva teamet har gjort i den forrige perioden, hva som har gått bra, hva som kan forbedres og hva man bør ta med seg inn i neste periode. Ifølge sjefsarkitekten settes disse sesjonene opp slik at de er tett etterfulgt av en sesjon der neste periode planlegges, der man tar inn input fra jevnlige brukermøter. Disse periodiske sesjonene er med og sørger for at den overordnede styringen av datalivssyklusen (Øvrelid et al., 2021, 133) ivaretas slik at forskerplattformen stadig holdes vedlike.

Vi har nå sett på hvilke endringer som er gjort i TSD sin kodebase og også hvilke endringer som er gjort i forhold til arbeidsmetodikken i utvikler-teamet for å øke graden av software systemets robusthet. I de neste tre seksjonene tar vi for oss hvor godt TSD har stått seg i forbindelse med implementasjon i en annen infrastruktur og om TSD har kunnet støtte integrasjonen av en ny plattform inn mot Educloud uten å rokke ved systemets integritet.

Fra TSD til Educloud

Educloud dukket først opp som TSD light og deretter Forskerplattformen, blant annet i UiOs Masterplan for IT (Ludvigsen et al., 2018). Man hadde da som nevnt sett en sterk vekst i bruken av TSD og innså at en del av prosjektene som kom til TSD ikke nødvendigvis kom dit på grunn av behov for å håndtere svarte data (data på høyeste sensitivitetsnivå), men mer på grunn av tjenestene TSD tilbød, med gode samarbeidsmuligheter, selvbetjening på prosjektnivå, mulighet til å få inn eksterne samarbeidspartnere nasjonalt og internasjonalt, samt deling av ressurser og data.

Samtidig så man i UH-sektoren en vesentlig utfordring rundt samarbeid på datasett med personer som ikke jobber på moderinstitusjonen (Thomassen, 2019). Man så dette på to måter, enten at datamengdene ble så store at flytting og kopiering ble uhensiktsmessig og/eller ulovlig, eller så brøt man General Data Protection Regulation (GDPR) ved å ha sensitive data liggende på private eller delte lagringsenheter slik at moderinstitusjonen var uten mulighet til å følge opp sine lovpålagte krav som behandlingsansvarlig organisasjon. Alle disse problemene var allerede løst i TSD, men på grunn av de svært høye kravene til sikkerhet ble sluttbrukerne påført uhensiktsmessig og unødvendig sterke restriksjoner.

Man begynte derfor på arbeidet med en ny plattform, heretter kalt Educloud. Føringene var at Educloud skulle være så lik som mulig som TSD, men med forenklinger der redusert sikkerhetsnivå gjorde dette mulig[18]. Ifølge den originale sjefsarkitekten og begge utviklerne i TSD som vi intervjuet skulle det teknisk sett være mulig å gjenbruke mye av kodebasen fra TSD inn i Educloud. Det viste seg også å være tilfellet, arbeidet med å modularisere TSD kunne i stor grad sies å ha gjort TSD robust, om man måler det i forhold til hvordan implementasjonen i Educloud gikk. Vi siterer en av informantene : "*As platform - it was not designed as anything different than TSD - everything was just swiftly translated to Educloud, the authorisation and user-management module was already completely compatible. Some changes [were needed - red.anm.] on the user-access. These differences are now reflected in the development process: earlier development was only for TSD, now the development takes into account the differences between Educloud and TSD and one codes with the two in mind.*"

[18] For eksempel er prosjektområder permanent montert på Educlouds HPC-anlegg Fox, i motsetning til TSDs Colossus hvor de monteres "on-demand" slik at kompleksiteten blir redusert mot noe redusert sikkerhetsnivå, men ikke mer enn at Educloud er sikkert nok for røde data.

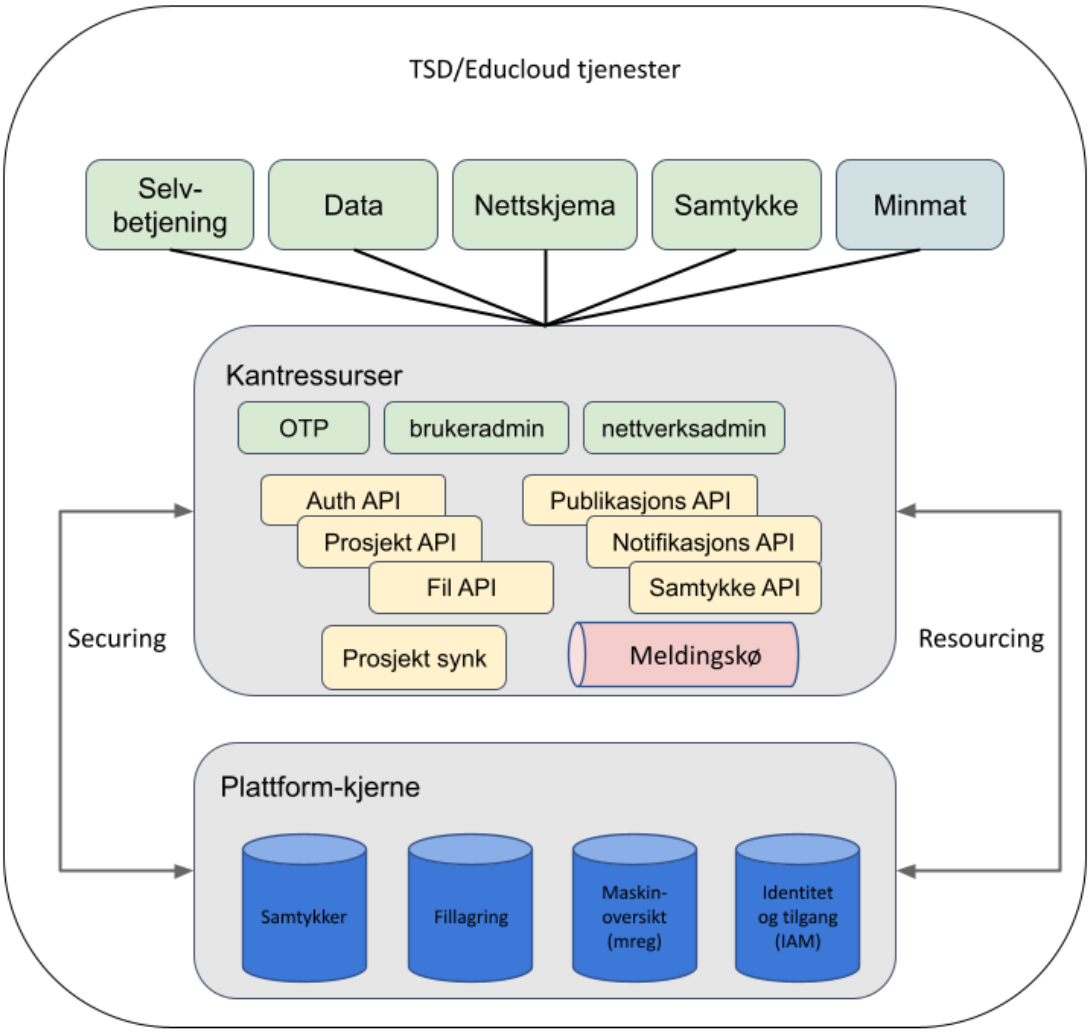
På ett punkt oppstod det allikevel problemer. Av flere årsaker, som mangel på ledige ressurser i teamet bak den raskt voksende TSD og ressurser med ledig kapasitet andre steder i organisasjonen, ble et nytt team med ny leder utenfor TSD satt opp for å utvikle Educloud. Dette, sammen med den type løs kultur med mye kompetanse og rom for egen-initiativ som IT-avdelingen har, kan ha gitt opphav til kommunikasjonsproblemer og at man snakket forbi hverandre i organisasjonen. Det ble i alle fall klart at man beholdt komponenter man egentlig ikke trengte å beholde og at man lagde ting på nytt som man kunne ha beholdt.

Et konkret eksempel på sistnevnte var selvbetjeningsløsningen i Educloud. Ifølge begge utviklerne i TSD vi snakket med var selvbetjeningsløsningen noe man kunne overføre nærmest as-is til Educloud. I stedet ble det igangsatt et sommerprosjekt der en ekstern utvikler uten tilknytning til TSD utviklet en ny selvbetjeningsløsning. Denne ble satt i produksjon og fungerte etter formålet, men ble nok en komponent å utvikle og vedlikeholde, og dermed et bidrag til teknisk gjeld. Det jobbes nå med å harmonisere denne løsningen slik at det kun blir en kodebase for både TSD og Educloud. En interessant effekt av denne inkurien er at den nye selvbetjeningsløsningen tvang TSD-utviklerne til å se på selvbetjeningsløsningen med nye øyne, og har ført til at man har utviklet ny og bedre mellomvarearkitektur for selvbetjening felles for både TSD og Educloud. Slikt sett kan man se på den løse kulturen i IT-avdelingen som en kalkulert risiko som kan gi endel rot, men som også gir en hvis grad av innovasjon.

Før vi tar for oss et eksempel på ny funksjonalitet inn mot Educloud og hvordan systemets integritet har stått seg, ønsker vi å gå litt mer i detalj rundt oppbyggingen av TSD og Educloud som forskerplattformer.

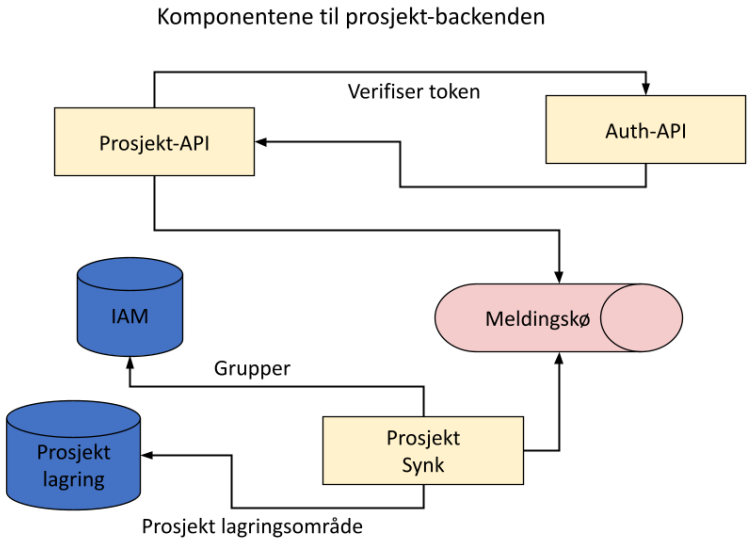
TSD og Educloud som forskerplattformer - tjenester og APIer

Som beskrevet i (Øvrelid et al., 2021) består en forskerplattform av en plattform-kjerne, kant-ressurser og en rekke brukertjenester. Forskerplattformer skiller seg fra kommersielle plattformer i at man i en forskerplattform ikke har det klassiske ekosystemet der man har en sentral database for store transaksjonsvolumer som man får forskjellige stakeholders til å benytte for å kjøpe og selge tjenester (f.eks. slik som Uber og AirBnB). En generell forskerplattform er beskrevet i Figur 2, og beskriver godt arkitekturen til både TSD og Educloud, men det kan være interessant å se litt mer i detalj hvilke komponenter som er i spill og hvordan de plasseres i arkitekturen.

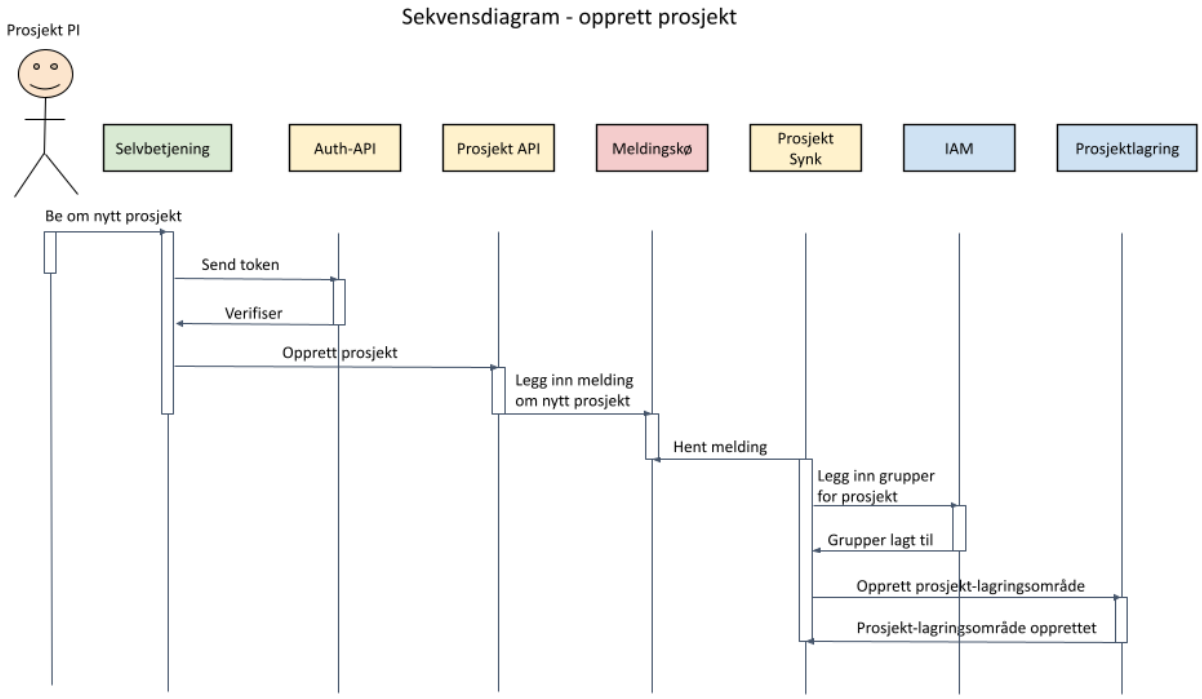


Figur 4 Skjematisk oversikt over oppbyggingen av tjenestene i forskningsplattformene TSD/Educloud

Figur 4, basert på innspill fra arkitektene og utviklerne i TSD, viser på toppen tjenester som er tilgjengelige for brukere utenfor TSD; selvbetjeningsløsningen, dataportalen, Nettskjema og Samtykke-portalen er eksempler på generelle tjenester utviklet av TSD-teamet, mens Minmat er et eksempel på en tjeneste spesifikt utviklet for et prosjekt i TSD. Disse tjenestene benytter seg av kantressurser som OTP, brukeradmin og nettverksadmin, og APIer for autentisering, prosjekter, filer, publikasjon, notifikasjon og samtykke, og tjenestene og APIene har tilgang til meldingskø for asynkron kommunikasjon der det er naturlig. Formålet til kantressursene er å gi tilgang til ressursene i plattform-kjernen, for eksempel database med samtykker, fillagring, maskinoversikt og identitet og tilgang (IAM).



Figur 5 Skjematisk oversikt over komponentene til prosjekt backenden



Figur 6 Sekvensdiagram for å opprette prosjekt i forskningsplattformen

Vi kan også se at arkitekturen er basert på mikrotjenester, der hver komponent har høy kohesjon med kun en eller maks to spesialiserte oppgaver. Som et eksempel viser vi i komponent-diagrammet i Figur 5 og sekvens-diagrammet i Figur 6 hvilke komponenter som

er involvert eksemplet: En prosjekt-administrator ønsker å opprette et prosjekt. Vi ser at det er to API-er involvert, Auth-API og Prosjekt-API - brukeren vekselvirker via selvbetjeningsportalen med Prosjekt-APIen, mens Auth-APIen er en intern komponent som de andre komponentene vekselvirker med. Prosjekt-APIen legger så en melding i meldingskøen om å opprette nytt prosjekt. Dette plukkes opp asynkront av prosjekt-synkroniseringstjenesten, som så oppdaterer IAM med aktuelle grupper og deretter setter opp prosjektlagring.

I tråd med mikrotjeneste-konseptet har vi nå sett at arkitekturen er lik for TSD og Educloud, mens de underliggende komponentene og infrastrukturen kan være forskjellig (Mikkelsen et al., 2019, 7059). Denne arkitekturen muliggjør løse koblinger. Et konkret eksempel på dette er IAM som tidligere nevnt i utgangspunktet var Cerebrum. Da Cerebrum ble byttet ut var det ikke merkbart for brukerne ettersom autentiserings-APIet forble uendret.

Legge til ny funksjonalitet i Educloud for å dekke konkrete forskerbehov

Som beskrevet i Problemstilling og avgrensninger ønsker vi å se på hvordan vi kan legge til ny funksjonalitet uten å endre den underliggende arkitekturen. Litt av bakgrunnen for dette er at IT-avdelingen og FFU jevnlig mottar nye forespørsler og behov fra forskere som utforsker nye problemstillinger eller skalerer opp gamle problemstillinger utover egne kontormaskiners rammer. Som et konkret eksempel mottok vi nylig en henvendelse fra en forsker som trenger å segmentere MR og CT bilder. Bildene er relativt store slik at "Jo sterkere maskin, jo raskere kan vi jobbe." (forskerens sitat). Forskeren veileder et 20-talls studenter og de har alle behov for å kjøre slike segmenterings-jobber. Verken forskeren eller noen av studentene hadde tidligere erfaring i bruk av HPC-anlegg, og erfaringsmessig vet vi at det er en relativt bratt læringskurve når det kommer til HPC. Samtidig var det klart at HPC-ressurser var en naturlig vei å gå, og den spesifikke applikasjonen som krevde mesteparten av ressursene var allerede tilgjengelig på RedHat Enterprise Linux (RHEL), som er det operativsystemet HPC-anleggene våre benytter.

Parallelt med at at dette behovet ble innmeldt hadde HPC-arkitekten sett etter nye løsninger for å gjøre Fox mer tilgjengelig for nye brukere. Standard bruk av HPC-anlegg består av å logge seg inn ved hjelp av SSH, generere et jobb-skript[19] som submittes i en tekst-basert terminal. Denne måten å bruke et HPC-anlegg har vært mer eller mindre uforandret de siste fire tiår, og kan vanskelig kalles brukervennlig. Siden Fox står plassert i Educloud, med tilhørende muligheter til å spinne opp nye tjenester ønsket HPC-arkitekten å se på mulighetene for å sette opp en mer brukervennlig tjeneste for å utnytte HPC-ressurser. Han kom da over Open OnDemand (OOD) (Hudak et al., 2018, 622), en web-basert klient-portal for HPC-anlegg. HPC-arkitekten forteller at han måtte starte mer eller mindre fra scratch "*siden webteknologi og slik ikke er noe jeg har jobbet stort med de siste 20 åra*". Etter sterk oppmuntring fra sjefsarkitekten og med bestilling om å sette opp en MVP for å undersøke potensialet av OOD gikk han videre til å sette opp portalen og fant fort ut at dette så ut til å passe godt inn sammen med resten av infrastrukturen rundt Fox og i løpet av siste halvdel av desember 2022 fikk han satt opp en MVP med de mest etterspurte web-baserte grafiske verktøyene - RStudio og Jupyter Notebook[20] koblet mot køsystemet i Fox.

[19] Et BASH-liknende skript med instruksjoner til et køsystem, i Fox og Colossus sitt tilfelle Slurm.

[20] Dette er verktøy vi tidligere har måttet si nei til på HPC-anlegg ettersom vi manglet passende infrastruktur for å tilby dem på en sikker måte .

De største utfordringene han møtte i det å sette opp OOD var som følger:

1. Sette opp VM i riktig nett, med tilgang til Fox sitt interne nett
2. Integrering av autentiseringen med Educloud sin autentisering
3. Sette opp selve tjenesten med Educloud sin lastbalansering og policier
4. Tilpasse OOD sine integrasjoner av verktøy mot Fox sitt oppsett av applikasjoner, moduler og køsystem
5. Installere nye software-moduler på Fox

For (1) og (2) bidro en kollega ved Seksjon for Operativsystemer, for (3) bidro en kollega ved Seksjon for Applikasjonsplattformer og for (5) bidro en kollega fra Seksjon for Beregningstjenester, ref Figur 1. De som bidro, ble kontaktet uformelt og prioriterte på eget initiativ å sette av tid til dette. I punktlisten over dreier (1) og (3-5) seg om tekniske forhold rundt infrastrukturen, mens (2) dreier seg om implementasjon inn mot Educlouds autentiseringstjeneste. Implementasjonen tok plass uten noen behov for endringer i Educlouds software arkitektur, som forventet fra et robust software system basert på mikrotjenester.

Selv om OOD allerede nå har all funksjonalitet på plass, påpeker HPC-arkitekten at tjenesten fortsatt har et stykke å gå før den kan tilbys som en operativ tjeneste. I tillegg til brukertesting må den inn i en ordentlig workflow med all konfigurasjon i Git, og oppsett med Ansible og OpenShift. Ifølge den originale sjefsarkitekten er det også meningen at denne tjenesten skal leveres i TSD, men dette vil kreve noen endringer på nettverksoppsettet og vil derfor kreve noen runder innom Endringsrådet[21] først.

Når vi så går tilbake til forskeren med behov for tilgang til ressurser, så kom forespørselen HPC-arkitekten i hende tidlig i januar 2023. Applikasjonen som krevdes var da ikke installert verken i OOD eller i Fox. 33 minutter senere hadde han løsningen på plass i OOD, og to dager senere var applikasjonen installert globalt på Fox gjennom modul-systemet og det var klart for brukertesting. Den nevnte forskeren er nå første testbruker av OOD.

Det skal sies at det var endel tilfeldigheter som gjorde at brikker falt så raskt på plass i dette eksempelet, og man må normalt forvente mer enn 33 minutter fra ny funksjonalitet forespørres til den er på plass. På den annen side er det et fint eksempel på to ting:

1. Å ha et HPC-anlegg plassert i en forskerplattform er ikke bare en måte å legge til mer regneressurser i plattformen. Det er også en kraftfull anledning til å forbedre brukeropplevelsen av HPC-anlegget slik at man kan gi enda flere forskere mulighet til å gjøre mer forskning mer effektivt.
2. IT-avdelingen har som nevnt en kultur med forholdsvis løs struktur med mye kompetanse og høy grad av egen-initiativ. Vi har allerede vært inne på risikoen for rot og unødvendig ekstraarbeid som vi så i overføringen fra TSD til Educloud, men som vi ser av oppsettet av OOD kan det også lede til nye og såpass banebrytende tjenester som det er tvilsomt om et prosjekt med dedikerte ressurser, nøyaktige spesifikasjoner og forankring fra høyeste hold ville hatt sjans til å komme i mål med.

Her ønsker vi igjen å trekke frem (Waterman et al., 2015) i forbindelse med punkt 2 over, nemlig strategi S1 - "evnen til å respondere på endringer" og de kreftene som påvirker S1. I

[21] Det formelle møtet med hovedarkitektene, sikkerhetssjefen på UiO og andre relevante stakeholders der endringer som kan påvirke arkitekturen, sikkerhetsnivå og/eller brukere blir tatt opp til godkjenning.

dette tilfellet kan man i alle fall slå fast at team-kultur (F4) spilte i riktig retning der alle involverte raskt forsto målet og bidro med kompetanse. Man skal nok heller ikke underslå arkitektens erfaring (F6) i dette tilfellet, med 20 års fartstid innen infrastruktur og HPC, som viktig bidrag til en så rask implementasjon.

Konklusjon

I denne oppgaven har vi sett på hvordan UiOs IT-avdeling jobber for å levere robuste IT-løsninger som dekker forskeres behov og hvordan arkitekturen til forskerplattformene TSD og Educloud er fleksibel nok til å få inn ny funksjonalitet uten å endres i større grad. Gjennom semistrukturerte intervjuer og uformelle samtaler med hoved-arkitektene og -utviklerne av TSD og Educloud har vi gjort fire hovedfunn:

1. Utviklingen og refaktoreringen av kodebasen i TSD fra ca 2017 da det nåværende teamet fikk hovedansvaret og fram til nå har bidratt til å gjøre TSD mer robust. Dette har de oppnådd ved å sette opp en agil arkitektur med fokus på løse koblinger, og APIer og mikrotjenester med høy kohesjon og klare oppgaver.
2. Arbeidsmetodikken med DevOps-prosessen og verktøyene som følger med har vært og er avgjørende for at TSD kan levere robust kode av høy kvalitet
3. Kommunikasjonsmåtene som benyttes i TSD og Educloud teamet, sosialt med uformelle kaffepauser og kultur for å si fra om og innrømme incidenter og fokusere på hva man kan lære av dem, samt formelt med jevnlige brukermøter, retrospekter, ukentlige kodegjennomganger og så videre, bidrar til å vedlikeholde og videreutvikle TSD og Educloud som forskerplattformer.
4. Det er nødvendig med en klar rollefordeling, spesielt i forhold til software arkitekter når nye prosjekter skal igang. Uten dette kan utviklingen raskt skli ut og resultere i teknisk gjeld. Dette så vi blant annet relatert til selvbetjeningsportalen i Educloud. Når rollene er ivaretatt blir også utviklingen av softwaren av høy kvalitet og med lav risiko for teknisk gjeld, som vi har sett i TSD.

Disse funnene har vi fått bekreftet og underbygget i to use-cases der vi så på (i) hvordan TSD sin arktitektur ble re-implementert i Educloud og (ii) hvordan Open On Demand ble implementert med Educlouds HPC-ressurs Fox. Også verdt å merke seg i disse use-casene var at den løse strukturen i IT-avdelingen, med stort fokus på kompetanse og egen-initiativ og noe mindre fokus på administrativ kontroll, kan gi en risiko for unødig dobbelt-arbeid, men dette kan synes å betale seg i agilitet og innovative løsninger.

Referanseliste

Bellomo, S., Kruchten, P., Nord, R. L., & Ozkaya, I. (2014). How to Agilely Architect an Agile Architecture. *Cutter IT Journal*.

Bergestuen, S. T., Rachlew, A., & Løken, G.-E. (2020). *Den profesjonelle samtalen: en forskningsbasert intervjumetodikk for alle som stiller spørsmål*. Universitetsforlaget.

Bloomberg, J. (2013). *The Agile Architecture Revolution: How Cloud Computing, REST-Based SOA, and Mobile Computing Are Changing Enterprise IT*. Wiley.

Clarke, C., & Milne, R. J. (2001). *National evaluation of the PEACE investigative interviewing course*. Home Office. http://www.researchgate.net/profile/Colin_Clarke3/publication/263127370_National_Evaluation_of_the_PEACE_Investigative_Interviewing_Course/links/53da3b620cf2e38c63366507.pdf

EU. (2016, May 4). *02016R0679-20160504 - EN - EUR-Lex*. EUR-Lex. Retrieved January 7, 2023, from https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-Based Software Architectures.* (AAI9980887 ed.) [Ph.D. Dissertation]. University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Gamma, E., Helm, R., Ralph Johnson, John Vlissides, & Booch, G. (1995). *Design Patterns*. Addison-Wesley Professional.

Hudak, D., Johnson, D., Chalker, A., Nicklas, J., Franz, E., Dockendorf, T., & McMichael, B. L. (2018). Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software*, *3*(25), 622. 10.21105/joss.00622

Humble, J., & Molesky, J. (2011, August). Why Enterprises Must Adopt Devops to Enable Continuous Delivery. *24, 8*(Cutter IT Journal).

Ludvigsen, S., Hjelmeng, E., Kristensen, S., Nederbragt, L., Jensenius, A., & Vinje, K. (2018, December 12). *Masterplan for universitetets IT*. Universitetets senter for informasjonsteknologi. Retrieved January 7, 2023, from https://www.usit.uio.no/om/it-dir/strategi/masterplan/anbefalinger/arbeidsgruppas-rapport.pdf

Mårtensson, T., Martini, A., Ståhl, D., & Bosch, J. (n.d.). Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles. *2019 IEEE International Conference on Software Architecture (ICSA)*. 10.1109/ICSA.2019.00022

Martini, A., & Bosch, J. (2016). A Multiple Case Study of Continuous Architecting in Large Agile Companies: current gaps and the CAFFEA Framework. *13th Working IEEE/IFIP Conference on Software Architecture*, (13). 10.1109/WICSA.2016.31

McGurk, B. J., Carr, M. J., & McGurk, D. (1993). Investigative Interviewing Courses for Police Officers: An Evaluation. *Police Research Series Paper*, *4*(N/A), 45. https://www.ojp.gov/ncjrs/virtual-library/abstracts/investigative-interviewing-courses-police-officers-evaluation

Mikkelsen, A., Grønli, T.-M., & Kazman, R. (2019). Immutable infrastructure calls for immutable architecture. *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 9. http://hdl.handle.net/10125/60142

Mikkelsen, A., Grønli, T.-M., Tamburri, D. A., Kazman, R., & Mikkelsen, A. (2020). Architectural Principles for Autonomous Microservices. HICSS. *Hawaii International Conference on System Sciences (HICSS)*. https://hdl.handle.net/10125/39610

Øvrelid, E., Bygstad, B., & Thomassen, G. (2021). A Research Platform for Sensitive Data. *Procedia Computer Science*, *181*, 127-134. 10.1016/j.procs.2021.01.112

Thomassen, G. (2019, April 1). *Behov for økt satsning på IT-tjenester for Forskning i UH, Helse og Instituttsektoren – 2020->*. Masterplan for universitetets IT - USIT.

Retrieved January 7, 2023, from https://www-int.usit.uio.no/om/it-dir/strategi/masterplan/arbeidsdokumenter/bidrag/digitalisering-av-forskning-2020-25.pdf

USIT. (n.d.). *Educloud Research - Universitetet i Oslo*. UiO. Retrieved December 19, 2022, from https://www.uio.no/tjenester/it/forskning/plattformer/edu-research/

USIT. (n.d.). *Services for sensitive data (TSD) - University of Oslo*. UiO. Retrieved November 28, 2022, from https://www.uio.no/english/services/it/research/sensitive-data/index.html

USIT. (2009, January 5). *Mål for virksomheten: Hva driver USIT med? - Universitetets senter for informasjonsteknologi*. Universitetets senter for informasjonsteknologi. Retrieved November 28, 2022, from https://www.usit.uio.no/om/formal.html

Waterman, M., Noble, J., & Allan, G. (2015). How Much Up-Front? A Grounded Theory of Agile Architecture. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, (37). 10.1109/ICSE.2015.54