Sikker og stabil drift – måling

[Formål 1](#_Toc122007028)

[Utfordring med å måle sikker og stabil drift 2](#_Toc122007029)

[Måleparametere 2](#_Toc122007030)

[Om systemet 4](#_Toc122007031)

[Målinger 5](#_Toc122007032)

[Kodestørrelse 5](#_Toc122007033)

[Kodestørrelse per 15. desember 2022 5](#_Toc122007034)

[Hva betyr dette? 7](#_Toc122007035)

[Kodeendringer 7](#_Toc122007036)

[Endringer mellom starten og slutten av 3. tertial 7](#_Toc122007037)

[Hva betyr dette? 7](#_Toc122007038)

[Netto vekst i kodelinjer (Kotlin og Typescript) 8](#_Toc122007039)

[Endringer mellom starten og slutten av 3. tertial 8](#_Toc122007040)

[Hva betyr dette? 8](#_Toc122007041)

[Detaljer i netto vekst i kodelinjer 8](#_Toc122007042)

[Endringer mellom starten og slutten av 3. tertial 8](#_Toc122007043)

[Hva betyr dette? 9](#_Toc122007044)

[Oppdatering av avhengigheter 10](#_Toc122007045)

[Så langt i 2022T3 10](#_Toc122007046)

[Hva betyr dette? 11](#_Toc122007047)

[Løpende endringer i systemet 11](#_Toc122007048)

[Så langt i 2022T3 11](#_Toc122007049)

[Hva betyr dette? 15](#_Toc122007050)

[Risiko 16](#_Toc122007051)

[Så langt i 2022T3 17](#_Toc122007052)

[Hva betyr dette? 17](#_Toc122007053)

[Konklusjoner 17](#_Toc122007054)

# Formål

Teamet har blitt bedt om å måle på noe som kan indikere om vi har sikker og stabil drift.

# Utfordring med å måle sikker og stabil drift

Utfordringen med tradisjonell måling av sikker og stabil drift er at den gjerne måler på ting som oppetid eller rettetid. Dette er målinger som først slår ut etter at driften ikke lenger er sikker og/eller stabil, og det kan ta lang tid fra driften faktisk har blitt problematisk til målingene slår ut.

Vår erfaring er at risiko øker når systemer ikke holdes vedlike. Risiko påvirkes av manglende vedlikehold gjennom flere prosesser:

1. Manglende oppdatering av avhengigheter gir økt sikkerhetsrisiko
2. Manglende teknisk oppdatering gir økt sikkerhetsrisiko og risiko for lav kjennskap til utdaterte tekniske løsninger
3. Lite andre endringer reduserer kunnskapen om systemet som øker rettetid og øker sjansen for feil når man en sjelden gang gjør enderinger

Vi mener det er verdt å forsøke å måle på utviklingsprosessen for å se om det kan gi indikatorer som sier noe om risikobildet.

Siden det er kostnader ved å gjøre målinger har vi sett etter målinger som er forholdsvis lette å innhente og som kan gi en indikasjon på at systemet vedlikeholdes.

# Måleparametere

Teamet tok sommeren 2022 i bruk teknikkene *micro-commits* og *Arlo’s commit notation* – dette gjør det mulig å benytte commit-statistikk til å si noe om omfang, intensjon og risiko ved commit’er.

Teamet har besluttet å se om

* tellinger og forholdstall på ulike typer commits og
* måling på kodestørrelse

kan indikere hvordan utviklingen er – og predikere eller illustrere en situasjon der det vil oppstå økt risiko for ustabil / usikker drift.

Grafen viser at det fra juli har vært høy andel commits med angivelse av intensjon og risiko, og derfor egner seg til å gjøre analyse. Siden vi normalt rapporterer på tertialnivå så baserer vi oss i resten av rapporten på tiden fra og med 3. tertial 2022 (2022T3).

# Om systemet

Systemet består av en frontendapplikasjon (saksbehandling-ui), 10 backendapplikasjoner, 7 delte bibliotek på backend og et antall bibliotek på frontend *(som foreløpig ikke er med i oversikten og statistikken).*

|  |  |
| --- | --- |
| **Applikasjoner** |  |
| saksbehandling-ui | Frontend for saksbehandlere |
| oppgave | Oppgaver til Oppgave (asynkron) |
| krav-initialisering | Krav til Pesys (asynkron) |
| statistikk | Statistikk til Statistikk-folket (asynkron) |
| onprem-proxy | For å kalle fra GCP til/fra onprem (Pesys mm) (online) |
| saksbehandling-api | Fasade for diverse API for saksbehandling-ui (online) |
| begrens-innsyn | Merking av saker med beskyttelse (asynkron) |
| journalforing | Journalføring av dokumenter (asynkron) |
| prefill | Preutfylling av SED (online) |
| fagmodul | Diverse API for saksbehandling-ui m fl (online) |
| pdl-produsent | Oppdatering av PDL med id’er og adresser (asynkron) |
| **Library** |  |
| ep-metrics | Metrikk-bibliotek |
| ep-eux | Domene-modell SED/BUC mm |
| ep-logging | Loggingsbibliotek |
| ep-personoppslag | Klient for oppslag mot PDL |
| ep-security-sts | Brukes onprem for token-utveksling |
| ep-pensjonsinformasjon | Klient for oppslag mot Pesys |
| ep-kodeverk | Klient mm - opplag i kodeverk |
| **Meta** |  |
| Meta | Verktøy for å jobbe på tvers av modulene nevnt ovenfor |

# Målinger

## Kodestørrelse

Kodestørrelse i applikasjoner korrelerer godt med kompleksitet, som igjen korrelerer med feil- og vedlikeholdsomfang.

Vi teller kodelinjer med cloc[[1]](#footnote-1). (Det er noen små frontend-bibliotek som deles mellom EESSI NAV og EESSI Pensjon som ikke er med i statistikken).

### Kodestørrelse per 15. desember 2022

Koden er stort sett skrevet i Kotlin og TypeScript, og noe LESS– øvrig kode er knyttet til bygg, deploy, konfigurasjon og diverse utviklings- og støtteverktøy.

|  |  |  |  |
| --- | --- | --- | --- |
| **Language** | **No of files** | **Code lines** | **Comment lines** |
| Kotlin | 754 | 62859 | 2328 |
| TypeScript | 363 | 43914 | 1049 |
| YAML | 142 | 5456 | 56 |
| Text | 14 | 5388 | 0 |
| Markdown | 35 | 1501 | 0 |
| Gradle | 35 | 1035 | 120 |
| LESS | 21 | 1028 | 50 |
| GraphQL | 9 | 946 | 71 |
| JavaScript | 10 | 337 | 60 |
| Bourne Shell | 21 | 250 | 174 |
| PlantUML | 3 | 188 | 0 |
| Python | 5 | 186 | 2 |
| make | 2 | 113 | 0 |
| Dockerfile | 12 | 53 | 0 |
| HTML | 2 | 41 | 0 |
| Properties | 15 | 15 | 7 |
| SVG | 4 | 4 | 0 |
| **Grand Total** | **1447** | **123314** | **3917** |

Frontend-app’en for saksbehandling utgjør om lag en tredel av den totale koden, i én app.

### Hva betyr dette?

Teamet har i dag en frontendutvikler på halv tid (som kom inn i teamet i sommer og den første tiden jobbet bare en dag i uken). Det tar tid å bygge opp kompetanse på en kodebase av denne størrelsen, og teamet er spesielt sårbart på frontend.

10-15% av koden er ikke i de to primære programmeringsspråkene (Kotlin og TypeScript). Dette gjenspeiler til dels at teamet også har hånd om bygg, deploy og applikasjonsdrift. (Koden for å lage underlag for denne rapporten er også en del av dette.)

## Kodeendringer

VI kan se på hvilke app’er det er gjort kodeendringer i ved å sammenlikne koden på starten og slutten av tertialet.

### Endringer mellom starten og slutten av 3. tertial

Vi ser at det er mest endring i kodelinjer i pdl-produsent og i frontenden for saksbehandlere.

Endringer i kodelinjer er en indikasjon, men den påvirkes av ting som reformattering av koden så det kan være vanskelig å si hva årsaken er.

### Hva betyr dette?

En komponent er ikke rørt i det hele tatt i tertialet («ui» innholder bibliotek for frontend). Vi ser at det vi har jobbet mest med pdl-produsent (adresser til PDL) og saksbehandling-ui (trygdetidtabell) også dukker opp i denne statistikken.

## Netto vekst i kodelinjer (Kotlin og Typescript)

Netto vekst i kodelinjer indikerer at det blir mer kode å vedlikeholde. Refaktorering kan til en viss grad motvirke veksten.

### Endringer mellom starten og slutten av 3. tertial

### Hva betyr dette?

Dette viser at systemet øker i størrelse for språkene man stort sett bruker til å implementere tester og funksjonalitet. Det har tydeligvis også skjedd noe opprydning for to moduler har minsket i størrelse, dette kan skyldes at kode er flyttet til bibliotek, eller annen opprydning. Mest sannsynlig har opprydning redusert størrelsen på koden i flere moduler, men tilveksten har spist opp gevinsten fra opprydningen.

## Detaljer i netto vekst i kodelinjer

### Endringer mellom starten og slutten av 3. tertial

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Module** | **Kotlin** | **Gradle** | **TypeScript** | **Python** | **Markdown** | **YAML** | **Bourne Shell** | **Dockerfile** | **make** | **HTML** | **Properties** | **JavaScript** | **Text** | **Grand Total** |
| meta | 28 | 131 |  | 186 | 126 | 75 | 25 | 9 | 6 |  |  |  | -3 | 583 |
| ep-kodeverk | 380 | 64 |  |  | 4 | 10 |  |  |  |  | 0 |  | 19 | 477 |
| pdl-produsent | 323 | 14 |  |  |  | 15 | 4 | 1 |  |  |  |  |  | 357 |
| saksbehandling-ui |  |  | 371 |  |  | -103 |  |  |  | 1 |  | 0 |  | 269 |
| ep-metrics | 119 | 45 |  |  |  | 14 |  |  |  |  | 0 |  |  | 178 |
| begrens-innsyn | 106 | 25 |  |  |  | 10 | 4 | 1 |  |  |  |  |  | 146 |
| statistikk | 95 | 26 |  |  |  | 7 | 4 | 1 |  |  |  |  |  | 133 |
| oppgave | 118 | 8 |  |  |  | -25 | 4 | 1 |  |  |  |  |  | 106 |
| journalforing | 80 | 12 |  |  |  | 5 | 4 | 1 |  |  |  |  |  | 102 |
| krav-initialisering | 42 | 24 |  |  |  | 7 | 4 | -2 |  |  |  |  |  | 75 |
| saksbehandling-api | 31 | 26 |  |  |  | 11 | 4 | 1 |  |  |  |  |  | 73 |
| ep-security-sts | 30 | 36 |  |  |  | 6 |  |  |  |  | 0 |  |  | 72 |
| ep-eux | 29 | 38 |  |  |  | 4 |  |  |  |  | 0 |  |  | 71 |
| prefill | 26 | 22 |  |  |  | 16 | 4 | 1 |  |  |  |  |  | 69 |
| ep-logging | 8 | 49 |  |  |  | 10 |  |  |  |  | 0 |  |  | 67 |
| ep-personoppslag | 75 | -2 |  |  |  | -27 |  |  |  |  | 0 |  |  | 46 |
| ep-pensjonsinformasjon |  | 36 |  |  |  | 6 |  |  |  |  | 0 |  |  | 42 |
| onprem-proxy | -26 | 24 |  |  |  | 19 | 4 | 1 |  |  |  |  | -17 | 5 |
| fagmodul | -78 | 26 |  |  |  | 16 | 4 | 1 |  |  |  |  |  | -31 |
| **Grand Total** | **1386** | **604** | **371** | **186** | **130** | **76** | **65** | **16** | **6** | **1** | **0** | **0** | **-1** | **2840** |

Ved siden av økningen i kodelinjer med Kotlin og Typescript ser vi at det er mye økning i kodelinjer med gradle. (Dette er imidlertid ikke helt reelt fordi vi har omorganisert mye av gradle-koden basert på en mekanisme der vi endrer ett sted og så kopierer ut endringene til hver modul. Om vi teller de kopierte filene én gang så er det reelt sett en reduksjon i antall linjer med gradle-kode.)

Det er også lagt til en del Python-kode.

### Hva betyr dette?

Å bare se på produksjonskoden gir ikke hele bildet av hva systemet består av.

Teamet må beherske endringer også for andre språk/teknologier.

Statistikken kan gi et skjevt bilde av utviklingen av vedlikeholdsbehovet (som for gradle i dette tilfellet).

## Oppdatering av avhengigheter

Oppdateringer av avhengigheter reduserer risiko for sikkerhetshull, og er med på å redusere problemene med sporadiske oppdateringer (mange ting som oppdateres samtidig, og at store hopp i versjoner gjør det vanskeligere å finne feil dersom noe går galt).

### Så langt i 2022T3

Teamet har så langt i 2022T3 gjort et stort antall oppdateringer på avhengigheter (i all hovedsak er dette oppdateringer til nye minor- og patch-versjoner).

Backend-modulene i systemet gjøres det jevnlig oppdateringer på, teamet forsøker å gjøre oppdateringer ukentlig («patch Tuesday»). Dette fører til mange oppdateringer, men jobben med oppdateringene er relativt liten – prosessen er støttet med script som gjør det enklere. Fremgangsmåten teamet benytter fører til at det blir mange små endringer (stort sett en endring pr avhengighet pr modul).

På frontend er det ikke noen fast rutine for dette, og det er også litt dårlig statistikk på dette fordi mekanismene og rutinene er forskjellige.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Count of commits** | **Måned** |  |  |  |  |
| **Module** | **9** | **10** | **11** | **12** | **Grand Total** |
| **app** | **220** | **195** | **253** | **128** | **796** |
| saksbehandling-ui | 1 | 1 |  |  | 2 |
| oppgave | 15 | 9 | 16 | 8 | 48 |
| krav-initialisering | 16 | 13 | 19 | 6 | 54 |
| statistikk | 20 | 13 | 23 | 9 | 65 |
| saksbehandling-api | 21 | 15 | 26 | 9 | 71 |
| onprem-proxy | 21 | 14 | 25 | 13 | 73 |
| begrens-innsyn | 21 | 15 | 23 | 15 | 74 |
| journalforing | 21 | 18 | 26 | 13 | 78 |
| prefill | 30 | 24 | 33 | 19 | 106 |
| fagmodul | 29 | 27 | 32 | 20 | 108 |
| pdl-produsent | 25 | 46 | 30 | 16 | 117 |
| **library** | **62** | **28** | **64** | **26** | **180** |
| ep-metrics | 5 | 1 | 5 | 3 | 14 |
| ep-eux | 6 | 1 | 7 | 3 | 17 |
| ep-logging | 5 | 3 | 6 | 3 | 17 |
| ep-personoppslag | 7 | 5 | 3 | 4 | 19 |
| ep-security-sts | 11 | 4 | 14 | 2 | 31 |
| ep-pensjonsinformasjon | 14 | 4 | 13 | 2 | 33 |
| ep-kodeverk | 14 | 10 | 16 | 9 | 49 |
| **meta** |  | **1** | **3** | **1** | **5** |
| meta |  | 1 | 3 | 1 | 5 |
| **Grand Total** | **282** | **224** | **320** | **155** | **981** |

*Oppdateringer av avhengigheter pr modul og måned (inntil 9. desember)*

### Hva betyr dette?

For backend-modulene ser det ut til at det er jevnlig oppdatering, for frontend-app’en saksbehandling-ui bør det nok ses nærmere på hvorfor det telles så få oppdateringer av avhengigheter.

## Løpende endringer i systemet

At det gjøres endringer i systemet er indikator på at systemet vedlikeholdes, noe som opprettholder kjennskapen til systemet. Vi ser her på endringer merket med B, D, F, R og T; de er knyttet til feilretting (B), dokumentasjon (nær/i koden) (D), funksjonalitet (F), interne forbedringer/refaktorering (R) og testforbedringer (T).

### Så langt i 2022T3

Vi ser det er gjort omkring 400 endringer som er av typer som krever litt innsikt i systemet (BDFRT), de er med andre ord er de med på å opprettholde kjennskapen til systemet.

Om vi snevrer inn til feilrettings-, refaktorerings- og funksjonalitetsrelaterte endringer så ser vi at feilretting utgjør litt over en tidel av disse endringene, og refaktorering en drøy tredel.

Feilrettingsendringer (B) er ikke nødvendigvis knyttet til feil som har truffet produksjon, men av disse endringene er knyttet til det. Ellers er feilene som regel nyoppdaget (vi prøver å ta unna når vi oppdager dem), men det er ikke lett å si om de er gamle eller innført nylig.

Endringene er merket refaktorering (R) er endringer som gjøres for å forbedre den interne kvaliteten, og ikke minst for at koden (som ofte er skrevet av noen andre) skal bli lettere å forstå for den som jobber med den i øyeblikket, eller i fremtiden. Sannsynligvis vil det gjøre systemet enklere å endre neste gang, med mindre risiko for feil og bittelitt mindre innsats. Dersom andelen refaktoreringsendringer faller er det viktig å undersøke hva som skjer.

I 2022T3 er det gjort flest endringer på pdl-produsent.

Dette er i stor grad knyttet til at det er implementert automatisk oppdatering av utenlandske kontaktadresser i PDL. Funksjonaliteten som er bygget i pdl-produsent er middels komplisert, men avgrenset i omfang.

Det er gjort lite arbeid på flere viktige moduler, spesielt gjelder det journalføring og oppgave-modulen (mens prefill og fagmodul som er to andre viktige komponenter, har noe endringer).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **modules** | **B** | **D** | **F** | **R** | **T** | **Total** |
| **app** | **34** | **14** | **138** | **97** | **39** | **322** |
| pdl-produsent | 13 | 3 | 59 | 52 | 29 | 156 |
| fagmodul | 3 | 5 | 17 | 24 | 2 | 51 |
| saksbehandling-ui | 12 |  | 11 | 12 | 2 | 37 |
| prefill |  |  | 14 | 7 | 4 | 25 |
| journalforing |  |  | 10 | 1 | 1 | 12 |
| statistikk | 4 | 2 | 5 |  |  | 11 |
| oppgave |  | 2 | 6 |  | 1 | 9 |
| begrens-innsyn | 2 |  | 5 |  |  | 7 |
| krav-initialisering |  | 2 | 4 |  |  | 6 |
| onprem-proxy |  |  | 4 |  |  | 4 |
| saksbehandling-api |  |  | 3 | 1 |  | 4 |
| **library** | **2** | **2** | **16** | **6** | **1** | **27** |
| ep-kodeverk | 2 | 1 | 7 | 3 |  | 13 |
| ep-metrics |  | 1 | 3 | 2 | 1 | 7 |
| ep-personoppslag |  |  | 4 |  |  | 4 |
| ep-eux |  |  | 1 | 1 |  | 2 |
| ep-logging |  |  | 1 |  |  | 1 |
| **meta** |  | **6** | **1** | **1** |  | **8** |
| meta |  | 6 | 1 | 1 |  | 8 |
| **Total** | **36** | **22** | **155** | **104** | **40** | **357** |

*BDFRT-oppdateringer pr modul i 2022T3 (inntil 9. desember)*

Vi ser at det er mest feilretting (B) på pdl-produsent og saksbehandling-ui.

### Hva betyr dette?

Kunnskapen om pdl-produsent er god etter nylig arbeid på den, og vi ser at de tre viktige modulene saksbehandling-ui, fagmodul og prefill er gjort litt på, mens to andre viktige moduler (journalføring og oppgave) er gjort lite arbeid på, noe som er bekymringsfullt med tanke på forvitring av kompetansen på disse.

De øvrige modulene med relativt lite endringer er litt mindre bekymringsfulle siden de er relativt små og enkle.

De feil-relaterte endringene på pdl-produsent og saksbehandling-ui har trolig ulike årsaker. For pdl-produsent handler det trolig om at det har blitt gjort mye endringer – og endringer korrelerer gjerne med at det introduseres – eller oppdages – feil. For saksbehandling-ui kan det kanskje også handle om at det er feil som ikke er nye, men at man der har lite (og har hatt enda mindre) kapasitet så feil har fått samle seg opp før de har blitt rettet.

## Risiko

Vi merker commits med risiko, med en skala fra 0 til 5:

1. Ingen risiko
2. Kjent trygg
3. Validert
4. Kjent restrisiko
5. Feil/sannsynlig feil
6. Ukjent risiko

Endringer med risiko 1 Kjent trygg har svært lav risiko og krever lite eller ingen kjøring av tester. Dokumentasjons- og testendringer skal ha risiko 1. For endringer i produksjonskode er det strenge regler for hvilke endringer som kan merkes risiko 1 – stort sett går det ut på at det er små endringer som er gjort med støtte av funksjonalitet i utviklingsverktøyet (IDE).

Endringer med risiko 2 Validert krever som regel at tester kjøres lokalt for at man skal bli trygg, noe som kan ta litt tid, avhengig av app og oppsett. For oppdateringer av avhengigheter er det ofte vanskelig å si hva risiko er, men vi kjører tester for alle oppdateringer så vi merker dem rutinemessig med risiko 2.

Endringer med risiko 3 Kjent restrisiko må ofte følges opp litt ekstra i forbindelse med produksjonssetting, eller de må testes manuelt før de går i produksjon.

Risiko 4 er normalt ikke akseptabelt å dytte til produksjon – dette er endringer som er uferdige eller har høy risiko.

Risiko 5 betyr at risiko er ukjent. Ofte vet man mer i situasjonen, men kunnskapen om risiko er tapt i ettertid (må eventuelt vurderes på nytt).

Det er nok litt mer slurv med vurdering av risikonivå enn det er med vurdering av intensjon.

De fleste endringer i produksjonskode vil være risiko fra 1 til 3.

### Så langt i 2022T3

De kjent trygge endringene utgjør ca 40%. For commits som typisk endrer produksjonskode ser vi at andelen med kjent restrisiko er under 10%.

### Hva betyr dette?

Få endringer med restrisiko er bra – det betyr mindre jobb med å følge opp endringen på vei ut i produksjon. Kjent trygge endringer er ofte endringer som bedrer lesbarhet og struktur i koden, at det er på 40% betyr trolig at det er mange små forbedringer.

# Konklusjoner

... skriver vi når tertialet er over ☺

1. cloc --csv --vcs git --exclude-dir=dist,build,gradle --exclude-list-file=.clocignore --exclude-lang=JSON,XML --quiet .  
   der .clocignore inneholder:   
   gradlew  
   gradlew.bat  
   public/pdf.worker.js  
   public/static/js/pdf.worker.js  
   package-lock.json  
   src/components/PostalCodes/Postal-codes-Norway-ansi.ts  
   src/minibootstrap.less [↑](#footnote-ref-1)