Utviklingsstatistikk

Oppsummering

I tredje tertial har teamet hatt en frontend-utvikler i 50 % stilling, samtidig som 1/3 av kodebasen er frontend-kode. Basert på oppdatering av avhengigheter, som det tilsynelatende skjer lite av på frontend, så mistenker vi at har hatt for liten kapasitet til å holde frontend-kode oppdatert. Det kan hende at situasjonen bedres litt ettersom utvikler blir mer kjent med koden, men vi er bekymret for kapasiteten. Teamet har i tredje tertial hatt nesten tre backend utviklere, men fra årsskiftet er det bare to som skal vedlikeholde de ti backend applikasjonene teamet har.

I siste tertial så vi at det var en del aktivitet på pdl-produsent, fagmodulen og prefill, men lite på spesielt journalføring og oppgave som er viktige applikasjoner. Når det skjer lite på viktige applikasjoner så forvitrer fort kunnskapen på disse og det tar lenger tid å gjøre endringer og sjansen for feil øker. Med forrige tertials bemanning fikk vi implementert ny funksjonalitet for utenlandsk ident og utenlandsk adresse ved oppdatering av PDL. Bortsett fra dette har vi ikke introdusert ny funksjonalitet for saksbehandlere. Med dagens bemanning er det svært liten kapasitet til å utvikle ny funksjonalitet, det er kun tilstrekkelig til å sikre daglig drift.

Formål

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

Hva betyr dette?

Teamet har i dag én 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.)

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.

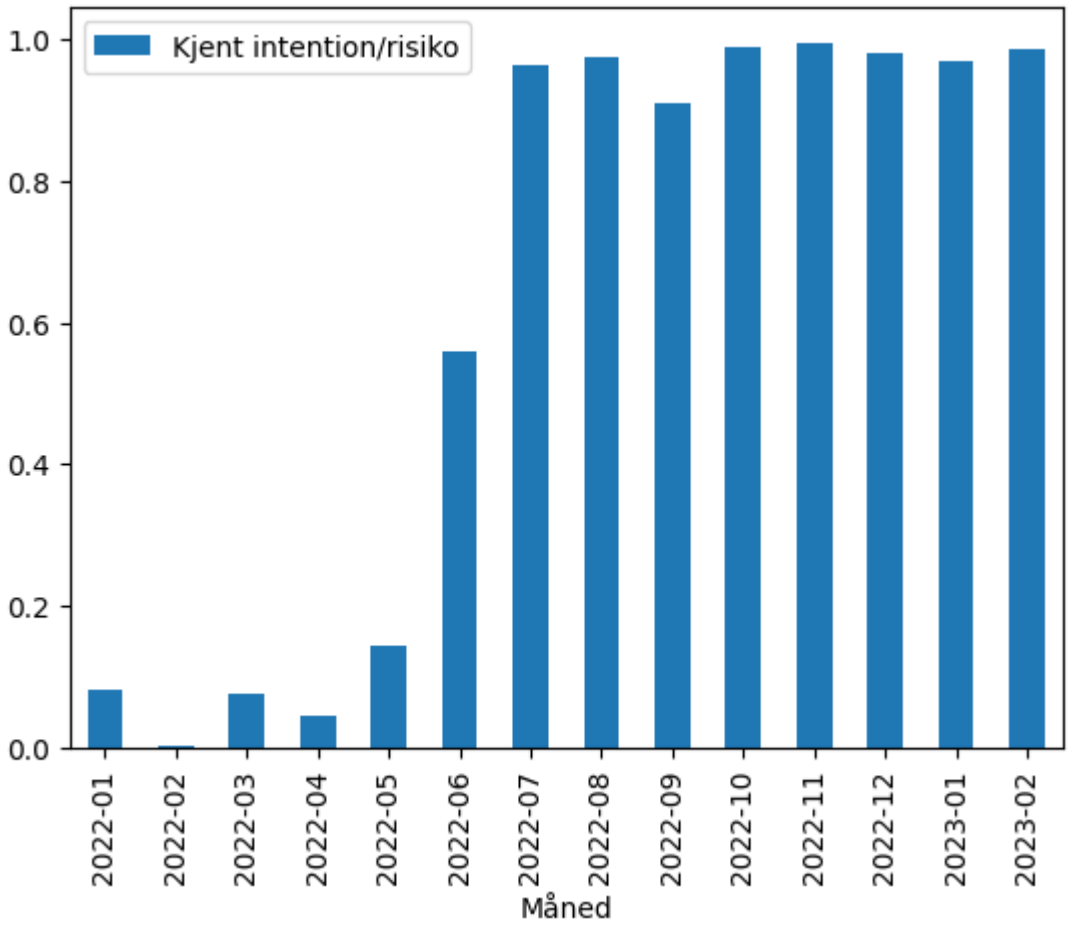
Siden det er kostnader ved å gjøre målinger har vi sett etter målinger som er forholdsvis lette å innhente og som kan gi en indikasjon på at systemet vedlikeholdes.

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 2022 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)

Applikasjoner	
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)

Bibliotek	
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.

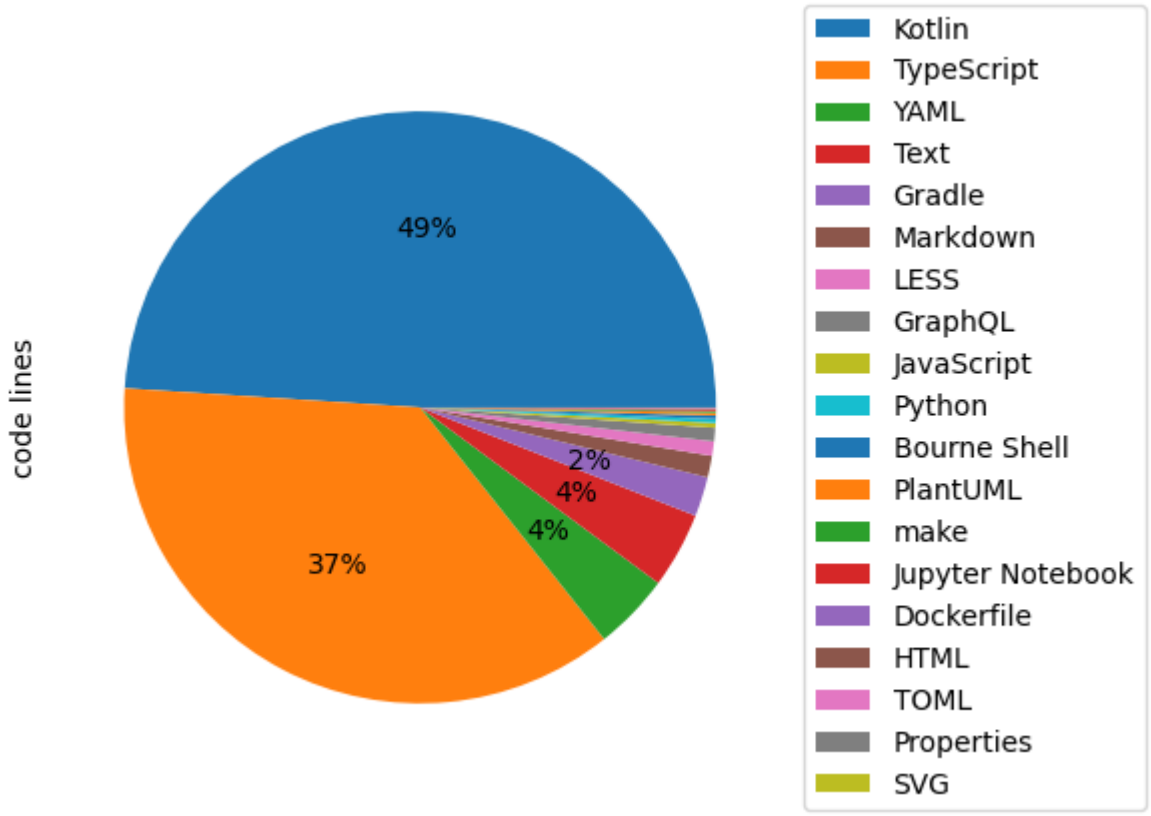
Vi teller kodelinjer med cloc[^1]. (Det er noen små frontend-bibliotek som deles mellom EESSI NAV og EESSI Pensjon som ikke er med i statistikken).

Kodestørrelse

`Kodestørrelse pr 14.02.2023.`

Out[9]:

language	files	code lines	comment lines
Kotlin	758	63622	2450
TypeScript	369	47707	1041
YAML	142	5456	56
Text	14	5388	0
Gradle	59	2814	159
Markdown	37	1525	2
LESS	21	1028	50
GraphQL	9	946	71
JavaScript	10	337	60
Python	6	280	2
Bourne Shell	21	250	174
PlantUML	3	188	0
make	3	129	0
Jupyter Notebook	1	117	2435
Dockerfile	12	53	0
HTML	2	41	0
TOML	1	19	0
Properties	15	15	7
SVG	4	4	0



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.

[^1]: Mer om cloc-kommandoen:

```
cloc --csv --vcs git --exclude-dir=dist,build,gradle --
exclude-list-file=.clocignore --exclude-lang=JSON,XML --quiet
.
```

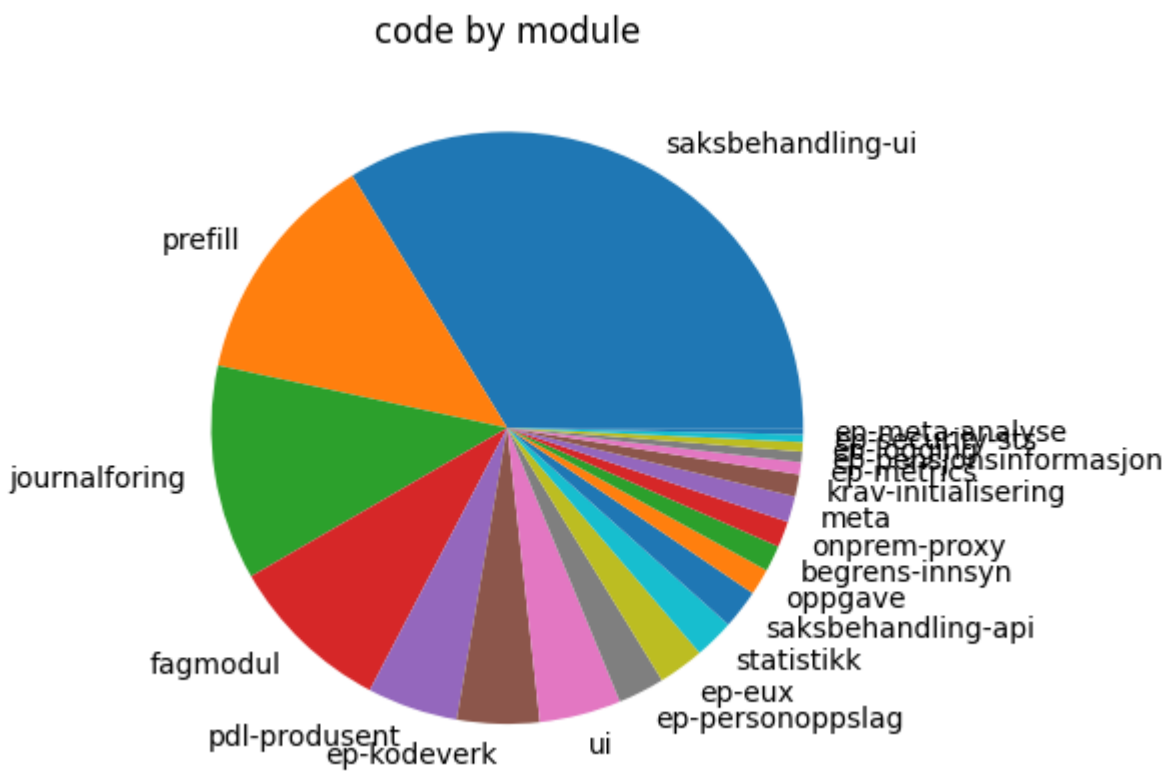
der .clocignore inneholder:

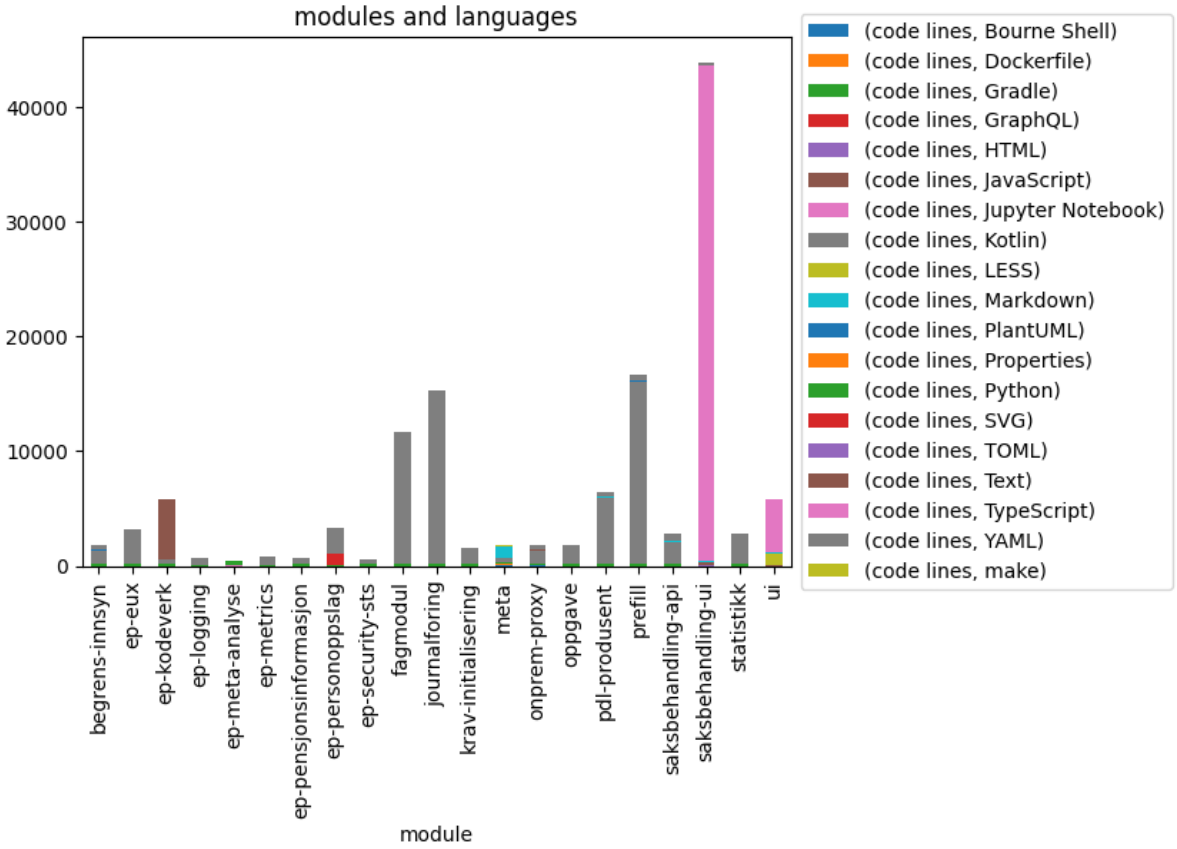
```
gradlew
gradlew.bat
public/pdf.worker.js
public/static/js/pdf.worker.js
package-lock.json
src/components/PostalCodes/Postal-codes-Norway-ansi.ts
src/minibootstrap.less
```

Out[11]:

module	code lines	code lines %
saksbehandling-ui	43875	33.771042
prefill	16688	12.844926
journalforing	15253	11.740392
fagmodul	11634	8.954810
pdl-produsent	6440	4.956935
ep-kodeverk	5835	4.491260
ui	5801	4.465090
ep-personoppslag	3296	2.536965
ep-eux	3251	2.502328
statistikk	2821	2.171353
saksbehandling-api	2780	2.139795
oppgave	1861	1.432431
begrens-innsyn	1858	1.430122
onprem-proxy	1852	1.425504
meta	1833	1.410879
krav-initialisering	1567	1.206136
ep-metrics	864	0.665030
ep-pensjonsinformasjon	752	0.578822
ep-logging	665	0.511857
ep-security-sts	535	0.411795
ep-meta-analyse	458	0.352527

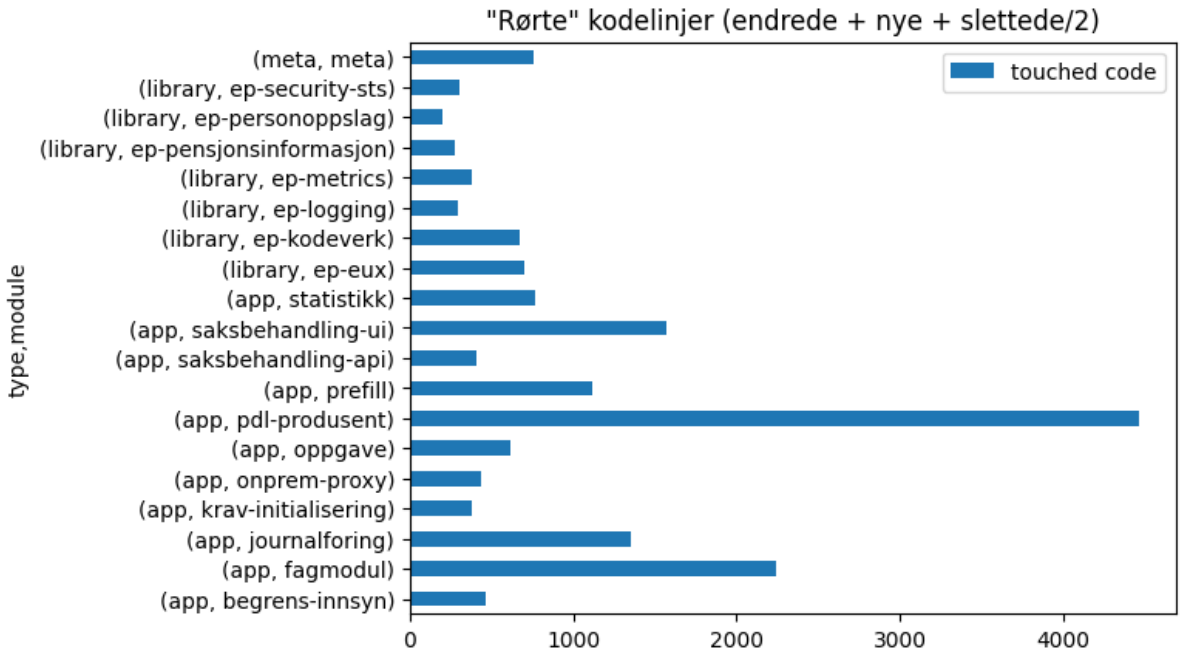
Frontend-app'en for saksbehandling utgjør om lag en tredel av den totale koden, i én app.





Kodeendringer

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



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

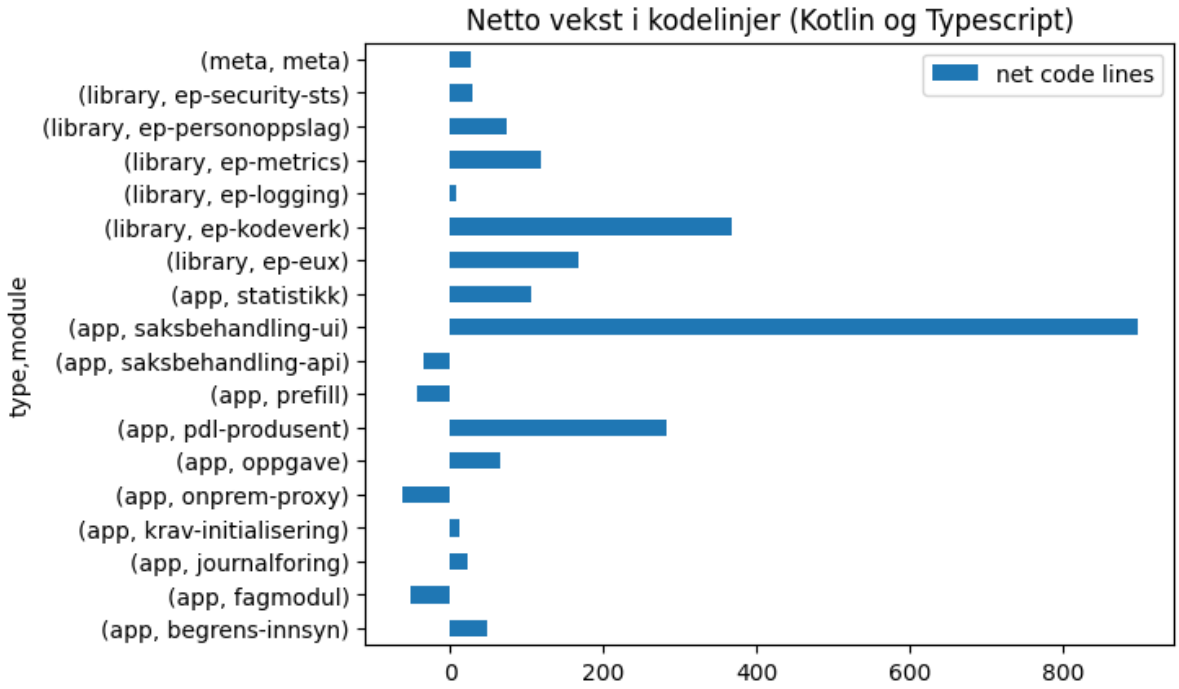
Endringer i kodelinjer er en indikasjon, men den påvirkes av ting som reformattering av koden så det kan være vanskelig å si hva årsaken er.

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

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



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 noen 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

Out[16]:

language	Bourne Shell	Dockerfile	Gradle	HTML	Kotlin	Markdown	Properties	Pyth
module								
saksbehandling-ui	0	0	0	1	0	0	0	
meta	25	9	131	0	28	126	0	
ep-kodeverk	0	0	64	0	368	4	0	
pdl-produsent	4	1	14	0	282	0	0	
ep-eux	0	0	38	0	168	0	0	
ep-metrics	0	0	45	0	119	0	0	
statistikk	4	1	26	0	106	0	0	
begrens-innsyn	4	1	24	0	49	0	0	
ep-security-sts	0	0	35	0	30	0	0	
ep-logging	0	0	49	0	8	0	0	
oppgave	4	1	8	0	65	0	0	
krav-initialisering	4	-2	27	0	12	0	0	
journalforing	4	1	12	0	24	0	0	
ep-personoppslag	0	0	-3	0	75	0	0	
ep-pensjonsinformasjon	0	0	36	0	0	0	0	
saksbehandling-api	4	1	26	0	-35	0	0	
prefill	4	1	22	0	-43	0	0	
fagmodul	4	1	25	0	-51	0	0	
onprem-proxy	4	1	23	0	-62	0	0	
Total	65	16	602	1	1143	130	0	

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.)

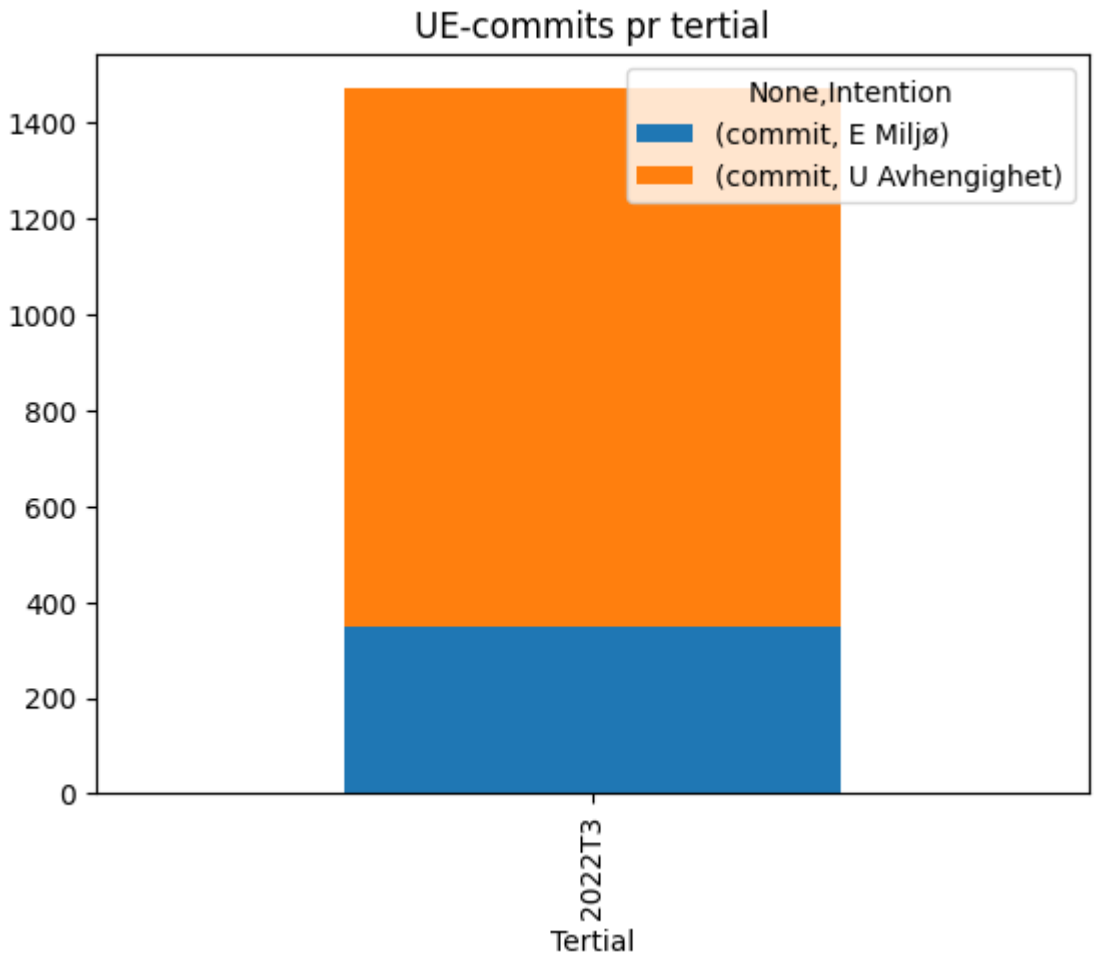
Det er også lagt til en del Python-kode.

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).



Teamet har 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.

Oppdateringer av avhengigheter pr modul og måned

Out[18]:

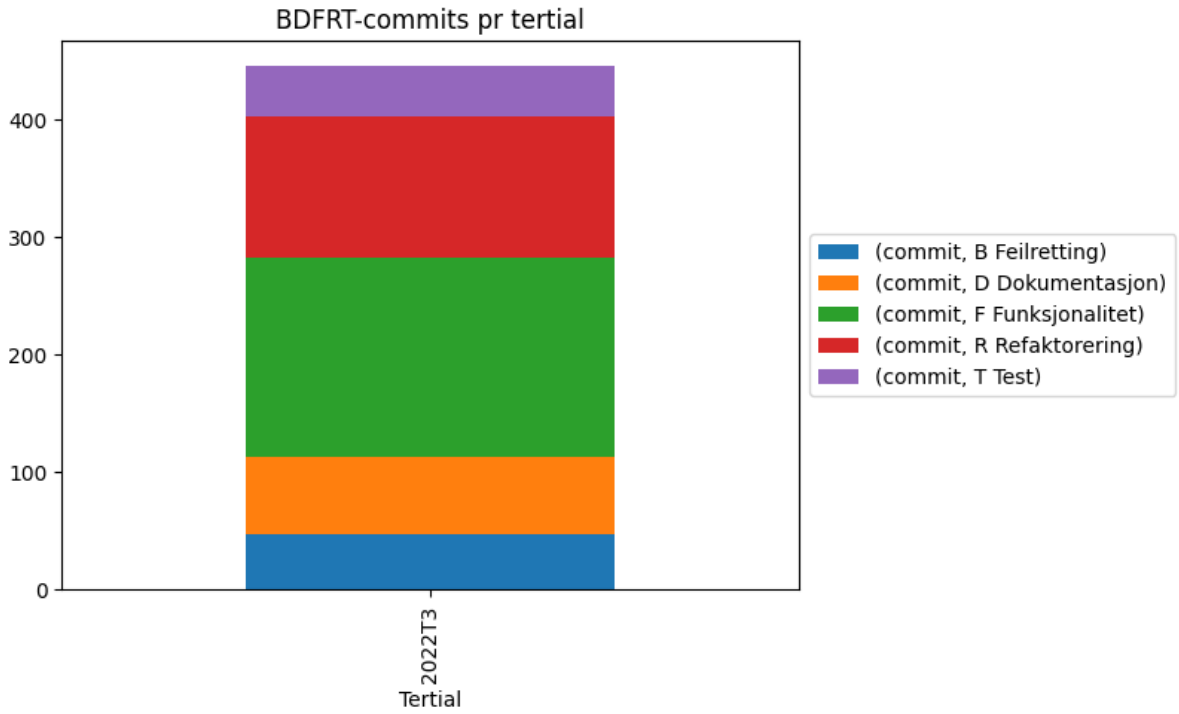
		commit				Total
	Måned	2022-09	2022-10	2022-11	2022-12	
type	module					
app	pdl-produsent	25	46	30	31	132
	fagmodul	29	27	32	36	124
	prefill	30	24	33	37	124
	journalforing	21	18	26	24	89
	begrens-innsyn	21	15	23	27	86
	onprem-proxy	21	14	25	24	84
	saksbehandling-api	21	15	26	19	81
	statistikk	20	13	23	22	78
	krav-initialisering	16	13	19	12	60
	oppgave	15	9	16	14	54
	saksbehandling-ui	1	1	0	0	2
library	ep-kodeverk	14	10	16	13	53
	ep-pensjonsinformasjon	14	4	13	6	37
	ep-security-sts	11	4	14	5	34
	ep-personoppslag	7	5	3	9	24
	ep-eux	6	1	7	4	18
	ep-logging	5	3	6	4	18
	ep-metrics	5	1	5	5	16
meta	meta	0	1	3	1	5

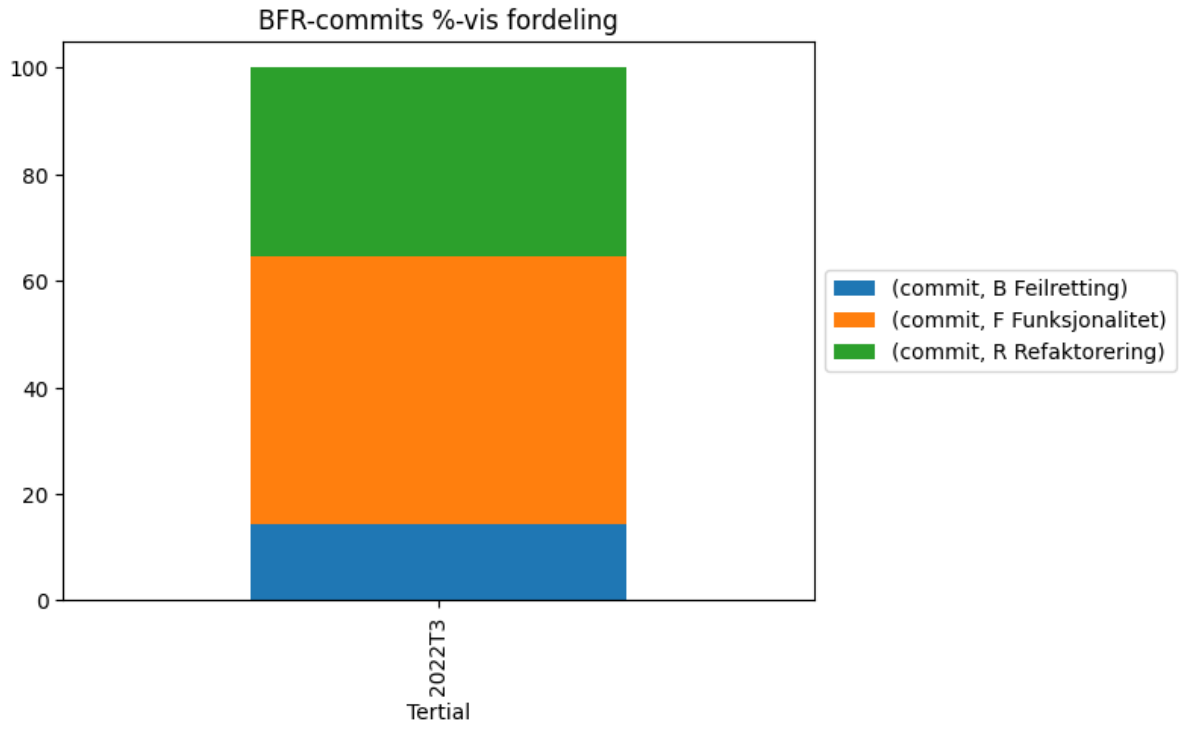
Hva betyr dette?

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).

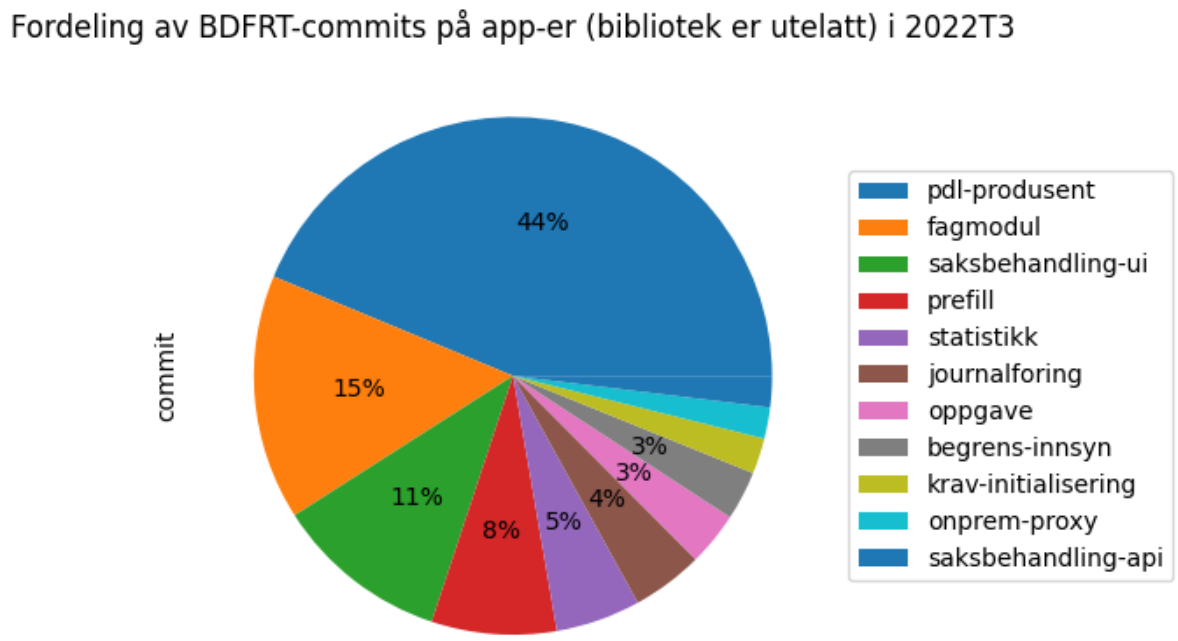




Om vi snevrer inn til feilrettings-, refaktorerings- og funksjonalitetsrelaterte endringer så ser vi at feilretting utgjør litt over en tidel av disse endringene, og refaktorering en drøy tredel.

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).

BDFRT-oppdateringer pr modul

Out[22]:

		commit					Total
	intention	B	D	F	R	T	
type	module						
app	pdl-produsent	13	4	59	52	29	157
	fagmodul	3	6	18	25	3	55
	saksbehandling-ui	13	1	11	12	2	39
	prefill	0	1	14	9	4	28
	statistikk	7	5	5	2	0	19
	journalforing	1	1	11	2	1	16
	oppgave	0	3	6	2	1	12
	begrens-innsyn	2	2	7	0	0	11
	krav-initialisering	0	3	4	1	0	8
	onprem-proxy	0	1	5	1	0	7
	saksbehandling-api	0	1	3	2	1	7
library	ep-eux	7	1	9	4	1	22
	ep-kodeverk	2	2	8	4	0	16
	ep-metrics	0	2	3	2	1	8
	ep-personoppslag	0	1	4	0	0	5
	ep-logging	0	1	1	0	0	2
	ep-security-sts	0	1	1	0	0	2
	ep-pensjonsinformasjon	0	1	0	0	0	1
	ui	0	1	0	0	0	1
meta	meta	0	19	1	1	0	21
	ep-meta-analyse	0	8	0	0	0	8

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:

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

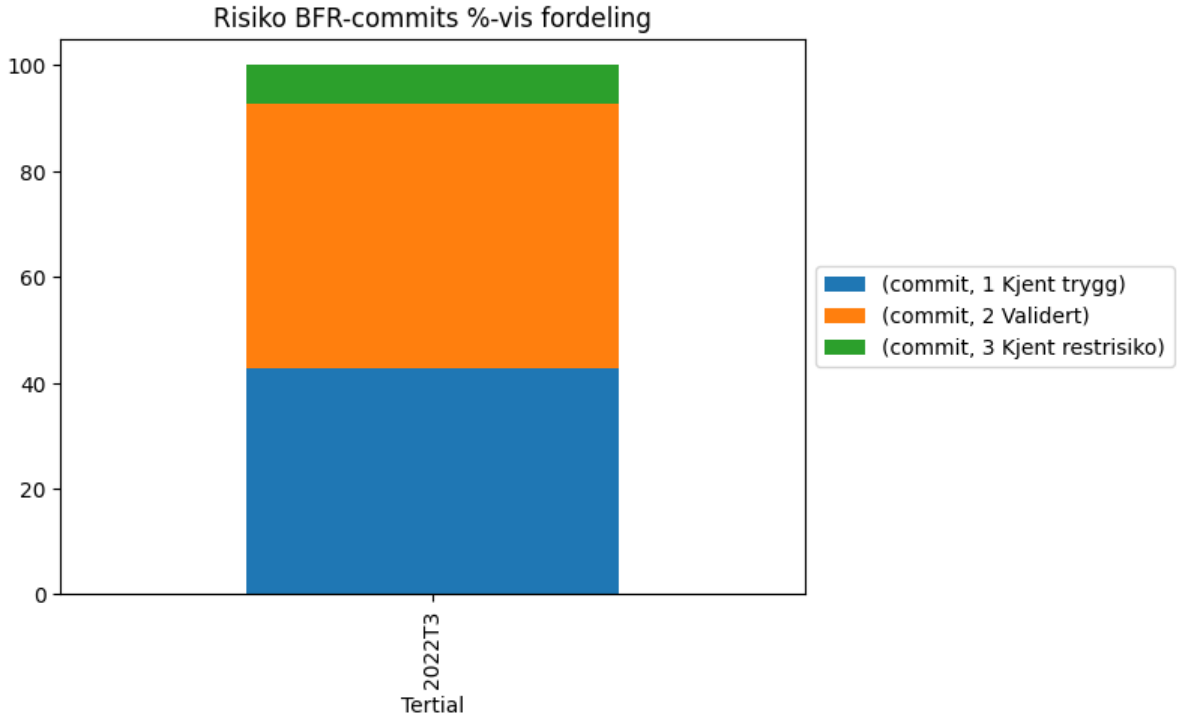
Endringer med risiko 3 Kjent restrisiko må ofte følges opp litt ekstra i forbindelse med produksjonssetting, eller de må testes manuelt før de går i produksjon.

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



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 commiter 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.