

**Forfatter: David Elboth**

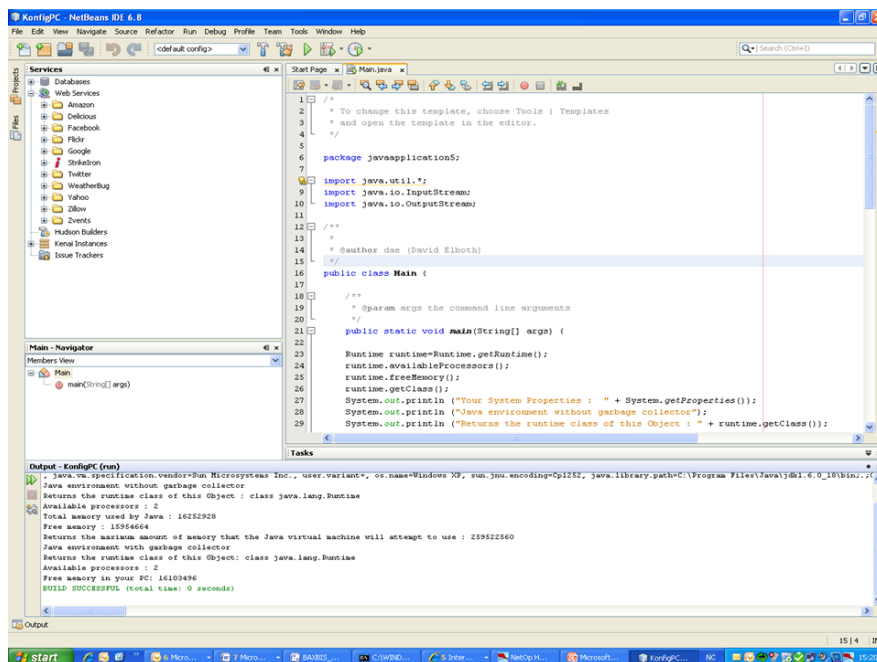
## **Nets sin bruk av Open Source komponenter i forretningsapplikasjoner.**

*Nets har tidligere gjennomført en feltstudie hvor vi har samlet inn data fra 14 større utviklingsprosjekter (innenfor to forretningsområder i Nets). Resultatet av de innsamlede data viser at 76 % av kildekoden er basert på Open Source biblioteker og et utviklingsmiljø og produksjonsmiljø som er basert på Open Source programvare.*

De siste årene har OSS (Open Source Software) eller det generaliserte åpent digitalt innhold fått en stor utbredelse i norsk IKT. Analyser gjort av blant annet SSB (Statistisk sentralbyrå) viser at over 50 % av alle norske programvareprodusenter nå bruker frie OSS-komponenter i sine systemløsninger. Den store andelen gjenbruk vi har i dag gir det norske samfunnet årlige besparelser på milliarder av kroner. Vi har i vår analyse under sett konkret hvor mye OSS vi bruker både i form av servertjenester/applikasjoner og i form av kode (les: biblioteker) i kritiske forretningsapplikasjoner. Rapporten gjenspeiler resultatet av 14 interne utviklingsprosjekter i Nets. Vi har ikke gjort noen analyse på hva Nets har spart ved å bruke OSS.

### **Utviklingsrammeverk basert på Java**

Mesteparten av Nets utvikling er basert på Java med bruk av vanlige Java-biblioteker og felles OSS- rammeverk som Spring og Hibernate. Prosjektene velger selv de åpne-kildekode-biblioteker som de har behov for.



**Figur 1: Viser arbeidsflaten fra Netbeans**

Vi bruker Maven som byggverktøy dermed kan enhver utvikler fritt velge IDE (Integrated Development Environment). Maven sørger for at korrekte versjoner av tredjepartsbiblioteker benyttes, bygger, pakker og distribuerer programvaren. Eclipse er i dag den mest brukte IDE, men IntelliJ IDEA er også populært i enkelte avdelinger. Vi anslår i dag at 60 % av alle utviklere bruker Eclipse, og resten bruker IntelliJ IDEA eller NetBeans. Eclipse, IntelliJ IDEA og NetBeans er gratis og OSS-verktøy. IDE-verktøyene gir en sømløs integrasjon mellom editor, kompilator, debugger, kildekodekontroll, test etc. Med et IDE er det betraktelig enklere å redigere og kompilere kildekode. Med et IDE blir det enkelt å navigere rundt i kodebasen, f.eks. for å finne ut hvilke metoder som finnes på en klasse og hva de gjør. I tillegg har man selvfølgelig muligheten til å kjøre programmer og debugge de etc. Til alle disse IDE-verktøyene finnes det plugin-støtte for mange forskjellige OSS-biblioteker og applikasjoner

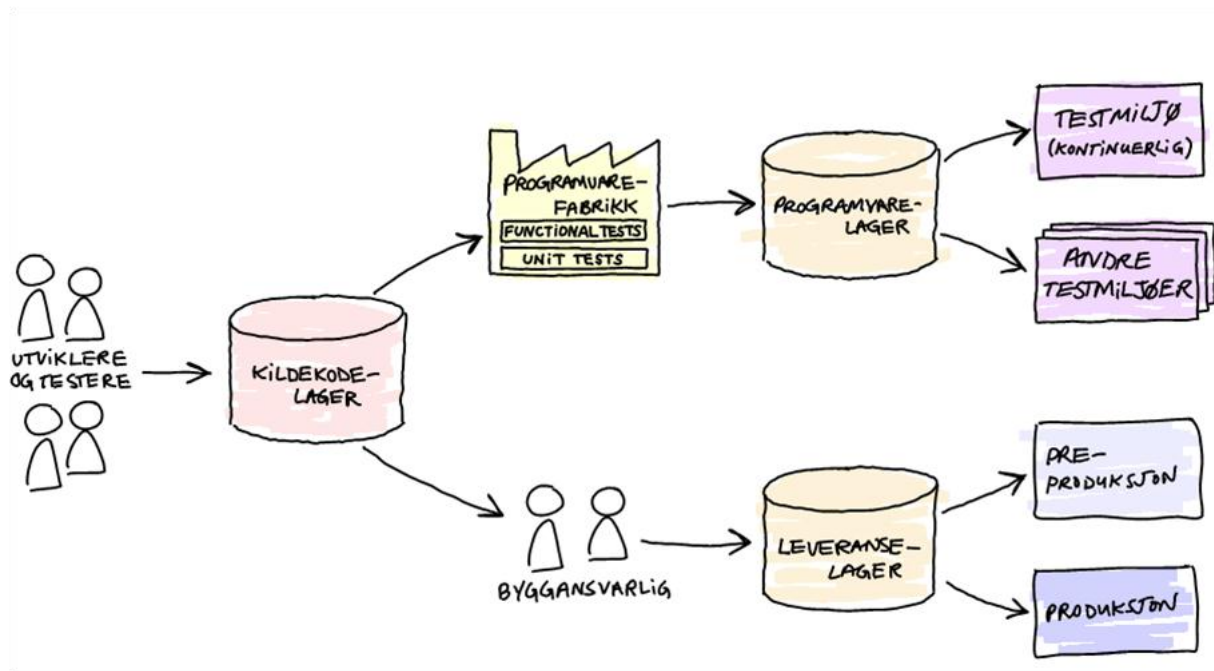
Vårt testmiljø er basert på åpen kildekode test-biblioteker som JUnit, HtmlUnit, JMeter og andre. For funksjonell testing brukes Fitness og Selenium, det siste for web-applikasjonstestning.

### **Bruk av Subversion for versjonskontroll**

Fra IDE-verktøyet kaller opp kompilatoren som oversetter kildekoden til objektkode (bytekode i Java). I tillegg interagerer det med versjonskontroll-systemet subversion for å sjekke ut og inn filer og finne riktig versjon av filer. Tidligere brukte vi CVS som versjonskontroll men i 2006 gikk vi over til subversion. Både CVS og subversion er OSS. Subversion-serveren har også et webgrensesnitt for enkel lesing, og kan enkelt nås av alle våre utviklere.

### **Automatisk daglige bygg fra kildekode**

I Nets sine utviklingsprosjekter er det alltid mange utviklere involvert. I tillegg til egen kode bruker vi mange ulike Java-biblioteker, tildels i ulike versjoner. Dette kan skape mange problemer i forbindelse med deployment. Vi er avhengig av at byggeprosessen er lik hver gang og at alle enhetstester og integrasjonstester er kjører feilfritt. Ved å bruke Hudson har vi kontroll på vårt eget bygg med hensyn til kildekode og tester. Hudson kan settes opp til å overvåke Subversion ved polling og skedulere automatisk bygging, og/eller kjøre bygging på faste tidspunkter. Ved hver bygging kjører man igjennom prosjektet egne enhetstester. Hudson har støtte for maven. Man kan derfor enkelt importere et komplett prosjekt ved oppgi en URLen til prosjekt-treet. Hudson laster leverer til slutt ferdige build-artifacts klare til deployment.



Figur 2: Vårt fabrikkanlegg<sup>1</sup>

### Uavhengig av tradisjonelle Java applikasjonsservere

Over tid har vi fått negativ erfaring med bruk av tradisjonelle Java EE-applikasjonsservere som for eksempel IBM WAS (WebSphere Application Server) og dens implementasjon av ESB (Enterprise Service Bus). Det er - eller har vært - nærmest umulig å utvikle JEE-applikasjoner uten at den får ulike knytninger og avhengigheter til den (kommersielle) JEE-serveren man har valgt å utvikle for. Dette skjer selv om spesifikasjonene er gratis og åpne: det er alltid huller i spesifikasjonene, og de enkelte leverandørene tilbyr gjerne "utvidelser" og "forbedringer" som det kan være fristende å benytte seg av. Vi har derfor i vår applikasjonsarkitektur unngått applikasjonstjenere med sine iboende avhengigheter. Vår applikasjonsarkitektur er derfor i dag hovedsaklig basert på åpne servlet-containerer som for eksempel Jetty og Tomcat. Ideen bak ESB passer ikke våre krav og vi har derfor utviklet vår egen meldningspumpe basert på Java og Oracle.

Støtter man for eksempel EJB-komponentmodellen med tilhørende kompliserte Java-klasser som er vanskelig eller nesten umulig å teste dette utenfor en kjørende EJB-container. Kjøring av tester blir plutselig en tung prosess. Det tar unødig tid å lage tester, og å kjøre dem. Dette bremser alle smidige prosjekter, og gjør utviklerne frustrerte. Dette med EJB-komponentenes unødige kompleksitet og deres problemer med testbarhet har så vidt jeg forstår vært hovedbegrunnelsen for at Spring ble utviklet. Mens EJB-komponentmodellen krever unødig kompliserte Java-klasser som i tillegg er vanskelige å teste, så kan man med Spring programmere server-komponenter (tjenester, entiteter) som helt vanlige java-klasser. Disse klassene blir i tillegg enkle å teste – fordi de ikke antar noe om sine omgivelser. Spring

<sup>1</sup> Tidligere definert av sjefsarkitekt Johannes Brodwall

leverer omgivelsene (database, meldingstjeneste, transaksjonstjeneste osv.) ”utenfra” via konfigurasjonsfiler eller annoteringer med såkalt dependency injection.

Siden vi ikke har behov for å bruke applikasjonsservere og dermed ikke distribuere programmene våre inn i et javaEE miljø kan vi pakke inn vår programvare med alle avhengigheter inkludert i en fil. Maven er konfigurert til å bygge opp en zip arkiv ferdig for deployment på våre applikasjonsservere. I deploy-filen ligger all konfigurasjon. Maven har i prinsippet ingen restriksjoner på hvilken IDE utviklerne ønsker å bruke.

Vi har på plass gode rutiner for utvikling og testing der også databaseskjema blir generert opp og testet ved hjelp av Maven, slik at man får en mer komplett releasestyring fra samme verktøy. Vi benytter Hibernate for persistering av data fra Java.

### **Databasearkitektur og verktøy**

Vi har en tredelt arkitektur som består av lagene web, applikasjon og database. I både web- og applikasjonslaget består de fleste basisservere av fiks ferdige OSS-komponenter. Unntaket er databaselaget hvor vi har standardisert på Solaris og Oracle som i hovedsak er lukket kildekode.

For optimalisering bruker vi lagrede prosedyrer i databasen med PL/SQL. Vi aksesser dataene direkte i databasen, via Java eller via Spring som kan kalle opp ferdige PL/SQL prosedyrer i databasen.

I noen prosjekter bruker vi Toad for å bygge, teste, og debugge PL/SQL prosedyrer, trigger, og funksjoner. Toad leveres både som OSS og kommersiell løsning. Med Toad kan programutviklere opprette og redigere databaseobjekter som tabeller, visninger, indekser, begrensninger og brukere. Toad SQL Editor gir deg en enkel og effektiv måte å skrive og teste prosedyrer og søk.

Tora (<http://tora.sourceforge.net/>) og SQL Nexus (<http://www.codeplex.com/sqlnexus>) er verktøy som hjelper våre utviklere å identifisere årsakene bak SQL-ytelsesproblemer. Tora er åpen kildekode og SQL Nexus er et kommersielt produkt. Begge verktøyene kan drastisk redusere tiden som man ellers måtte analysere data manuelt. Tora er definitivt blant de mest modne åpen kildekodeprogrammer. Tora er definert som en multiplattform for database management GUI som støtter tilgang til de fleste databaseplattformene som finnes i dag.

### **Test og feilrapportering**

Vi har et dynamisk utviklings- og test-miljø med ofte deployment med automatiske tester. Vi skriver tester til all kildekode for å sikre at koden vil virke hver gang vi gjør endringer. Vi kan også kjøre de fleste testene inne i IDE'et med rask eksekvering av tester. JUnit ([www.junit.org](http://www.junit.org)) er et mye brukt OSS-rammeverk for enhetstesting av Java-klasser. JUnit tilbyr enkel enhetstesting (unit testing) som i prinsippet er en systematisk testing av enkeltklasser eller grupper av tett koblede klasser. Vi lager egne TestCase-klasser (en helt vanlig Java-klasse som skrives etter spesielle regler) tilpasset JUnit-rammeverket. FitNesse benyttes for å teste funksjonalitet fra brukeren og forretningsiden. Ved hjelp av FitNesse kan

vi foreta funksjonelle tester og se om de dekker bruker- og forretningskrav. En funksjonell test er ikke nødvendigvis en ende-til-ende test. Vi har også i noen prosjekter brukt andre testverktøy som for eksempel TestNG som er mer slagkraftig testrammeverk, sterkt inspirert av JUnit.

I noen av våre prosjekter benytter vi oss av Test Driven Development (TDD). TDD er basert på en utviklingsteknikk med repetisjon av svært korte utviklingscykluser. Utvikleren skriver først enhetstesten, som vil feile, og deretter selve koden som skal få testen til å kjøre feilfritt. Ofte er også mange systemtester laget av testere som en del av kravspesifikasjonen. Vi ser at kvaliteten på koden blir høyere når utvikler tvinges til å forstå hva applikasjonen skal gjøre.

Vi bruker også Jmeter ([jakarta.apache.org/jmeter/](http://jakarta.apache.org/jmeter/)) til stresstesting av webgrensesnittet (ikke batchkjøringer). Med Jmeter (i noen prosjekter brukes også Jwebunit) kan vi undersøke stabiliteten til en applikasjon ved simulert høy last dvs. høy samtidighet (concurrency). Spørsmålet er: "Hvor mye last tåler applikasjonen før den knekker sammen?". For Nets er ytelsesutfordringen å starte med ytelsestesting tidlig i utviklingsfasen gjerne fra første iterasjon. Med Jmeter-verktøyet genererer vi en last for å verifisere at responstidskravene holder, at løsningen er stabil, og at det ikke er minnelekkasje eller andre forhold som bygger seg opp over tid. Slik kan også eventuelle flaskehalsen i systemet fjernes.

I noen prosjekter bruker vi avvikshåndteringssystemet Jira til å rapportere mangler eller feil etter testing. Disse blir deretter prioritert/estimert og sendt til riktig avdeling og eventuelt person. Med Jira ([www.atlassian.com/software/jira/](http://www.atlassian.com/software/jira/)). har vi muligheten til å prioritere, tildele, følge, rapportere og kvalitetssikre oppgaver/prosesser (programvarefeil og brukerstøtte henvendelser). Det er enkelt å tilpasse Jira til de enkelte oppgavene som vi har i Nets.

### **Bruk av OSS-komponenter i forretningskode**

I dag finnes det over 200.000 tilgjengelige gjenbrukbare OSS-komponenter i Java (en god del av dette er klassebiblioteker)<sup>2</sup>. Alle våre nye Java-prosjekter er avhengige av åpen kildekode, og mesteparten av all ny produksjonskode er OSS. Den største bidragsyteren her er åpne kildekodebiblioteker. En viktig faktor før vi velger åpne kildekodekomponenter eller programmer, er at produktet utvikles aktivt. Dette kan man ofte utlede av tilstanden på produktets hjemmeside. Før vi velger OSS-kode sjekker vi om produktet har en aktiv utviklingsbase. Er det et aktivt miljø rundt produktet? Er produktet støttet av et selskap? Det er også viktig å avklare om produktet er støttet av OSS-organisasjoner som for eksempel Apache eller av et programvarehus.

Vi velger kun produkter som har et minimum av aktivitet. Når man skal vurdere nye åpne kildekode-produkt/biblioteket har vi alltid risikoen for at samfunnet bak produktet vil dø, og dermed forsvinner også muligheten for at andre tar ansvar for bugfixer og utvikling av nye funksjoner. Mange prosjekter i dag er støttet av store selskaper som tilbyr støtte. Andre

---

<sup>2</sup> Verdikt, NTNU, 16 mars 2009, v2.1 (<http://www.idi.ntnu.no/emner/ttd10/curricula/P1-7-conradi09.pdf>)

prosjekter er vurdert så populære og har så stor bruker- og utviklerbase at risikoen anses som liten. Når du velger mindre prosjekter uten selskapsstøtte eller stor brukerbase må man akseptere risikoen for å ta ansvar for å fikse feil og legge til nye funksjoner selv.

Når punktene over er tilfredstilt står vi igjen med lisensmodellen. Lisensmodeller som har spesifikke gjensidige forhold brukes vanligvis ikke av Nets. Mindre produkter med APL (APL=Adaptive Public License, en Open Source lisens fra Universitet i Victoria), MIT (MIT License er en free software lisens som kommer fra Massachusetts Institute of Technology), og BSD-lisenser kan brukes av de enkelte prosjektene etter skjønn. Stort sett det meste av programvare som følger disse lisensene krever ingen juridisk vurdering fra Nets, eksempler på populær programvare som bruker disse lisensmodeller er Spring-rammeverket og Apache webserver. Når det gjelder produkter som støtter GPL og LGPL brukes i disse stort sett i serverløsninger (Linux-kernel, Linux/Unix-serverprosesser, operativsystem/nettverkssystem programmer) og ikke i Nets-kildekode siden disse lisensene ikke krever gjensidighet. De fleste løsningene basert på GPL og LGPL kjøres det en formell evaluering i Nets.

Under har jeg listet opp en enkel sjekklister:

- En enkelt pakke kan inneholde kode under forskjellige lisenser. Sjekk alle lisens variantene.
- Hva innebærer lisensen for deg og din organisasjon
- Unngå all programvare som er tvetydige, uklare, eller kompliserte lisenser.
- Skal du eller din bedrift utgi programvare under en fri / åpen lisens, bruk en av de kjente og etablerte lisensene ikke lag en ny lisensmodell

I vår anskaffelsesprosess i forbindelse med å ta i bruk Open Source biblioteker, står det at all programkode skal gjennomleses, og at større etablerte mye brukte biblioteker kan godkjennes uten gjennomlesing.

Når det velges eksterne OSS-klassebiblioteker ser vi at det i mange tilfeller blir bedre kodekvalitet i tillegg reduserer dette også mengden av spesiell kunnskap og nødvendig dokumentasjon internt.

Noen enheter i vår organisasjon har valgt komplette åpen kildekodeapplikasjoner. Produksjonsmiljøene våre har valgt Linux-operativsystemet med standard serverprosesser som samba, nfs, smtp, sendmail etc. Vi har også valgt OSS-produkter høyere opp i næringskjeden som for eksempel osCommerce løsningen. osCommerce er en nettbutikk e-handelsløsning som er gratis tilgjengelig under GNU General Public License. osCommerce gjør det enkelt å sette opp, drifte og vedlikeholde nettbutikker. Vi har integrert løsningen til vårt eget system Netaxept. OsCommerce er et eksempel på et komplett åpen kildekode-program som dekker et forretningsmessig behov og er ikke bare et Java bibliotek eller verktøy.

## **Nets-ansatte gir tilbake til OSS-miljøet på sin fritid**

Nets vurderer kontinuerlig hvilke egne Java-programmer som det kan egne seg for videreutvikling som åpne kildekodeprosjekter. Ved å gi tilbake til fellesskapet bidrar Nets mer til gjenbruk og får bedre og mer robust kode. I tillegg får man en større utviklerbase (mennesker og kompetanse utenfor Nets) og en markedsføringskanal inn mot Nets.

All kode som ikke er forretningsspesifikk, eller på annen måte gir Nets en stor konkurransemessig fordel, vurderer vi å legge ut som Open Source. I utgangspunktet skal man bruke Apache sin lisensmodell (<http://www.apache.org/licenses/>). Nets ansatte kan i prinsippet selv velge hvor de ønsker å legge ut kildekoden. For å oppnå stor utbredelse kan man benytte github, bitbucket, sourceforge etc.

Internt utvikler vi svært lite domenespesifikk kode som vi deler internt. Grovt sett er dette fortrinnsvis infrastrukturkode og utility biblioteker. Eksempler på dette er shared og common bibliotekene. Vi har en del utfordringer med denne koden da vi har mange avhengigheter og mange forskjellige ønsker om funksjonalitet samtidig som vi har et distribuert og uklart eierskapsforhold til koden. Ved å legge koden ut som Open Source i en tidlig utviklingsfase øker vi kvaliteten på koden, sikrer høyere grad av gjenbruk, og reduserer fristelsen for å legge inn for mye funksjonalitet og for mange avhengigheter.

Fra Gartner Groups prognoser for 2012 anslås det at over 90 % av alle nye applikasjoner vil benytte seg av OSS. I Nets benytter alle Java-prosjektene både interne og eksterne OSS-klasse-biblioteker.

## En Nets forretningsavdelings bruk av Open Source

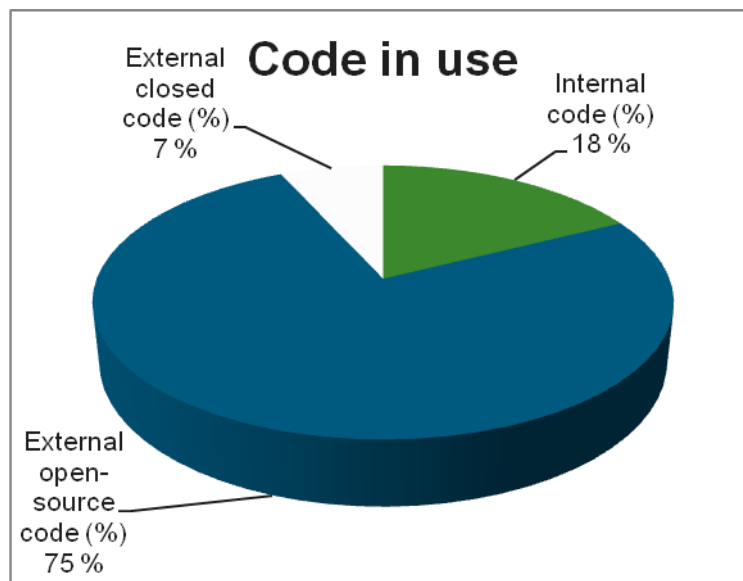
Vår analyse har tatt utgangspunkt i 14 større utviklingsprosjekter. Fra disse prosjektene har vi sett på bruk av OSS i forbindelse med biblioteker, verktøy og applikasjoner. Under har jeg listet opp deler av resultatet:

1. **Klassebiblioteker:** Biblioteker er kode som inkluderes i Nets kode og produksjonssettes som en del av denne. Et eksempel er log4j. Resultatet av vår analyse viser at 76 % av all forretningskode i disse prosjektene er basert på OSS.
2. **Servertjenester:** Over 95 % av alle servertjenester som brukes på dagens applikasjonsservere som for eksempel Linux-kernel prosesser, apache httpd, smtp, nfs etc er basert på OSS.
3. **Utviklingsverktøy:** Nesten alle **verktøy** som utviklere og tester benytter i dag er OSS. Eksempler på Open Source verktøy er Eclipse, NetBeans, JUnit, Fitness, FireFox.

## Konklusjon

Nesten alle komponentene som brukes i vårt utviklingsrammeverk er OSS. Disse komponentene fungerer meget bra både i vårt utviklingsrammeverk, testmiljøer og oppgave/feilrapporteringsrutiner.

Vi ser at forum og diskusjonsgrupper gir mye tilbake av både hjelp og ideer til videreutvikling. Vi ser det som en stor fordel å benytte OSS i stedet for å kjøpe dyre komponenter/program eller utvikle funksjonalitet selv. OSS løsninger utvikles og testes av mange utviklere og brukere. Dette gir ofte løsninger av meget god kvalitet.



**Figur 2: Fordeling av egen kode, åpne kode og lukket kode**

**Fra vår interne rapport basert på 14 større utviklingsprosjekter ser vi at ca 76 % av vår kode er åpen kildekode<sup>3</sup>, mindre enn 7 % er lukket ekstern kode, og 17 % er egenutviklet.** Den eksterne koden er redusert dag for dag etterhvert som vi blant annet konverterer vår kode fra WAS (Web Application Server) til Jetty.

Forskjellene i bruk av intern kode, åpen kildekode, lukket kildekode gir det relative bidrag fra Nets sin utviklingsavdeling i det bestemte prosjektet. Vi har ikke målt størrelsen på individuelle bidrag. Det er mange måter å måle størrelsen på de forskjellige komponentene. Selv om vi importerer et bibliotek, betyr ikke det at vi bruker mye av det eller at det brukes i det hele tatt. I vår analyse har vi satt bruken lik både for eksterne og interne biblioteker, men vår gjetning er at vi bruker mer av de interne bibliotekene enn de eksterne siden de interne bibliotekene er utviklet spesielt for Nets. En kjøring som analyserer vår "run time" kode vil gi en mer eksakt konklusjon men er ikke en del omfanget av denne rapporten.

Vi har heller ikke tatt med avhengighetene til operativsystemet, forskjellige Linux kjernekomponenter, tilleggspakker og GNU-verktøyene og selve Java-plattformen. Hadde vi tatt med disse komponentene vil dette økt bruken av ekstern åpen kildekode antallet betraktelig.

Suksessen med åpen kildekode i Nets antas å være et resultat av vår interne høye kompetanse i Java og åpne kildekode produkter. Kontinuerlig investering i denne kompetansen er en suksessfaktor for fremtidig Open Source strategi. På prosjekt-siden vi har gått fra en fossefallsmodell for utvikling til smidige metoder og Scrum.

En nøkkelkomponent i vår utviklingsprosess er Maven. Maven ble valgt tidlig og gir oss kontroll over prosjektenes eksterne avhengigheter og gir oss frihet når det gjelder IDE, testrammeverk, byggeservere, miljø og distribusjonsstrategi.

Fra vårt statistiske materiale vil du finne at de fleste av våre java prosjekter er basert på:

- ✓ Linux web og applikasjoner servere har blitt etablert som en standard
- ✓ Ulike GNU-verktøy (utvikling, distribusjon og produksjon)
- ✓ Maven som byggeverktøy og prosjektbeskrivelsesverktøy er etablert som en standard
- ✓ Eclipse som IDE, men de ulike personlige preferanser og den raske endringen av IDE gjør det ikke mulig å ha noen offisiell Nets standard
- ✓ Subversion versjonskontroll er etablert som en standard
- ✓ Hudson som byggeserver har blitt etablert som en standard
- ✓ Utviklingsmiljøet er basert på Java (JDK)
- ✓ Kjøring av enhetstester og funksjonelle tester er basert på JUnit og Fitness

Vi har etablert en beste praksis for innhenting og bruk av åpen kildekodebiblioteker som er beskrevet tidligere i dokumentet. Det viktigste er hvilken type lisens brukes for OSS-produktet. Det er også viktig å sjekke populariteten til produktet og hvilken støtte produktet

---

<sup>3</sup> Tallene representerer kun antall kodelinjer med eksterne Open Source biblioteker og ikke den reelle andelen bruk av disse bibliotekene (kan kun ses i runtime). I tallene er det heller ikke tatt bruk av Open Source operativsystemet Linux med tilhørende TCP/IP-stakk.

har fra bedrifter/organisasjoner. Dette er innspill til for å kunne vurdere risikoen for at et produkt ikke lenger vil bli vedlikeholdt. OSS-produkter som er implementert i Nets som ikke lenger blir vedlikeholdt av fellesskapet, må Nets være forberedt på å fortsette utviklingen med hensyn til både ny funksjonalitet og feilrettinger.

*Kommentar: En takk til Bjørn Nordlund og Truls Thirud for faglig bidrag og gjennomlesning av artikkelen (basert på et ekstrakt av en intern rapport skrevet på engelsk)..*