

# Prosjektrapport

## **Informasjonssikkerhet og risiko ved bruk av fri programvare eller åpen kildekode i IKT-løsninger for helsesektoren**

En utredning

Eva Henriksen  
Eva Skipenes



**Tittel:**                   **Informasjonssikkerhet og risiko ved bruk av fri programvare eller åpen kildekode i IKT-løsninger for helsesektoren**

NST-rapport:           08-2008

Prosjektleder:       Eva Henriksen

Forfattere:            Eva Henriksen og Eva Skipenes

ISBN:                   978-82-8242-000-6

Dato:                   23.10.2008

Antall sider:         79 (inkl. vedlegg)

Emneord:              Åpen kildekode. Fri programvare. Informasjonssikkerhet. Applikasjoner for helsesektoren. Lisenser.

Oppsummering:        Problemstillingen for rapporten er om bruk av åpen kildekode og fri programvare i helsesektoren kan være en trussel mot informasjonssikkerheten. Rapporten er basert på: 1) Litteraturstudium mhp ulike lisenstyper og sikkerhetsrelaterte problemstillinger knyttet til åpen kildekode / fri programvare. 2) Intervju av et utvalg aktører innen det norske helsevesen, både leverandører og driftsmiljø, samt forskere i noen pilotprosjekt knyttet til NST. På bakgrunn av dette har vi gjort en oppsummering av sikkerhetsmessige aspekter og trusler. Generelt sett ser vi ingen tydelige sikkerhetsmessige forskjeller på å bruke proprietær programvare kontra åpen kildekode. For programvare som har en begrenset målgruppe, slik tilfellet ofte er i helsesektoren, kan det imidlertid være usikkert i hvilken grad åpen kildekode vil bli gjenstand for nødvendig kvalitetssikring. Fagapplikasjoner i helsesektoren er underlagt strenge krav til funksjonalitet og sikkerhet. Dersom slike fagapplikasjoner skal utvikles som åpen kildekode, vil det derfor være avgjørende at det står en organisasjon bak som vurderer kvaliteten og om nødvendig funksjonalitet er implementert.

Utgiver:                Nasjonalt senter for telemedisin  
Universitetssykehuset Nord-Norge  
Postboks 35  
9038 Tromsø  
Telefon: 77 75 40 00  
E-post: [info@telemet.no](mailto:info@telemet.no)  
Internett: [www.telemet.no](http://www.telemet.no)

Det kan fritt kopieres fra denne rapporten hvis kilden oppgis. Brukeren oppfordres til å oppgi rapportens navn, nummer, samt at den er utgitt av Nasjonalt senter for telemedisin og at rapporten i sin helhet er tilgjengelig på [www.telemet.no](http://www.telemet.no).

## English summary

**Title:** Information security aspects related to use of open source or free software in ICT systems for healthcare

**Abstract:** The problem addressed in this report is whether use of open source or free software in the healthcare sector is a threat to information security. The report builds on 1) a literature study with respect to licenses for open source or free software and information security aspects related to the use of open source or free software; and 2) interviews with a wide set of actors within the Norwegian healthcare system, both commercial application suppliers, hospital IT departments, network and service providers, and research and development projects at the Norwegian Centre for Telemedicine (NST). Based on this input we have summarised security related issues and threats.

In general, we see no clear security differences in using proprietary software or open source software. The main issues are whether security is emphasised during system development and whether information security management is a focus in the operational environment of the systems.

For software with a limited user group, like the healthcare sector, a question can be raised whether open source software will be subject to a necessary amount of quality assurance. Professional software applications and systems for the healthcare sector are subject to strict requirements regarding functionality and security. In the development of such systems it is necessary to make sure that the system makes it possible for the user to obey to legal regulations and rules. If these professional systems should be developed as open source software, it would be a benefit to have an organisation assuring the quality and the necessary functionality.

# Forord

Regjeringen mener økt bruk av fri programvare legger grunnlaget for en positiv utvikling av gode IKT-løsninger i det offentlige og i samfunnet ellers, bl.a. fordi:

- Bruk av fri programvare forenkler deling av løsninger på tvers i offentlig sektor.
- Økt bruk av fri programvare gir grunnlag for økt IKT-kompetanse i offentlig sektor.
- Utvikling av en delingskultur i offentlig sektor vil kunne øke den totale utviklingen av gode løsninger.

Høsten 2007 utlyste derfor Fornyings- og administrasjonsdepartementet (FAD) og Nasjonalt kompetansesenter for fri programvare prosjektmidler til prosjekt basert på eller innrettet mot fri programvare. Utlysningen var på totalt 5 mill NOK. Prosjektene skulle startes innen utgangen av januar 2008 og avsluttes innen 1.november 2008. Prosjektene skulle støttes med inntil 50 % av kostnadene, oppad begrenset til 1 mill NOK pr prosjekt.

Dette prosjektet er et resultat av innvilgede midler fra denne utlysningen sammen med interne midler fra NST.

Målgruppen for utredningen er beslutningstakere, ledere og IT-ledere i helsesektoren, leverandører av IT-system til sektoren, prosjektledere, forskere, utviklere og andre med interesse for åpen kildekode og fri programvare.

Tromsø, 23. oktober 2008

Eva Henriksen og Eva Skipenes



# Innhold

1	Innledning .....	7
1.1	Vår motivasjon .....	7
1.1.1	Aspekt ved informasjonssikkerhet .....	8
1.1.2	Juridisk grunnlag for informasjonssikkerhet .....	8
1.2	Metode .....	9
1.3	Rapportens oppbygning.....	10
2	Bakgrunn: Om fri programvare og åpen kildekode.....	11
2.1	Begrepsavklaring .....	11
2.1.1	Historikk: Åpen kildekode og/eller Fri programvare?.....	12
2.1.2	Mange forkortelser .....	16
2.2	Lisenstyper.....	16
2.2.1	GPL.....	17
2.2.2	BSD.....	18
2.2.3	Andre lisenser .....	20
2.3	Bruksaspekt .....	20
2.4	Juridiske og økonomiske aspekt.....	22
2.4.1	Utvikling, oppgradering, vedlikehold og support.....	22
2.4.2	Juridiske problemstillinger .....	24
2.4.3	Økonomiske og forretningsmessige spørsmål .....	24
3	Åpen kildekode og informasjonssikkerhet .....	27
3.1	Sammenhengen mellom kvalitet og sikkerhet i programvare.....	28
3.2	Argument for og mot åpen kildekode.....	29
3.2.1	Egen evaluering av sikkerheten.....	29
3.2.2	Raskere feilretting .....	29
3.2.3	Åpen kildekode gir liten garanti for systematisk gjennomgang av kode.....	30
3.2.4	Eksponering av sårbarheter.....	31
3.2.5	Planting av ondsinnet kode.....	32
3.2.6	Mulighet for lokale tilpasninger .....	33
3.3	Sammenligning av sikkerhet og pålitelighet i lukket kontra åpen kildekode.....	33
3.4	Oppsummering .....	34
4	Åpen kildekode i helsesektoren .....	35
4.1	Åpen kildekode i helsesektoren – internasjonalt .....	35
4.1.1	Noen internasjonale prosjekt og løsninger .....	35
4.1.2	Hvorfor åpen kildekode ved Beaumont hospital.....	36
4.1.3	Hvorfor ikke åpen kildekode i helsesektoren i Quebec .....	38
4.2	Åpen kildekode i helsesektoren i Norge .....	38
4.2.1	Utbredelse av åpen kildekode i helsesektoren .....	39
4.2.2	Fordeler og utfordringer ved bruk av åpen kildekode .....	40
4.2.3	Sikkerhetsmessige aspekt .....	41
4.2.4	Har helsesektoren spesielle utfordringer mht. programvare? .....	42
4.2.5	Bruk av åpen kildekode og åpne standarder .....	42
4.2.6	Aktørenes bidrag til utvikling av åpen kildekode.....	42
5	Trusler og mottiltak .....	44
6	Konklusjoner og anbefalinger .....	47

Forkortelser .....	51
Ordlister .....	52
Litteraturliste .....	53
Vedlegg A: Tabell over sikkerhetstrusler og -fordeler .....	56
Vedlegg B: Intervju .....	63
B.1 Intervjuguide .....	63
B.2 Hvem som ble intervjuet .....	63
B.3 Sammendrag av synspunkt i intervjuene .....	66
B.3.1 Utbredelse av åpen kildekode i helsevesenet .....	66
B.3.2 Support av åpen kildekode .....	69
B.3.3 Fordeler med å bruke åpen kildekode .....	69
B.3.4 utfordringer med bruk av åpen kildekode .....	70
B.3.5 Sikkerhetsmessige aspekt .....	72
B.3.6 Har helsesektoren spesielle utfordringer mht. programvare? .....	73
B.3.7 Bruk av åpen kildekode og åpne standarder .....	74
B.3.8 Aktørenes bidrag til utvikling av åpen kildekode .....	75
Vedlegg C: Opplisting av noen åpen kildekode-prosjekt innen helsesektoren .....	77

# 1 Innledning

Regjeringen i Norge, som i mange andre land, mener økt bruk av fri programvare legger grunnlaget for en positiv utvikling av gode IKT-løsninger i det offentlige og i samfunnet ellers. Argumentene er bl.a. at bruk av fri programvare forenkler deling av løsninger på tvers i offentlig sektor, gir grunnlag for økt IKT-kompetanse og en delingskultur som vil kunne øke den totale utviklingen av gode løsninger [1].

Undersøkelser gjennomført av Gartner Group<sup>1</sup> i 2006 og 2007 viser en sterk økning i bruken av åpen kildekode i offentlig sektor, både når det gjelder operativsystem og applikasjoner på klientsiden (f.eks. kontorstøtte). Tendensen er den samme både i Europa og Nord-Amerika. Blant de viktigste årsakene til denne utviklingen er forventningen om reduksjon av kostnader og forenkling i anskaffelsesprosesser. Langt nede på listen kommer begrunnelser som politisk press og støtte til lokalt næringsliv.

I helsesektoren vil problemstillinger knyttet til sikkerhet ved bruk av åpen kildekode og fri programvare være avgjørende for om sektoren vil velge å ta i bruk slik programvare.

## 1.1 Vår motivasjon

Den gjengse oppfatningen er at bruk av fri programvare er en styrke for sikkerhet, tillit og pålitelighet til IKT-system. Mange mener at åpenhet gir bedre kvalitet på programvaren og rettinger gjøres raskere. Åpenhet er et hinder for "security by obscurity".

I forbindelse med risikovurdering av informasjonssikkerheten i IKT-system for helsesektoren har det imidlertid dukket opp spørsmål om bruk av åpen kildekode og fri programvare i seg selv kan være en trussel mot informasjonssikkerheten. Er det virkelig slik at fri programvare er sikrere enn annen programvare? Eller:

- Kan åpenheten gjøre det enklere for hackere å lage ondsinnet programvare som gir seg ut for å være helsevesenets system?
- Hvordan kontrollerer man at vedlikehold av fri programvare gjøres på en sikker måte:
  - at nye versjoner eller nye programvaremoduler ikke inneholder ondsinnet programvare som bevisst er plantet der
  - at kvaliteten er så god at programvaren ikke utgjør en sikkerhetsrisiko
  - at nødvendig support er tilgjengelig over tid
- Hvem tar ansvar for at programvaren gjør det mulig å oppfylle lovpålagte sikkerhetskrav?

Målet med denne utredningen er å forsøke å gi noen svar på disse spørsmålene.

Nasjonalt senter for telemedisin (NST) har brukt og bruker åpen kildekode i flere av sine pilotprosjekt. Vi har imidlertid hatt liten eller ingen oversikt over hvor utbredt slik programvare er i helsesektoren. Vi har derfor også gjort undersøkelser omkring helsesektorens bruk av fri programvare.

Vi har en bred målgruppe for denne utredningen: beslutningstakere, ledere og IT-ledere i helsesektoren, leverandører av IT-system til sektoren, prosjektledere, forskere, utviklere og andre med interesse for åpen kildekode og fri programvare.

---

<sup>1</sup> Peter Hidas i Computerworld 08.02.08

### 1.1.1 Aspekt ved informasjonssikkerhet

Informasjonssikkerhet er svært viktig ved bruk av IKT i helsesektoren. Informasjonssikkerhet omfatter begrepene *konfidensialitet*, *integritet*, *tilgjengelighet* og for helsesektoren også *kvalitet*. Konfidensialitetsaspektet er viktig å oppfylle for å ivareta taushetsplikten. Integritet er viktig for å sikre at opplysninger ikke endres på en urettmessig måte, med mulig feilbehandling som resultat. Tilgjengelighetsaspektet skal sikre at pasientopplysninger er tilgjengelig for de som trenger det når de trenger det, f.eks. i forbindelse med behandling av pasienten. For helseopplysninger er det også viktig at kvaliteten på dataene som registreres er god.

Det finnes mange måter å definere sikkerhetsaspektene på. Her er en sammenfatning:

- **Konfidensialitet** er av ISO [28] definert som den egenskapen at informasjon ikke gjøres tilgjengelig for eller avsløres til uautoriserte personer, instanser eller prosesser. Det innebærer en forsikring om at informasjon bare deles mellom autoriserte personer eller organisasjoner, at tilgang begrenses til "de rette folk" og holder "de gale folk" utenfor. Informasjonen er beskyttet slik at bare autoriserte personer kan se eller tilegne seg informasjonen.
- **Integritet** er relatert til informasjonens troverdighet, at man kan ha tiltro til informasjonen og stole på den, at informasjonen er korrekt og ikke er blitt forfalsket eller ødelagt, at den er autentisk og fullstendig. Integritet er den egenskapen at data ikke med overlegg er blitt klusset med eller er blitt endret ved et uhell. Integritet innebærer å trygge informasjonens nøyaktighet og fullstendighet [28], og beskytte mot uautorisert og feilaktig endring. Integritetsbrudd kan gi to ulike scenarier: 1) Data er endret, men gir fortsatt mening. 2) Data er endret slik at det ikke lenger gir mening, informasjonen kan ikke forstås. For helsesektoren kan det første scenariet være mer alvorlig enn det andre, fordi data kan gi grunnlag for feilbehandling av pasienten. Det andre scenariet vil resultere i manglende eller utilgjengelig informasjon.
- **Tilgjengelighet** er den egenskapen at informasjonen kan nås og er nyttbar når en autorisert person skal ha tilgang til den [28]. Informasjonen må forefinnes der det er bruk for den og når den trenges.
- **Kvalitet** henspiller på at informasjonen er korrekt og ikke misvisende. Kvalitet kan synes å være overlappende med integritet, og disse blir ofte slått sammen i en analyse av sikkerhetstrusler og risiko. Men spesielt i system og tjenester som innbefatter bilder, lyd og video er det nyttig å se på disse to aspektene hver for seg.

### 1.1.2 Juridisk grunnlag for informasjonssikkerhet

Personopplysningsloven § 13 [29] og helseregisterloven § 16 [31] stiller omtrent likelydende krav til informasjonssikkerhet ved behandling av person-/helseopplysninger: *"Den data-behandlingsansvarlige og databehandleren skal gjennom planlagte og systematiske tiltak sørge for tilfredsstillende informasjonssikkerhet med hensyn til konfidensialitet, integritet, kvalitet og tilgjengelighet ved behandling av helseopplysninger."* (sisert fra helseregisterloven).

Helsepersonelloven § 21 [32], Hovedregel om taushetsplikt, krever videre: *"Helsepersonell skal hindre at andre får adgang eller kjennskap til opplysninger om folks legems- eller sykdomsforhold eller andre personlige forhold som de får vite om i egenskap av å være helsepersonell."*

Dette kravet om taushetsplikt stammer fra Hippokrates, legevitenskapens "far", som levde for 2400 år siden. Den Hippokratiske ed sier om taushetsplikten<sup>2</sup>: *Alt som kommer til min viten under utøvingen av mitt yrke eller i daglig samkvem med mennesker, som ikke burde bli kjent for andre, vil jeg holde hemmelig og aldri avsløre.*

I forskrift til personopplysningsloven [30] (som også gjelder som sikkerhetsforskrift for helseregisterloven) står det i § 2-11 Sikring av konfidensialitet: *"Det skal treffes tiltak mot uautorisert innsyn i personopplysninger hvor konfidensialitet er nødvendig. Sikkerhets-tiltakene skal også hindre uautorisert innsyn i annen informasjon med betydning for informasjonssikkerheten."*

## 1.2 Metode

Arbeidet med denne rapporten har vært tredelt:

1. Litteraturstudium for å få oversikt over ulike lisenstyper relatert til åpen kildekode, og for å få en introduksjon til sikkerhetsrelaterte problemstillinger knyttet til program som er gitt ut som åpen kildekode eller fri programvare.
2. Intervju av et utvalg aktører innen det norske helsevesen, både på leverandørsiden og på bruker-/driftssiden, for å få innblikk i holdninger til og omfang av åpen kildekode i sektoren. I tillegg har vi intervjuet forskere i noen pilotprosjekt knyttet til NST. Et par eksempler på holdninger til innføring av åpen kildekode innen helsesektoren utenfor Norge har vi også tatt med.
3. Basert på litteraturstudiene og intervjuene har vi laget en oppsummering av sikkerhetsmessige problemstillinger og trusler som har blitt nevnt. Vi har ikke sett det som hensiktsmessig å gjennomføre en fullstendig risikovurdering av de kartlagte truslene, fordi det ikke er mulig å si noe spesifikt om sannsynlighet for truslene uten å se dem i sammenheng med omgivelsene en gitt programvare skal anvendes innenfor, og de tilhørende organisatoriske rutiner og prosedyrer.

Avslutningsvis trekker vi noen konklusjoner og gir noen anbefalinger.

Grunnleggende system og verktøy basert på åpen kildekode, som er og lenge har vært i utbredt bruk, anses som lite kontroversielle og har ikke vært hovedfokus i vårt arbeid. Det gjelder bl.a.:

- *Operativsystem: f.eks. Linux, og ulike andre varianter av Unix*
- *Kontorstøttesystem: f.eks. OpenOffice*
- *Nettlesere: f.eks. Mozilla/Firefox*
- *Web-server: f.eks. Apache*

Vi er mer opptatt av:

1. Åpent tilgjengelige programvaremoduler som tas i bruk eller inkorporeres i nyutviklet programvare – hvor trygge/sikre er disse, hvem har ansvar for testing, vedlikehold, support, etc.
2. Fullstendige (fag)applikasjoner som er lagt ut som fri programvare – er det en sikkerhetsrisiko at hvem som helst kan inspisere koden og se hvilke løsninger som er valgt? Hva om de svakhetene som finnes blir utnyttet på en ondsinnet måte? Og er det så sikkert at noen som helst vil gjennomgå koden med hensyn på rettinger/

---

<sup>2</sup> [http://no.wikipedia.org/wiki/Hippokratiske\\_ed](http://no.wikipedia.org/wiki/Hippokratiske_ed)

forbedringer og videreutvikling? Hva skjer når de som i sin tid utviklet dette, mister interessen eller er gått konkurs?

### 1.3 Rapportens oppbygning

Kapittel 2 er et bakgrunnskapittel med fakta om åpen kildekode og fri programvare. Disse begrepene brukes til dels om hverandre. Kapitlet starter med definisjoner og historikk, beskriver lisenstyper, bruksområder for fri programvare, ansvarsforhold omkring oppdatering og vedlikehold av fri programvare, juridiske aspekt knyttet til lisensieringen og noen forretningsmessige problemstillinger og modeller.

Kapittel 3 tar spesielt opp problematikken omkring *informasjonssikkerhet* og åpen kildekode.

I kapittel 4 gir vi en oversikt over utbredelse/bruk av åpen kildekode/fri programvare i *helse-sektoren*, både internasjonalt og nasjonalt. Den nasjonale delen er basert på intervju med ulike aktører i sektoren.

Kapittel 5 inneholder en gjennomgang av de sikkerhetstruslene som har blitt omtalt i de foregående kapitlene.

Kapittel 6 gir noen avsluttende kommentarer og anbefalinger.

Vedlegg A har en detaljert oversikt over trusler ved og fordeler med bruk av åpen kildekode, hentet fra litteraturen og intervjuene.

Vedlegg B inneholder en oversikt over hvem vi intervjuet, hvilke spørsmål vi baserte intervjuene på og en sammenstilling av synspunktene som kom fram i intervjuene.

Vedlegg C gir en oversikt over en del programvare for helsesektoren, som er gitt ut som åpen kildekode.

## 2 Bakgrunn: Om fri programvare og åpen kildekode

Man sier gjerne at i utgangspunktet var *all* programvare fri eller åpen. Programvare var i første omgang verktøy og metode for forskere ved universitetene, og metoder forskere bruker skal være offentlig tilgjengelige og reproduserbare.

På den annen side koster det tid og penger å utvikle programvare, og man så snart at disse kostnadene kunne dekkes inn ved å selge programvaren. Det kan ligge stor fortjeneste i det å masseprodusere programvare for salg. Dermed ble koden for mange en forretningshemmelighet som man måtte beskytte, lukke, ikke gi fra seg. Programvare ble sett på som åndsverk og som noe det kunne tas patent på.

I denne rapporten velger vi i hovedsak å bruke betegnelsen ÅPEN KILDEKODE. Dette er det argumentert for i avsnitt 2.1.1. Men der det er viktig for sammenhengen vil vi fortsatt bruke betegnelsen fri programvare.

### 2.1 Begrepsavklaring

Det er en viss forvirring ute og går når det gjelder begrepene åpen kildekode og fri programvare. Hovedhensikten med dette avsnittet er å bidra til en begrepsavklaring. Historien bak disse begrepene har betydning, og vi gir derfor en kort gjennomgang av denne. For å forstå problemstillingene av forretningsmessig og juridisk art som er knyttet til åpen kildekode, så må man, slik Michael Overly [4] sier, ha kunnskap om forhistorien til bevegelsen(e), hvordan det hele startet og hvordan det har utviklet seg de siste 15-20 årene.

Den største misforståelse er at "fri programvare" er det samme som "gratis programvare". Richard Stallman, grunnleggeren av Free Software Foundation (FSF), poengterer hva som ligger i begrepet "fri programvare" ved å si at programvaren er: "*Free as in 'free speech', not as in 'free beer'*" [9]. Andre sier om programvaren at den er "*Free as in 'freedom'*". På den annen side er det mange proprietære produkt som kan lastes ned gratis: "Gratis" er i så tilfelle en del av markedsføringen og ikke et prinsipp for utviklingen av slik programvare. Et eksempel på slik markedsføring er Adobe som gir Acrobat Reader gratis til alle.

Men åpen kildekode og fri programvare er ofte gratis eller svært billig. Det er et resultat av den utviklingsmodellen og -filosofien som ligger bak: Mange bidrar av idealisme og egen interesse og venter å få like mye nyttig tilbake når de trenger det. Denne tankegangen og utviklingsfilosofien er beskrevet av Eric S. Raymond i boka "The Cathedral and the Bazaar" [6]. Utviklingen kan ses på som et spleiselag mellom mange aktører. Når programvaren først er utviklet, koster det svært lite å mangfoldiggjøre den.

Før vi definerer de ulike betegnelsene for åpen kildekode og fri programvare, vil vi si noe om hva det *ikke* er.

Kildekode som er åpen men ikke kan modifiseres, bare sees, kan *ikke* regnes som åpen kildekode eller fri programvare<sup>3</sup>.

Det motsatte av åpen kildekode og fri programvare er lukket eller proprietær programvare. Slik programvare er i utgangspunktet forbudt å gjenbruke, modifisere og distribuere uten at man har spesiell tillatelse til det, og det kan være lagt inn tekniske begrensninger som hindrer at dette lar seg gjøre.<sup>4</sup> Begrepet "proprietær" brukes også om programvare som ikke

<sup>3</sup> <http://opensource.org/docs/osd> og <http://www.gnu.org/philosophy/free-sw.html>

<sup>4</sup> <http://www.gnu.org/philosophy/categories.html#ProprietarySoftware>

er basert på standarder. Men litteraturen bruker i stor utstrekning "proprietær" som en motsats til fri/åpen programvare, og vi har også valgt denne betydningen av ordet.

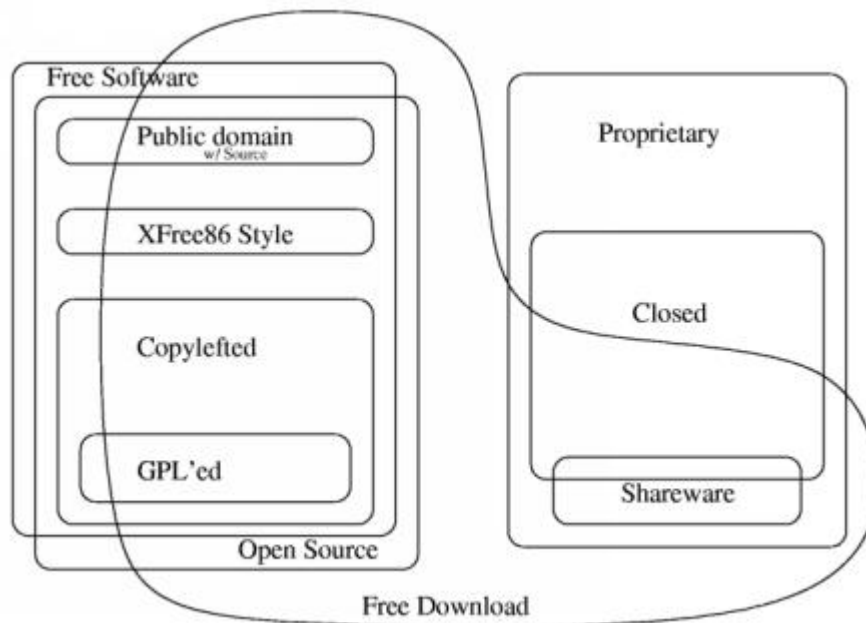
Blant andre ofte brukte betegnelser og begrep finner vi *Public Domain Software*, *Freeware* og *Shareware*.

*Public Domain Software* betegner program som forfatteren har gitt fra seg opphavsrett (copyright) til og som er gratis. I praksis er det et program uten lisens. Ved å laste ned programmet blir det din egen eiendom og du kan bruke det slik det passer. Du kan f.eks. lage en ny lisens til programmet, du kan fjerne forfatterens navn og til og med sette inn ditt eget. Det siste er å anbefale hvis man gjør større endringer, da bør man også sette inn en egen lisens, f.eks. lisensiere det som fri programvare [8].

*Freeware* og *Shareware* er proprietær programvare som gis bort gratis, men man får ikke (nødvendigvis) tilgang til kildekoden. *Shareware* kan være gratis i en prøveperiode, og så må man kanskje betale for videre lisens, mens *freeware* fortsetter å være gratis [4].

Microsoft har noe de kaller "*shared source*" [3], som lar utvalgte kunder få tilgang til kildekoden for visse produkt. Men programvaren kan ikke endres og ikke distribueres for kommersiell bruk.

Forholdet mellom mange av disse betegnelse er illustrert gjennom følgende figur<sup>5</sup>, som også illustrerer hva som er gratis programvare (free download). Figuren antyder også noe om sammenhengen mellom åpen kildekode og fri programvare.



### 2.1.1 Historikk: Åpen kildekode og/eller Fri programvare?

Skal man kalle det "åpen kildekode" eller "fri programvare"? I de etablerte miljøene er det en viss diskusjon, for ikke å si konflikt, mellom de to leirene.

<sup>5</sup> <http://www.gnu.org/philosophy/categories.html>

### ***Fri programvare (Free Software, FS)***

Fri programvare (Free Software, FS) er den eldste av disse betegnelsene. Den stammer fra organisasjonen/bevegelsen Free Software Foundation (FSF) som ble stiftet i 1985 av Richard Stallman fra MIT (Massachusetts Institute of Technology). Stallman sier selv at bevegelsen har kjempet for databrukernes frihet siden 1983. FSF er nært knyttet til hans arbeid med det Unix-lignende operativsystemet GNU<sup>6</sup> som startet i 1984, og utviklingen av GPL.

Når programvare er *fri* i FSF sin betydning av ordet, kan hvem som helst bruke den som de ønsker, men aldri gjøre den "ufri". Eventuelle endringer som viderefremmes, deles eller selges må være frie, på samme måte som originalen. Denne tankegangen blir omtalt som "Copyleft" – for å poengtere motsetningen til "Copyright". Dette er illustrert gjennom symbolbruken i følgende figur<sup>7</sup>.



*Copyright – all rights reserved*



*Copyleft – all rights reversed*

Programvare som er utviklet i tråd med FSF sin tankegang, er underlagt lisens selv om man velger å distribuere den gratis; en lisens som skal sikre at "copyleft"-prinsippet følger koden i all framtid.

FSF har oppsummert begrepet "fri programvare" gjennom følgende fire fundamentale friheter<sup>8</sup>:

- Frihet 0. Frihet til å bruke programmet til et hvilket som helst formål*
- Frihet 1. Frihet til å studere hvordan programmet fungerer og til å tilpasse det til egne behov*
- Frihet 2. Frihet til å spre kopier for å hjelpe andre*
- Frihet 3. Frihet til å forbedre programmet og til å spre egne forbedringer slik at alle kan ha nytte av dem*

Som vi ser, er åpen kildekode en forutsetning for fri programvare, spesielt frihetene 1 og 3.

<sup>6</sup> <http://www.gnu.org/>

<sup>7</sup> Fra Wikipedia

<sup>8</sup> Vår oversettelse fra FSF Europe: <http://www.fsfeurope.org/documents/freesoftware.no.html>

### **Åpen kildekode (Open Source Software, OSS)**

Parallelt med Stallmans arbeid med GNU og GPL pågikk det ved University of California, Berkeley lignende arbeid med en egen versjon av Unix. Den lisensen de brukte, BSD (Berkeley Software Distribution), skilte seg ganske mye fra Stallmans GPL, særlig når det gjelder "copyleft"-prinsippet. (Mer om lisensene i avsnitt 2.2.)

Vi får i det hele tatt inntrykk av at det har vært en stadig større misnøye i miljøet med den sterke fokuseringen på de etiske, politiske, filosofiske – nesten religiøse – synspunktene til Stallman og FSF. Man var bl.a. redd for at copyleft-prinsippet ville hindre mange i å ta i bruk programvare lisensiert under GPL i kombinasjon med egenutviklet kode, og dermed hemme utbredelsen av slik programvare. For at fri programvare skulle lykkes kommersielt, var man avhengig av sameksistens med proprietær programvare [34]. Man argumenterte også med at betegnelsen "free software" svært ofte ble misforstått til å bety *gratis* programvare (en misforståelse som i første rekke er knyttet til det engelske språket).

Da Netscape Communications Inc. i 1998 lanserte en åpen/fri versjon av sin nettleser under navnet Mozilla, fikk de utarbeidet en egen lisens. De fikk hjelp til arbeidet av Bruce Perens fra FSF-miljøet og Eric S. Raymond. Disse to utarbeidet da utkastet til en definisjon av åpen kildekode (Open Source Software, OSS), som ledet til etableringen av OSI – the Open Source Initiative. En av oppgavene til OSI er å vurdere om nye lisenser er kompatible med allerede godkjente OSI-lisenser. (En oversikt over godkjente OSI-lisenser finnes bl.a. i Appendix B i [4].)

Det finnes for øvrig også en organisasjon for "Open Hardware" som Bruce Perens står bak<sup>9</sup>.

Definisjonen av åpen kildekode består av ti punkter. Her er tittelen på de ti punktene; nærmere beskrivelse av hvert punkt finnes på OSI sitt eget nettsted<sup>10</sup>:

---

<sup>9</sup> <http://www.openhardware.org/>

<sup>10</sup> Vår oversettelse fra <http://opensource.org/docs/osd>

### Hovedpunkter i definisjonen av åpen kildekode (open source):

1. Fri distribusjon
2. Kildekode
3. Avledet arbeid
4. Integriteten til den opprinnelige forfatterens kildekode
5. Ingen diskriminering av personer eller grupper
6. Ingen diskriminering av felt eller bruksområde
7. Distribusjon av lisens
8. Lisensen må ikke være spesifikk for et produkt
9. Lisensen må ikke legge begrensning på annen programvare
10. Lisensen må være teknologinøytral

Det er en tendens til at de som bruker betegnelsen *åpen kildekode* framhever tekniske fordeler ved programvaren, som f.eks. pålitelighet og sikkerhet, mens de som bruker betegnelsen *fri programvare* vektlegger frihet fra kontroll og etiske spørsmål [3].

Stallman sier [10] at for "the Open Source movement" er spørsmålet om kildekode skal være åpen et praktisk anliggende, ikke et etisk spørsmål. "Open source" er en utviklingsmetodikk, mens "Free Software" er en sosial bevegelse. Som Christian Payne [14] poengterer, innebærer begrepet "Open source" mer enn at kildekode er åpen: Det skal også være tillatt å endre og videreformidle koden. Ved "Open source" vektlegges en utviklingsmetodologi basert på samarbeid.

Men til tross for disse (tilsynelatende) motsetningene mellom de to miljøene (FSF og Stallman på den ene siden og OSI på den andre), så er det slik at lisenser som er godkjent som åpen kildekode i stor grad også anses som fri programvare, og omvendt [34]. Og begge miljøene har som mål å bidra til utbredelse av åpen kildekode.

### **Diskusjoner om begrepene *Åpen kildekode* og *Fri programvare* i Norge**

Her i Norge sies det<sup>11</sup> at enkelte kommersielle aktører mener at begrepet *åpen kildekode* er forvirrende og ikke bør benyttes. Dette begrunnes gjerne med at man her i landet ikke behøver å ta inn over seg motsetningene i miljøet. De mener at man i Norge dermed burde klare seg med begrepet *fri programvare* for å beskrive fenomenet.

På den annen side bruker Peter Hidas i Computerworld<sup>12</sup> betegnelsen *åpen kildekode* – "selv om myndighetene i Norge har falt ned på det tvetydige, men selgende navnet *fri programvare*", som han sier.

### **Begrunnelse for vårt valg av begrepet *Åpen kildekode***

Vi vet ikke hva som er begrunnelsen for navnebruken fra myndighetenes side, dvs. hvor bevisst de er på navnevalget. Vi sier som Ross Anderson [17] at vi ikke ønsker å ta stilling i

<sup>11</sup> Norsk Wikipedia: [http://no.wikipedia.org/wiki/%C3%85pen\\_kildekode](http://no.wikipedia.org/wiki/%C3%85pen_kildekode) (sist sett 19.09.08)

<sup>12</sup> Computerworld 08.02.08

debatten eller konflikten mellom leirene for *fri programvare* og *åpen kildekode*. Ross Anderson faller ned på betegnelsen "åpne system" (open systems). I sin juridiske avhandling om "Copyleft" velger Torger Kielland å kalle dette for "åpen programvare" [34].

For enkelthets skyld, og for å unngå kompliserte kombinasjonsuttrykk som "åpen kildekode/ fri programvare" eller en av de mange forkortelsene som er i bruk (se avsnitt 2.1.2), måtte vi enten velge én av betegnelse som er i bruk, eller lage en ny.

I denne rapporten velger vi å bruke betegnelsen ÅPEN KILDEKODE. Fordi:

Utgangspunktet for *vår* problemstilling var den åpne kildekode, tilgangen til kildekode, det å kunne se koden og modifisere koden: Er det en trussel at uvedkommende kan se kildekode, se hvordan sikkerhetsløsninger er implementert, kanskje til og med gå inn og endre kode eller legge til sitt eget? Vi hadde ikke de politiske/filosofiske ideene fra FSF som utgangspunkt (selv om vi ikke dermed tar avstand fra disse synspunktene).

I enkelte sammenhenger i rapporten er det fortsatt naturlig å holde på betegnelsen "fri programvare", der dette er viktig for sammenhengen.

### 2.1.2 Mange forkortelser

Selv om definisjonene for fri programvare og åpen kildekode i hovedsak er like, brukes det fortsatt flere ulike begrep og tilhørende forkortelser. Her oppsummerer vi noen av dem:

- **FS** – Free Software (norsk: fri programvare). Dette er betegnelsen som brukes av Richard Stallman og FSF-miljøet. De poengterer at "free" handler om frihet, ikke pris (like "free speech" not "free beer"), at "free" må forstås i betydningen "freedom". De definerer fire grunnleggende friheter for programvare-brukerne (se avsnitt 2.1.1).
- **OSS** – Open Source Software (norsk: åpen kildekode). Begrepet ble introdusert bl.a. for å redusere forvirringen om de eksisterende betegnelse "Free Software" og "Freeware" (en forvirring som stammer fra tvetydigheten i det engelske ordet 'free': *gratis* og *fri* i betydningen fritt tilgjengelig, ikke bundet).

Men også begrepet "open source" kan misforstås til å bety kun det at kildekode kan ses, og ikke det at den også kan endres og distribueres videre. Sammen med de mange typene lisenser har dette bidratt til å opprettholde forvirringen, og det har derfor dukket opp flere betegnelse og forkortelse:

- **F/OSS** – Free/Open Source Software
- **FOSS** – Free and Open Source Software
- **FLOSS** – Free/Libre and Open Source Software (der 'Libre' brukes for å poengtere at *free* er noe annet enn *gratis*)

## 2.2 Lisenstyper

Det finnes et stort antall<sup>13</sup> lisenser i kategoriene fri programvare og åpen kildekode. I dette avsnittet beskriver vi spesielt ytterpunktene representert av GPL (GNU General Public Licence) som den mest restriktive, og BSD (Berkely Software Distribution) som den mest liberale. Man snakker gjerne om BSD-type lisenser og GPL-type lisenser. Skillet går spesielt

---

<sup>13</sup> Over 60 – se <http://www.gnu.org/philosophy/license-list.html#GPLCompatibleLicenses> og [http://en.wikipedia.org/wiki/List\\_of\\_FSF\\_approved\\_software\\_licenses](http://en.wikipedia.org/wiki/List_of_FSF_approved_software_licenses)

på om lisensene er "copyleft" eller ikke, dvs. om programvaren kun kan distribueres videre under samme lisens.

### 2.2.1 GPL

GPL (GNU General Public License) er tilknyttet FSF og er en copyleft-lisens. GPL er den mest utbredte av alle lisensene for fri programvare og åpen kildekode. Lisensen er svært restriktiv ved integrering med annen programvare. Ved integrering av programvare lisensiert under GPL i annen programvare, må det resulterende programmet i sin helhet lisensieres under GPL for alle andre formål enn intern bruk.

Man kan lett forledes til å tro at GPL krever offentlig frigjøring av kildekode. Men det er ikke tilfelle. Det GPL krever er at de som får eller kjøper binærkode også får kildekode. Hvis man f.eks. gjør endringer i koden for internt bruk, trenger man ikke frigi disse endringene. Det er kun hvis programvaren gis videre at kildekode må følge med (men ikke nødvendigvis publiseres offentlig).

GPL-lisensen er eksplisitt knyttet til aktivitetene **kopiering, distribuering og modifisering** av koden. Det betyr at kjøring/eksekvering av programvare lisensiert under GPL *ikke* omfattes av lisensen. Det gjør heller ikke nedlasting av applikasjoner lisensiert under GPL til eget bruk, dvs. programvare som kjører på interne servere og *ikke* innebærer kopiering til andre virksomheter. I slike tilfeller, hvor lisensen ikke gjelder, dekkes utviklerne av programvaren heller ikke av de ansvarsfraskrivelsene som ellers ligger i lisensen, f.eks. mot "direkte" skade<sup>14</sup> som følge av bruk av programvaren. Det er en av grunnene til at noen utviklere velger å "dobbel-lisensiere" for å sikre seg ansvarsfraskrivelse [4]. De lisensierer da programvaren under både GPL og en annen lisens. Den andre lisensen ivaretar ansvarsfraskrivelse i de tilfeller GPL-lisensen ikke gjelder.

GPL er et politisk manifest i tillegg til å være en programvarelisens [8]. Dette holder nok noen borte fra å bruke lisensen. Men juridisk sett er GPL bedre enn de fleste andre lisenser fordi den er skrevet med hjelp av juss-professorer.

Lisensen GPL er selv ikke underlagt GPL: Det er nemlig ikke lov til å endre den. Alle endringer til koden må gis videre under samme uendrede lisens, og all kode som integreres med kode lisensiert under GPL må gis ut under GPL.

#### *GPL-versjoner*

GPLv1 ble publisert i 1989. GPLv2 kom i 1991. Versjon 3 (v3) av lisensen kom i november 2007.

En årsak til å lage v3 var at man ønsket å hindre "tivoization" – den egenskapen at utstyr (HW) som inneholder programvare lisensiert under GPL slutter å fungere dersom det oppdages at man har modifisert denne programvaren. En annen viktig grunn til å endre lisensen var å bekjempe patentering av programvare.

Merk at GPLv2 og GPLv3 "pr. def." er inkompatible<sup>15</sup>. Begge sier at dersom man legger inn kode med denne lisensen i annen kode, så må resten av koden ha samme lisens.

---

<sup>14</sup> Eksempel på "direkte" skade er eventuelle tapte utgifter til kjøp av programvaren ("lisens-avgift"). Annet regnes som "indirekte" skade og omfattes *ikke* av garantien, f.eks. evt. tap av inntekt fordi programvaren ikke virket slik den skulle. Merk at dette vanligvis også er tilfelle for kommersiell programvare: Garantien dekker utgifter til kjøp av programvaren, men ikke de indirekte tapene.

<sup>15</sup> Richard Stallman: <http://gplv3.fsf.org/rms-why.html>

## Omfanget av bruken av GPL

I 2002 var 72-73 % av fri/åpen programvare lisensiert under GPL, og de nærmeste på lista hadde mindre enn 10 % hver [3]. Computerworld<sup>16</sup> sier i desember 2007 at ca 65 % av alle registrerte åpne prosjekt er lisensiert under GPL. De fleste er GPLv2 (bl.a. Linux), men stadig flere tar i bruk GPLv3.

### GPL og kompatibilitet med andre lisenser

GPL er kompatibel med en del av de øvrige lisensene for fri/åpen kildekode, bl.a. Apache-lisensen (i følge Stallmann selv<sup>15</sup> gjelder dette GPLv3, men ikke v2.1). Også den modifiserte versjonen av BSD-lisensen (se avsnitt 2.2.2) er kompatibel med GPL.

En fullstendig oversikt over kompatibilitet mellom GPL og andre lisenser finnes her:

- <http://www.gnu.org/philosophy/license-list.html#GPLCompatibleLicenses>

Liste over lisenser som er godkjent av FSF finnes her:

- [http://en.wikipedia.org/wiki/List\\_of\\_FSF\\_approved\\_software\\_licenses](http://en.wikipedia.org/wiki/List_of_FSF_approved_software_licenses)

Kompatibilitet i denne sammenheng betyr at programvare som er lisensiert under en av disse lisensene, kan tas inn i GPL-programvare, og alt blir dermed lisensiert under GPL. Kompatibiliteten går bare den ene veien, kode lisensiert under GPL kan ikke gis ut under en annen lisens. En lisens er kompatibel med GPL dersom denne lisensen ikke forbyr at programvare lisensiert under denne lisensen tas inn i GPL-programvare og at resultatet i sin helhet lisensieres under GPL. To lisenser som begge krever at det resulterende programmet lisensieres under opprinnelig lisens, er altså **inkompatible**.

Lisenser som ikke er kompatible med GPL medfører praktiske problem, fordi mesteparten av GPL-programvare dermed ikke kan kombineres med disse. Dette gjelder bl.a. for de aller fleste lisenser for proprietær programvare.

GPL-programvare som endres kan brukes til internt bruk uten lisens, men så snart den skal frigis eller brukes eksternt, må lisensen følge med. Program som tar inn (bruker) kode lisensiert under GPL må i sin helhet lisensieres under GPL. Dette blir noen ganger nedsettende kalt "the open source virus" [4], fordi kravet kan føre til at stadig mer programvare blir lisensiert under GPL.

I Europa er det laget en egen versjon av GPL, EUPL (European Union Public License, versjon 1.0 fra 9. januar 2007)<sup>17</sup>. De sier de har som mål å være kompatibel med GPL. EUPL er "copyleft". Den er i dag kompatibel med GPLv2, men ikke med GPLv3. Det er uklart om man i Norge vil ta i bruk EUPL<sup>18</sup>. FSF liker ikke at EU lager sin egen variant, og de godtar ikke at kildekode lisensiert under GPL flyttes over i EUPL.

## 2.2.2 BSD

BSD-lisensen (Berkeley Software Distribution License) har en historie som går minst like langt tilbake i tid som GPL. Med sin opprinnelse i universitetsmiljø (se avsnitt 2.1.1), blir BSD og lignende lisenser også omtalt som "akademiske lisenser" [34].

BSD er den mest liberale lisensen, den tillater det meste. Programvare man har fått under en BSD-lisens kan gis videre under en annen lisens så lenge den opprinnelige copyright-

---

<sup>16</sup> Computerworld 14.12.07

<sup>17</sup> Se: <http://en.wikipedia.org/wiki/EUPL> og <http://ec.europa.eu/idabc/en/document/7330/5980>

<sup>18</sup> <http://www.digi.no/juss+&+samfunn/ny+%ABeuropeisk+gpl%BB+skaper+lisenskr%F8ll/art499937.html>

notisen og tilhørende "disclaimer" tas med. BSD er ikke "copyleft", hvem som helst kan bruke slik kode og "låse den inne" i sin egen lukkede/proprietære lisens<sup>19</sup>. For dem som sverger til bruk av akademiske lisenser, er retten til å modifisere programmet uten restriksjoner det sentrale [34].

Den største forskjellen fra GPL er altså at programvare med BSD-lisens kan gjøres "privat", dvs. kan gjøres om til proprietær/lukket kode. Programvaren vil da ikke lenger være åpen, og vil heller ikke være under BSD-lisensen. Den opprinnelige versjonen av programmet vil fortsatt være tilgjengelig på åpne vilkår, men den nye lisensstakeren kan modifisere programmet og gjøre slike modifikasjoner private [34]. Det er dette som kan føre til "forking" – det oppstår ulike varianter av (opprinnelig) samme programvare, slik det skjedde med BSD-versjonen av Unix. Perens [8] hevder at noen også frykter at denne muligheten i BSD kan føre til at noen ondsinnet modifiserer kode og utgir den for å være fra den opprinnelige utvikleren, og således setter denne i et dårlig lys. Noen kunne også modifisere den til å inneholde sikkerhetshull, f.eks. legge inn en trojansk hest, og så gjøre den lukket. Dette takles imidlertid av straffelovgivningen. Det er *ingen* lisenser som garanterer mot kriminalitet [8].

Det er to varianter av BSD-lisensen:

1. BSD-old (original BSD)
2. BSD-new (revised BSD, modified BSD, eller 3-clause BSD)

Den opprinnelige BSD-lisensen hadde et ekstra punkt som krevde at all programvare som var basert på BSD-lisensiert kode skulle inneholde en henvisning til utviklerne av den opprinnelige kildekoden. Dette var tredje punkt (av fire), og hadde følgende formulering<sup>20</sup>:

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:  
*"This product includes software developed by the University of California, Berkeley and its contributors."*

Et problem med dette punktet var at de som brukte denne lisensen satte inn navnet på sin egen organisasjon (i tillegg til eller i stedet for Berkeley), og når mange slike program ble satt sammen til et større system, ville annonseringen måtte oppgi en lang rekke slike henvisninger. Richard Stallman sier<sup>21</sup> at han hadde talt opptil 75 ulike henvisninger i en og samme programvare. BSD ble på grunn av dette punkt også inkompatibelt med GPL, som ikke tillater at det legges ekstra restriksjoner på kode lisensiert under GPL.

Dette tredje punktet ble fjernet fra BSD-lisensen i 1999, og det ble fra da av oppfordret til å heller bruke den modifiserte versjonen av lisensen (BSD-new). Men fortsatt finnes det mye programvare som har beholdt den opprinnelige BSD-lisensen. Av den grunn mener både Richard Stallman<sup>19</sup> og Bruce Perens [8] at det er bedre å bruke en annen BSD-lignende lisens (X-lisensen, også kalt X11-lisensen eller MIT-lisensen).

---

<sup>19</sup> Joe Barr, 2001: <http://www.itworld.com/AppDev/350/LWD010523vcontrol4/>

<sup>20</sup> Fra Wikipedia: [http://en.wikipedia.org/wiki/Bsd\\_license](http://en.wikipedia.org/wiki/Bsd_license)

<sup>21</sup> <http://www.gnu.org/philosophy/bsd.html>

### 2.2.3 Andre lisenser

Det finnes et stort antall lisenser som ligger mellom det vi kan kalle ytterpunktene, GPL og BSD. Overly [4] lister opp 34 lisenser som pr. 2003 var godkjent av OSI, mens listen vist til i avsnitt 2.2.1 over lisenser som i dag er godkjent av FSF<sup>22</sup> inneholder over 60.

Blant de mest kjente av de GPL-lignende lisensene nevner vi:

- LGPL ("Lesser GPL" eller "Library GPL") – var opprinnelig tiltenkt programvarebibliotek, men brukes i dag også for annen programvare. Slik kode kan under visse forutsetninger innebygges i et annet program, uten at hele det nye programmet får LGPL. Denne lisensen kan konverteres til GPL.
- Netscape Public License (NPL) og Mozilla Public License (MPL) har utgått fra GPL og har lignende betingelser som denne [4]. En viktig egenskap ved NPL-lisensen var at endringer som kom inn kunne re-lisensieres under en proprietær/lukket lisens. Dette gjorde de for å fortsatt kunne ivareta de kundene som hadde kjøpt Netscape Navigator med lukket lisens. Men Netscape fikk i tillegg utarbeidet MPL som ikke har denne muligheten. Bruce Perens [8] beskriver utviklingen av disse lisensene. Det arbeidet Bruce Perens og Eric Raymond gjorde med disse lisensene dannet grunnlaget for etableringen av OSI (se avsnitt 2.1.1).

På den annen side er det andre lisenser som er svært lik BSD [4], slik som:

- Apache Software License
- MIT-lisensen (også kalt X-lisens eller X11-lisens)

Bruce Perens [8] påpeker at ett og samme program kan utgis med flere lisenser, f.eks. både fritt under GPL og selges under en kommersiell lisens. På denne måten skapes en betalingsvillighet for den proprietære lisensen fra kommersielle selskap som ikke ønsker å gi ut egne program under GPL og dermed frigi kildekoden til disse slik copyleft-klausulen ville kreve [34]. Det er også mulig å "dobbel-lisensiere" (*dual-license*) under to "frie" lisenser.

## 2.3 Bruksaspekt

Åpen kildekode er i bruk på alle nivå av IKT-system, fra basale funksjoner som operativsystem (f.eks. Linux og FreeBSD), desktop-applikasjoner og verktøy (f.eks. OpenOffice og Eclipse), til bibliotek og programvaremoduler (f.eks. krypto-bibliotek) og komplette spesial-applikasjoner.

De ulike typene betegnes ofte som henholdsvis *horisontale* og *vertikale* system eller applikasjoner [24]:

- **Horisontale** applikasjoner: grunnleggende, generelle applikasjoner som kan brukes innenfor mange områder, f.eks. operativsystem og generelle verktøy og desktop-applikasjoner.
- **Vertikale** applikasjoner: spesifikke for sine områder, f.eks. applikasjoner for helsesektoren.

Markedet for vertikale applikasjoner er selvsagt mye mindre enn markedet for horisontale applikasjoner.

---

<sup>22</sup> [http://en.wikipedia.org/wiki/List\\_of\\_FSF\\_approved\\_software\\_licenses](http://en.wikipedia.org/wiki/List_of_FSF_approved_software_licenses)

En stor andel av den åpne kildekoden som er i bruk er programvarebibliotek og -moduler. Store applikasjoner, også proprietære, inneholder dusinvis av komponenter som er åpen kildekode. Men de fleste kommersielle leverandører vil holde tett med om og i hvilken grad dette er tilfelle for deres produkt [4].

Den økte utbredelsen av åpen kildekode ville knapt vært mulig uten den voldsomme utviklingen av **Internett**. Gjennom Internett blir utviklerne i stand til å dele sine program med andre og utvikle program uavhengig av hvor i verden de befinner seg. Dette gjør at programvare med åpen kildekode har en utpreget internasjonal karakter [34].

Både store og små leverandører og utviklere av åpen kildekode har egne nettsted der programvaren kan lastes ned fra, via Internett. SourceForge<sup>23</sup> er ett slikt nettsted der prosjekt og enkeltpersoner kan legge ut programvare for nedlasting. Pr. 1. oktober 2008 ligger det ute programvare fra mer enn 180.000 prosjekt på SourceForge.

Andre utviklere av åpen kildekode, f.eks. Linux, kan ha profesjonelle distributører. Linux har selv en liste med linker til ulike distributører som programvaren kan lastes ned fra<sup>24</sup>.

Via Internett har man også tilgang til diskusjonsfora, e-postlister og nyhetsgrupper for utveksling av informasjon om programvaren. Når man skal ta i bruk denne typen programvare er det viktig å vurdere om programvaren kommer fra et aktivt, levende miljø. Det ser man både ut fra hvor nye og hvor hyppige oppdateringene på nettstedet er og hvor stor aktivitet det er i diskusjonsfora og lignende.

Utbredelsen av og suksessen til åpen kildekode-program vil også avhenge av hvor enkelt det er å finne og få tak i slik programvare, dvs. hvordan presentasjon og tilgjengelighet på et nettsted laget spesielt for det aktuelle programmet er [20].

Åpen kildekode innebærer at kildekoden skal være tilgjengelig for de som kjøper, får eller laster ned programvaren. Det vanlige er at det er den kjørbare versjonen som lastes ned, slik det f.eks. er med Linux-distribusjoner. Så er det opp til kunden/brukeren selv å laste ned kildekoden i tillegg. Payne [14] mener at dette er noe de færreste gjør. Flertallet har verken interesse av eller kunnskap nok til å bry seg om kildekoden.

Selv om det bare er utviklere og kode-review'ere som har et reelt *behov* for kildekoden, er *muligheten* for å kunne se koden viktig for alle. Wheeler [3] sammenligner det med å kjøpe ny bil: Vil man kjøpe en bil der det ikke er mulig å åpne panseret, selv om man ikke skjønner noe av det som finnes under panseret? Det å kunne se kildekoden betyr at vi som kunde har kontroll over produktet vi har kjøpt, uten å være avhengig av leverandøren.

Også når det gjelder valg av lisensstyper vil behovene og interessen være forskjellig, alt etter om det er sluttbrukere eller utviklere som laster ned programvaren. Torger Kielland [34] mener at *sluttbruker* kun er ute etter å kjøre programmet og vil være likegyldig til den mulighet lisensen gir ham til å modifisere programmet og tilgjengeliggjøre endringene. I valget mellom to lisenser som gir ham tillatelse til å kjøre programmet, vil han derfor, av egen nytte, velge den billigste. En *utvikler*, derimot, vil kunne etterspørre programmet under en lisens som gir ham tilgang til kildekoden og tillatelse til å bearbeide og tilgjengeliggjøre programmet for andre.

Motivasjonen for å ta i bruk åpen kildekode kan være forskjellig, men mange velger det nok av ulike økonomiske grunner: Uten lisenskostnader blir programvaren billigere i innkjøp og

---

<sup>23</sup> <http://sourceforge.net/>

<sup>24</sup> <http://www.linux.com/distributions/>

drift, og utviklingsmiljøene sparer ressurser og penger på å ta inn ferdig utviklede moduler i stedet for å utvikle alt selv.

Wheeler [3] påpeker at åpen kildekode allerede er i utstrakt bruk i offentlig sektor i mange land, eller at de har en politikk på å bruke slik programvare. Han mener hovedmotivasjonen for dette er kostnadsreduksjon, men det tas også hensyn til faktorer som programvarens pålitelighet og ytelse. Blant andre argument nevner han:

- å dra nytte av lokalt næringsliv (for vedlikehold og videreutvikling) er lettere å få til med åpen kildekode enn med proprietær programvare
- økt konkurranse og redusert avhengighet av én enkelt leverandør
- enklere å tilpasse programvaren til lokale behov, f.eks. til små språkgrupper
- økt sikkerhet; ved at myndighetene har mulighet til å sette seg inn i sårbarheter og forbedre sikkerheten til produktet. Noen myndigheter kan ha problem med å stole på lukket programvare utviklet i andre land.

Også Payne [14] hevder at myndighetene i enkelte land (han nevner Kina og Tyskland som eksempler) har mistillit til lukket programvare utviklet av f.eks. amerikanske firmaer. De velger åpen kildekode (Linux m.fl.) av frykt for at programvare de ikke får kildekode til kan inneholde noen slags "overvåkningsmoduler".

## 2.4 Juridiske og økonomiske aspekt

Wheeler [3] argumenterer mot en rekke (unødvendige) bekymringer når det gjelder bruk av åpen kildekode. De kan i hovedsak oppsummeres i tre tema:

1. Ansvar for utvikling, oppgradering, vedlikehold og support.
2. Juridiske problemstillinger.
3. Økonomiske og forretningsmessige spørsmål.

I de følgende avsnittene gir vi en kort utredning om disse tre temaene.

### 2.4.1 Utvikling, oppgradering, vedlikehold og support

Overly [4] sier at de *største* bidragene til åpen kildekode-produkt kommer fra betalte medarbeidere i profesjonelle organisasjoner eller stiftelser. De konkurrerer også seg imellom.

Wheeler [3] viser til en undersøkelse for å finne ut *hvem* utviklerne av åpen kildekode er. Undersøkelsen konkluderte med at ca 25 % av disse er profesjonelle, dvs. har behov for programvaren profesjonelt, i jobbsammenheng. Resten gjør det for moro skyld, som hobby, eller av overbevisning.

Dette kan lede til en bekymring for at support for åpen kildekode dermed er fundamentalt dårligere enn for proprietær programvare.

Det er i hovedsak to typer support for åpen kildekode: a) vanlig support fra leverandører eller utenforstående, som det betales for, og b) uformell support fra "det åpne miljøet" via nyhetsgrupper, diskusjonsfora, web eller andre elektroniske fora.

I tillegg til frihet og tilgang til kildekode, dreier det seg her om en egen utviklingsmetodikk, Raymond [6] kaller det "basar<sup>25</sup>-stilen" – i motsetning til "katedralstilen".

---

<sup>25</sup> Basar er her i betydningen et åpent (arabisk) marked der hvem som helst kan komme med sine varer og produkt som selges for en billig penge, og der kunden kan få tak i det meste mellom himmel og jord.

Basar-stilen som utviklingsmetodikk innebærer et stort antall bidragsyttere, et stort antall utviklere som leverer rettinger, oppdateringer og forbedringer. Det er snakk om små, inkrementelle releaser (versjoner). Når de leveres må de kompilere og være kjørbare slik at de kan testes. En liten andel bidragsyttere leverer størstedelen av programvaren, men bidragene fra de øvrige må ikke undervurderes. Det at mange gjennomgår koden og finner og/eller fikser feil, er med på å gjøre andre utviklere mer produktive. Når mange øyne saumfarer koden, er det stor sannsynlighet for at feil blir oppdaget. Dette er utgangspunktet for Linus' lov om "de mange øyne som ser" [6] *"Given enough eyeballs, all bugs are shallow."*

Men alle øyne er ikke like gode i denne sammenheng. Feilrapportering fra dem som ser og forstår kildekoden er mye mer verdifull enn fra dem som bare bruker koden. Forslag til forbedringer kommer ikke fra et tilfeldig utvalg av mennesker, men fra personer som er tilstrekkelig interessert i å bruke programmet, lære hvordan det virker, prøve å finne løsninger på problem de oppdager og produsere en tilsynelatende fornuftig retting.

Videreutvikling (rettinger, oppdateringer, forbedringer) av åpen kildekode skjer vanligvis i vel strukturerte former, ofte innenfor rammene av et prosjekt som har en form for toppstyring i form av en gruppe kjerneutviklere som styrer prosjektet og bestemmer hvilken kode som tas inn og hvilken som avvises. En av utviklerne av Unix BSD har sagt at de forkastet 90 % av de bidragene og forslagene som kom inn. Det er få av de åpne prosjektene som vokser seg ut over en håndfull kjerneutviklere [34].

Her er noen eksempler på utviklingsmetodikk fra de åpne miljøene [3]:

#### Linux-hierarkiet:

Utviklingen av Linux-kjernen foregår i et hierarki i fire nivå:

- 1) Vanlige utviklere, "fotfolket" – de kan foreslå endringer, men leverer sine forslag til en "ivaretaker" (engelsk: maintainer).
- 2) Ivaretakere – som leverer et samlet sett av forslag de anbefaler videre til en tiltrodd "løytnant".
- 3) Løytnanter – som leverer videre til diktatoren.
- 4) Den velvillige diktatoren – "her: Linus Torvalds"

#### Apache-web server:

Utviklingen gjennomføres av ei gruppe, en prosjektorganisasjon. For å bli med i gruppen må man bli *invitert* inn. Ny funksjonalitet tas inn etter en gjennomgang/vurdering ("Review-then-commit"), mens vedlikehold og rettinger legges inn av en tiltrodd utvikler, og gjennomgås i etterkant ("Commit-then-review").

#### SourceForge:

Mange åpen kildekode-prosjekt er lagt inn under SourceForge, som tilbyr nødvendig verktøy for versjonskontroll. De få som har skrivegang her gjør sine oppdateringer selv, mens de som ikke har skrivegang, legger ut sine forslag i en database eller sender det til ei e-postliste og ber noen med skriveaksess om å legge det inn.

Man kan spørre seg om det er større risiko for at åpen kildekode blir overlatt til seg selv enn at kommersielle produkt blir det. Det er det neppe: Leverandører av programvare legges ned, og personer mister interessen, i begge leire. Det er et faktum at kommersielle leverandører "går under", legges ned, gir opp produkt, stopper vedlikehold av gamle versjoner eller mister kontrollen med kildekoden. Firma kan opphøre å eksistere, bli opp-

kjøpt, eller gå ut av markedet av andre grunner, f.eks. dersom markedet blir for lite. Dersom dette skjer med proprietær, lukket kildekode, er ikke programvaren tilgjengelig for videreutvikling, med mindre det er inngått spesielle avtaler om dette. På den annen side, med åpen kildekode er det større sjanse for at noen før eller senere griper tak i koden/programmet og viderefører det. I sin beskrivelse av basar-stilen poengterer Raymond [6] at når man mister interessen for et program bør man sørge for å overlevere det til andre som kan ivareta programmet, videreutviklingen og vedlikeholdet.

#### 2.4.2 Juridiske problemstillinger

Wheeler [3] spør om åpen kildekode er mer risikabel enn proprietær programvare juridisk og avtalemessig. Det er ingen som kan saksøkes om ting går galt. Han påpeker at det samme faktisk også er tilfelle for de fleste proprietære lisenser. Den største forskjellen er at med proprietær programvare kan brukeren bli saksøkt dersom han bryter lisensen.

Programvare er underlagt *copyright* (opphavsrett), programvare blir sett på som "åndswerk", også i Norge (§1 i Åndsverkloven [33]). Norske forhold er belyst i en avhandling av Torger Kielland [34], med hovedfokus på *copyleft*-lisensiering.

Det blir ofte hevdet at "copyleft" kan virke ødeleggende på Intellectual Property Rights (IPR). Overly [4] diskuterer hvordan man kan unngå at man mister IPR til proprietær programvare som er utviklet og kjøres under Linux – som jo er lisensiert under GPL .

Lisensen for åpen kildekode følger med kildekoden og er vanligvis ikke synlig for sluttbrukeren av programmet. Lisensen ligger som ei av flere filer på CD-en som inneholder programvaren, eller den kommer med i den komprimerte programvare-fila som lastes ned fra nettet. Det er *ikke* slik at den som laster ned eller installerer slik programvare får presentert lisensen og må trykke på en "godta"-knapp. Det er uvisst hvordan dette ville bli tolket i en evt. rettssak om copyright [4]. Har du akseptert lisensen hvis du ikke visste at den var der?

Avtaler og lisenser knyttet til fri programvare har vært prøvd for retten bare i et fåtall saker. En rettssak i München i 2004, mot et firma som ikke overholdt lisensieringsreglene i GPL, endte med at firmaet straks gjorde nødvendige tilpasninger til lisensen [3]. En lignende sak fra USA om MySQL, er kort omtalt av Torger Kielland [34].

#### 2.4.3 Økonomiske og forretningsmessige spørsmål

Er åpen kildekode et levedyktig konsept økonomisk? Det finnes bedrifter som tjener penger på åpen kildekode. HP sier de tjente 2 mrd US\$ på Linux i 2003. IBM sa i 2002 at de hadde tjent inn nesten hele sin 1 mrd US\$ satsing på Linux i løpet av ett år [3].

Det er ikke slik at all åpen kildekode utvikles gratis og på hobbybasis. Ofte står det store organisasjoner med tung finansiering bak.

Watson et al. [26] skiller mellom fem ulike modeller når det gjelder utvikling og distribusjon av programvareprodukt:

1. Proprietære fellesskap (*Proprietary communities*)

Koden blir sett på som firmaets intellektuelle ressurs og beskyttes gjennom copyright og patenter. Programvaren selges vanligvis med lisenser, men kan i noen tilfelle distribueres gratis ("freeware"). Det er ikke prisen som skiller proprietær kode fra åpen kode, men publikums mulighet til å se og modifisere koden.

2. Åpne fellesskap (*Open communities*)

Programvareutvikling og support gjøres av frivillige med liten eller ingen kommersiell interesse. Det største antall åpen kildekode-prosjekt er i denne kategorien. De fleste kan finnes på internett, f.eks. hos SourceForge<sup>26</sup>.

3. Distribusjonsselskap (*Corporate distribution*)

Disse tjener pengene sine på distribusjon og support for et utvalg av åpen kildekode. Mange brukere har problem med å identifisere og holde kontakten med de åpne utviklingsmiljøene, og disse firmaene tar på seg denne formidlingen. Eksempler på slike er Linux-distributøren RedHat og det norske selskapet LinPro<sup>27</sup>.

4. Sponset utvikling (*Sponsored Open Source*)

Mange åpen kildekode-prosjekt støttes av organisasjoner eller stiftelser. Et slikt eksempel er Apache Software Foundation, som i tillegg til Apache server også sponser utviklingen av et femtitalls andre produkt, og IBM som bidrar med utviklere til Apache server. Noen åpen kildekode-produkt var i utgangspunktet lukket kode som senere er blitt frigjort. Et eksempel på dette er utviklingsverktøyet Eclipse som ble frigitt av IBM. Det er fortsatt IBM's egne utviklere som vedlikeholder produktet og bidrar til videreutvikling av det. Åpningen av Netscape-kildekoden og etableringen av Mozilla er et lignende eksempel.

5. Andre generasjon åpen kildekode (*Second generation Open Source, OSSg2*)

Dette kalles også "Professional OSS", og er en blanding av typene 3 og 4 foran. Professional OSS-organisasjoner tjener penger på support og tilleggstjenester, og de bidrar med utviklingsressurser. De eier og kontrollerer programvaren og det er de som bestemmer hvordan videreutviklingen skal være, basert på strategiske planer for hvordan brukerne kan få best mulig nytte av programvaren [27]. Flere av disse firmaene baserer seg på dobbelt-lisensiering, både kommersiell og åpen lisens. Kunder som gjør endringer i programvaren, men ikke ønsker å gi endringene tilbake, må da kjøpe en kommersiell lisens.

Som eksempler på slike firmaer nevnes JBoss, MySQL, Trolltech og Sleepycat. Alle disse er firmaer med rask vekst og god fortjeneste – og som er blitt oppkjøpt av andre i løpet av de siste par årene.

De to ytterpunktene er 1 og 2. De tre siste er hybrider mellom disse. Watson et al. konkluderer med at type 5, det de kaller Second generation OSS (OSSg2), er den forretningsmodellen som vil ha størst overlevelsessevne de nærmeste årene.

Fitzgerald [27] betegner type 5 som OSS 2.0, en balanse mellom kommersiell profitt og åpen kildekode-verdier. Dette er en utviklingsmodell som er mindre "bazaar-lik", med sterkere prosjektstyring, utviklere som får betalt for å utvikle åpen kildekode, og leverandører som gjør forretning på support. Bedriftene/organisasjonene kontrollerer utviklingsmiljøet og stort sett all kode som tas inn i produktet.

Man må også se på etterspørselssiden, ikke bare tilbyderne. Det er brukerne/kundene som oftest sparer store kostnader – i tillegg til at de blir fritatt for kontroll eller begrensninger fra leverandører. Disse brukerne kan være villige til å betale for å holde åpen kildekode i live i

---

<sup>26</sup> <http://sourceforge.net/> – som hadde over 183.000 registrerte prosjekt ved utgangen av juli 2008, hvorav 670 i kategorien "Medical Science Applications".

<sup>27</sup> <http://www.linpro.no/>

stedet for å stadig kjøpe nye versjoner av proprietær programvare. Åpen kildekode er heller ikke så sterkt knyttet opp mot leverandørens økonomiske situasjon som proprietær programvare er.

Det koster praktisk talt ingenting å lage kopier av programvare. Det er *det* programvareleverandører som Microsoft har tjent seg rik på: Å selge produkt det nesten ikke koster noe å kopiere. Men det trengs noen ressurser for å produsere programvaren i utgangspunktet, bl.a. arbeidskraft. Det er ressurser det kan være knapphet på og som koster penger. Om man ikke får dekt slike kostnader vil det kunne bli produsert mindre programvare enn det er behov for [34].

Utbredelsen av åpen kildekode vil neppe ødelegge programvareindustrien. Stadig mer av den åpne kildekoden utvikles og vedlikeholdes kommersielt. Det er laget mange system for å knytte sammen utviklere og folk som er villig til å betale for oppgraderinger og forbedringer.

Åpen kildekode utelukker heller ikke konkurranse. Det finnes mange eksempler på konkurrerende produkt i denne kategorien. Og dersom det bare finnes ett produkt kan det konkurreres på vedlikehold og support.

Motivasjonen for bedrifter som utvikler åpen kildekode er også vidt forskjellig: Noen gjør det for å selge support, noen gjør det for å selge komplementære produkt ved siden av. Eric S. Raymond har i sin bok "The Magic Cauldron" [7] beskrevet flere ulike forretningsmodeller for hvordan man kan gjøre penger på åpen kildekode.

- **Markedsposisjonering**  
Poenget her er å skape/vedlikeholde et marked for proprietær (lukket, ikke-åpen) programvare, f.eks. ved å tilby klient-programvare som åpen kildekode, men som krever at det finnes en proprietær server-programvare som må kjøpes.
- **Utstyrsavhengighet**  
Programvare som er knyttet til spesielt utstyr (hardware), f.eks. drivere, leveres som åpen kildekode, men man tar seg godt betalt for selve utstyret.
- **"Gi fra deg oppskrifta, åpne en restaurant"**  
Gjøre fortjeneste på å yte tjenester knyttet opp mot åpen kildekode, f.eks. installasjon, tilpassing, support. (Eksempel er Red Hat og andre distributører av Linux.)
- **Tilbehør**  
Gjøre fortjeneste på alt fra kaffekrus og t-skjorter – til dokumentasjon og brukerveiledninger.
- **"Free the future, sell the present"**  
Salgsavtale med en klausul som sier at etter X antall år, eller ved en konkurs eller under andre gitte omstendigheter gjøres koden fritt tilgjengelig (går over til GPL).
- **"Free the software, sell the brand"**  
Spekulativ modell: Selge en garanti, et stempel, som sier at denne programvaren er kompatibel med et visst annet merke.
- **"Free the software, sell the content"**  
Enda en spekulativ modell: Ta seg betalt for det som kommer ut av programvaren, f.eks. adresselister.

Men Richard Stallman liker tydeligvis ikke denne måten å tjene penger på [10]. Han sier disse løsningene kom med utbredelsen av begrepet Open Source, og at de som gjør forretning på disse måtene ikke bruker betegnelsen Free Software.

### 3 Åpen kildekode og informasjonssikkerhet

Diskusjonene om sikkerheten i åpen kontra lukket programvare er mange. Vi vil her forsøke å oppsummere de viktigste aspektene.

Som beskrevet i avsnitt 1.1.1 omfatter informasjonssikkerhet begrepene konfidensialitet, integritet, tilgjengelighet og kvalitet. Trusler mot disse er bl.a. at informasjon blir tilgjengelig for uvedkommende (noe som f.eks. kan medføre brudd på taushetsplikten for helsepersonell og helsevirksomheter), at informasjon blir endret på en urettmessig og uautorisert måte (kan medføre at pasienter ikke får riktig eller optimal behandling), at informasjon ikke er tilgjengelig for de som har rettmessig krav på den når de trenger den og at kvaliteten på informasjonen ikke er tilfredsstillende (f.eks. bilde- eller lyd-kvalitet).

Vi nevner først noen generelle trusler mot informasjonssikkerhet.

Trusler mot *konfidensialitet*, dvs. måter uvedkommende kan få tilgang til opplysninger:

- Uvedkommende får tilgang til å søke direkte i systemet etter opplysninger.
- Opplysninger formidles elektronisk til uvedkommende (målrettet eller tilfeldig).
- Ondsinnet programvare i form av f.eks. virus eller trojanske hester brukes til å formidle opplysninger til uvedkommende.

Trusler mot opplysningenes *integritet*, dvs. måter informasjon kan bli endret utrettmessig på:

- Uvedkommende får tilgang til dataene og kan endre dem.
- Ondsinnet kode endrer dataene.
- Feil i programvaren forårsaker (utilsiktede) endringer.
- Feil i programvaren kan også være en indirekte årsak, ved at endringer blir mulig-gjort, f.eks. at dokumenter som er signert av brukeren likevel lar seg endre.

Trusler mot *tilgjengelighet* av opplysninger:

- Dokumenter eller informasjon blir slettet eller ødelagt pga. ondsinnet kode eller brukerfeil.
- Dårlig brukergrensesnitt forårsaker at brukere ikke får tilgang til nødvendig informasjon.
- Feil i programvaren forårsaker at system eller informasjon er utilgjengelig.
- Feil ved oppgraderinger av programvaren eller manglende oppgraderinger (ingen tar ansvar for å lage oppgraderinger).
- Overbelastning ved Dos-angrep ("Denial of service"-angrep).
- Server-crash og mangelfull backup-løsning.
- Ytre årsaker, f.eks. strømbrytning, som det ikke er tatt høyde for.
- Systemet er utilgjengelig uten at man kjenner årsaken.

Trusler mot opplysningenes *kvalitet*, dvs. årsaker til at kvaliteten blir utilstrekkelig:

- Båndbredden er ustabil, spesielt ved lyd- og bildeoverføring.
- Dårlig programvare gir misvisende resultater (feil, eller bare dårlige algoritmer).
- Feil i integrasjon mellom system (uklare grensesnittspesifikasjoner) kan f.eks. føre til feilregistrering av data.

Som vi ser av kulepunktene over, kan det være ulike årsaker til brudd på informasjonssikkerheten. Mangelfulle organisatoriske prosedyrer, dårlige rutiner og mangelfull opplæring av de ansatte utgjør ofte den største trusselen mot informasjonssikkerheten i en virksomhet, i kombinasjon med feilkonfigureringer av systemene. Feil og sårbarheter i programvare er også årsak til en vesentlig andel av de potensielle sikkerhetstruslene. Vi gjør ikke noe stort

skille her mellom mer generelle feil i programvaren kontra spesielle sikkerhetsfeil. Mange typer generelle feil kan føre til sårbarheter i et system, f.eks. "buffer overflow". Mer spesielle sikkerhetsfeil, som f.eks. feil i tilgangskontrollsystem, utgjør andre sårbarheter. Konsekvensen kan være like alvorlig i begge tilfelle.

God kvalitet på programvaren reduserer alle typer feil og er derfor avgjørende for å redusere antall sikkerhetsbrudd. Kvaliteten på programvaren er både avhengig av utviklingsrutiner og prosedyrer, og ikke minst av testingen av programvaren før og etter at den slippes på markedet.

### 3.1 Sammenhengen mellom kvalitet og sikkerhet i programvare

Sikkerheten i et system er i stor grad avhengig av hvilket fokus som har vært satt på sikkerhet ved design og utvikling av systemet. Det er stor forskjell på å kun ha fokus på funksjonalitet i første omgang, og så skulle gjøre systemet sikkert etter at det viser seg å ha tilfredsstillende funksjonalitet – i forhold til å tenke på sikkerhetsfunksjonalitet som en integrert del av systemet allerede fra designstadiet. Ved å ha fokus på sikkerhet fra tidlig i designfasen og gjennom implementasjonsfasen, er det også mer sannsynlig at testingen i større grad vil omfatte sikkerheten i systemet.

Alle dataprogram, enten de er gitt ut med åpen eller lukket kildekode, inneholder feil og svakheter. Noen av disse utgjør sikkerhetstrusler. En av årsakene til at program kommer på markedet med feil og svakheter, er at leverandøren ønsker å komme konkurrentene i forkjøpet, og derfor ikke tar seg tid til å teste programvaren godt nok før den lanseres. Payne [14] mener at ett av problemene med å utvikle sikker kode er at proprietære leverandører har sterke økonomiske incentiver for *ikke* å gjøre dette. Det tar lengre tid og koster mer å utvikle sikker programvare, man kommer senere på markedet enn konkurrentene og mister markedsandeler. I tillegg blir programvaren dyrere for kunden, som ikke klarer å skille mellom sikre og mindre sikre program og gjerne velger det billigste alternativet. Andre årsaker kan være slurv, at utviklerne og/eller testerne ikke har god nok fokus på alle typer feil og svakheter, at de mangler testmiljø for en del typer feil, at de mangler nødvendig kompetanse, etc.

Hoepman et al [15] mener at man må kunne anta at et minimumsnivå når det gjelder standard utviklingspraksis, prosjektledelse, endringshåndtering og kvalitetskontroll blir ivaretatt i enhver utviklingsprosess. Ved å åpne kildekoden blir det vanskeligere for utviklingsprosjekt å skjule dårlig prosjektledelse og dårlig kvalitetskontroll. Åpen kildekode-utvikling tvinger utviklingsmiljøene til å være mer påpasselig, og til å bruke de beste tilgjengelige verktøyene for å sikre sine system. Det tvinger dem også til å skrive enklere og klarere kode og følge standarder, og til å legge mer arbeid i kvalitetskontrollen. Man har likevel ingen garanti for at kvaliteten på åpen kildekode er bedre enn lukket kode.

Et initiativ for kvalitetssikring av åpen kildekode gjøres i OpenBRR – Business Readiness Rating<sup>28</sup>. Hensikten med dette initiativet har vært å vurdere åpen kildekode for å se om den er "moden" nok til å tas i bruk. Ideen er god, men vi er usikre på om dette er et forum som fortsatt er i drift, da det ikke har vært synlig aktivitet på forumet de siste par år. – Det ville være nyttig å ha en organisasjon og et nettsted for vurdering av kvaliteten av programvare i forhold til bruksområde, her helsesektoren, slik det er foreslått i en prosjektoppgave fra UiT i 2006 (se avsn. 4.4.2 i [20]).

---

<sup>28</sup> [www.openbrr.org](http://www.openbrr.org)

## 3.2 Argument for og mot åpen kildekode

Mye litteratur tar opp sikkerhetsproblematikk i åpen kildekode. I de følgende avsnittene har vi sortert argumentene for og mot åpen kildekode ut fra ulike synsvinkler.

### 3.2.1 Egen evaluering av sikkerheten

Mange av forkjemperne for åpen kildekode mener at en større andel av feilene i programvaren vil oppdages og rettes langt raskere dersom kildekoden er tilgjengelig slik at mange kan undersøke den og foreslå rettinger. Dette er av Eric Raymond [6] kalt Linus's Law: *"Given many eyeballs, all bugs are shallow."*

Hoepman et al [15] mener bruk av åpen kildekode er et nødvendig krav for å bygge sikrere system. Å åpne kildekoden tillater uavhengig bedømmelse av hvor utsatt eller sårbart et system er og hvor stor risiko som er forbundet med å bruke systemet. Brukerne kan selv evaluere sikkerheten, eller det kan etableres flere uavhengige team som evaluerer sikkerheten i systemet. På denne måten unngår man avhengighet mellom de som evaluerer koden og de som i utgangspunktet har utviklet den.

Et steg i riktig retning er å gjøre systemdesignet åpent. Det er imidlertid forskjell på å offentliggjøre designet av systemet og å offentliggjøre den implementerte koden. Dersom man kjenner designet av systemet, kan man vite noe om logikken bak koden, men man har ingen garanti for at koden er implementert i henhold til designet [15].

Payne [14] trekker fram et annet moment: De som forsøker å finne svakheter i proprietær programvare er som regel ute etter å utnytte svakhetene (uten å rapportere dem), mens de som forsøker å finne svakheter i åpen kode som regel gjør det fordi de ønsker å forbedre koden. Han påstår også at mange sikkerhetsekspertene har slått fast at åpen kildekode har fordeler framfor proprietær programvare med hensyn til sikkerhet.

### 3.2.2 Raskere feilretting

Payne [14] poengterer at fleksibiliteten og friheten i åpen kildekode gjør det mulig for organisasjonene selv å undersøke koden, modifisere den for å tilfredsstille egne sikkerhetsbehov og raskt oppdatere sine systemer uten å måtte vente på leverandøren.

Hvis de i tillegg publiserer sikkerhetsoppdateringer på et sentralt, offentlig tilgjengelig nettsted, kan alle andre oppdatere sine systemer med disse og forbedre sine versjoner av systemet. Erfaring viser at oppdateringer for åpen kildekode frigis nesten dobbelt så ofte som oppdateringer for lukket kode. Dette medfører en halveringstid for sårbarhetene.

Alvorlige feil i Linux som blir rapportert inn, vil bli rettet raskt, mens samme type feil kan forbli uoppdaget og urettet i Windows i månedsvis eller årevis [8]. En grunn til at kommersielle selskaper ønsker at sårbarheter forblir ukjent, kan være at de ønsker å unngå negativ publisitet som kan sette produktet og selskapet i vanry.

Witten et al [16] presenterer et diagram som viser hvordan tiden det tar å finne sikkerhetshull avtar (eksponentielt) med økende antall personer som ser på koden – selv om det er personer som ikke er like mye eksperter som de få som blir betalt for å gjøre jobben.

Dersom flere (beta-)testere får tilgang til kildekoden vil det være flere feil som oppdages, rapporteres og må rettes. Det virker som om store leverandører venter med å komme med nye oppdateringer til flere har rapportert samme feil. Et eksempel er IBM som ventet til åttende gang en og samme feil ble rapportert i deres operativsystem for stormaskiner før de retta feilen [17].

### 3.2.3 Åpen kildekode gir liten garanti for systematisk gjennomgang av kode

Andre er imidlertid av den mening at det å åpne kildekoden ikke gir noen garanti for at den blir gjennomgått systematisk med tanke på å finne feil og svakheter.

Ross Anderson [17] påpeker at utviklere av sikkerhetskritisk programvare i mange år har visst at for at et stort, komplekst system skal ha "mean time before failure" (MTBF) på f.eks. 100 000 timer, må det underkastes minst like mange timer med testing.

Anderson sier videre at ettersom testing er kjedelig, og frivillige vanligvis kun ønsker å fikse feil som irriterer dem, vil mengden med konsentrert oppmerksomhet som vilkårlige utviklere gir et system, sannsynligvis ikke kunne måle seg med innsatsen fra dem som virkelig er ute etter å finne og utnytte sårbarheter.

John Viega [13] hevder at mesteparten av programvaren aldri blir undersøkt med hensyn på sikkerhet, enten det er åpen kildekode eller ikke. Hvis den blir det så er det en tendens til at kommersiell programvare får mye mer kvalifisert oppmerksomhet, ganske enkelt fordi de som må gjøre den kjedelige og omstendelige jobben får betalt for det. Han påpeker også at kunden ikke betaler for programvaresikkerhet. De forventer at det de kjøper eller får har nødvendig sikkerhet innebygd.

Viega sier videre at når det gjelder "the many eyeballs"-argumentet, er det tvilsomt om disse ser spesielt etter sikkerhet og sårbarhet. Det er ikke sikkert det er de riktige "øynene" som undersøker koden. De har ikke nødvendigvis sikkerhetseksperise. De er ute etter funksjonaliteten og foretar som regel ingen grundig gjennomgang av hele programmet. Og hvis de ser etter sikkerhetsfeil og sårbarheter, leter de etter de store og mest innlysende, ikke etter de små detaljene. De gjør det med andre ord ikke på en strukturert måte.

I de fleste tilfelle blir nok programvare kjøpt, installert og satt i produksjon uten en grundig gjennomgang av sikkerheten, uavhengig av om det er åpen eller lukket kildekode [14]. I så måte er produkt fra kommersielle leverandører bedre stilt, for disse har de nødvendige testverktøy og -omgivelser som brukere og kunder vanligvis ikke har tilgjengelig. Det er neppe noen som gjennomgår (åpen) kildekode for artighet. Man kan ikke garantere at slik sikkerhetsgjennomgang blir gjort av tilfeldige frivillige [16]. Gjennomgang, forbedring og oppgradering av kildekode blir nok oftest gjort i forbindelse med at "noen" skal bruke koden til "noe", og disse "noen" er vanligvis ansatt i en bedrift/organisasjon der de får betalt for å gjøre jobben.

Overly [4] mener at etter hvert som et program utvikles og blir mer komplisert, og mange utviklere fra ulike miljø har vært inne med sine bidrag, blir det vanskeligere å rette opp sikkerhetsfeil eller legge til nye sikkerhetsløsninger. "Many eyeballs" vil gjerne føre til mange oppdateringer, som igjen kan resultere i ustabil kode. Med mindre noen eksplisitt har påtatt seg ansvaret for å sjekke kvaliteten i programmet i sin helhet, er det lite sannsynlig at dette vil bli gjort.

Det finnes verktøy for kvalitetssjekking av kode, hovedsakelig i kommersielle miljø. Når slike verktøy blir mer vanlig også i åpen kildekode-miljø vil det være med på å styrke "many eyeballs"-argumentet [13], da blir det enklere å sjekke koden for de som ønsker det, ved at verktøyene hjelper til med å peke ut svakhetene.

I USA har regjeringen<sup>29</sup> finansiert et program på over 1 mill. US\$ for å finne og rette sikkerhetsfeil i åpen kildekode. Programmet har fått navnet "The Open Source Hardening

---

<sup>29</sup> Department of Homeland Security, DHS, [www.dhs.gov](http://www.dhs.gov)

Project". Arbeidet utføres av Stanford University<sup>30</sup> sammen med sikkerhetsselskapene Coverity<sup>31</sup> og Symantec<sup>32</sup>. Ved inngangen til 2008 ble det rapportert at de i løpet av to år hadde analysert 50 millioner kodelinjer i mer enn 250 prosjekt og fått rettet opp nærmere 8000 sikkerhetsfeil. I gjennomsnitt har de funnet én sikkerhetsfeil pr. 1000 kodelinjer [19].

### 3.2.4 Eksponering av sårbarheter

Når kildekoden gjøres tilgjengelig får også *angripere* tilgang. Payne [14] spør: Hvis det finnes en sårbarhet i et program, men ingen kjenner til denne, utgjør den da noen trussel? Tilhengere av lukket kildekode kan argumentere med at dersom det i lukket programvare finnes feil og svakheter som aldri blir oppdaget, er det på linje med at sårbarhetene ikke finnes, ikke utgjør noe sikkerhetsproblem. Men etter vår mening kan også ukjente sårbarheter medføre sikkerhetsbrudd, f.eks. ved at noen utilsiktet forårsaker buffer-overflow pga. ukjente feil i koden.

Å gjøre kildekoden tilgjengelig gir angriperne en fordel fordi de får mer informasjon som gjør dem i stand til å finne svakheter og benytte dem til å angripe systemet. Hoepman et al [15] lister opp noen typiske argument for at åpen kildekode gir potensielle angripere en fordel:

- Ved å holde kildekoden skjult forhindrer man at angriperen får lett tilgang til informasjon som kan være nyttig i forbindelse med et angrep.
- Offentliggjøring av *designet* av et system kan avsløre alvorlige logiske feil. (Hvis designet offentliggjøres er det mulig for andre å avsløre feil og svakheter i designet. Dette er kanskje enklere enn å avsløre feil i koden).
- Offentliggjøring av koden gir angriperen en stor fordel – han trenger kun å finne én sårbarhet for å kunne gjennomføre et angrep, mens utvikleren/forsvareren må patche alle sårbarheter for å forsvare seg fullstendig.
- Det å offentliggjøre koden er ingen garanti for at kvalifiserte personer vil evaluere koden, finne svakheter og eventuelt komme med forbedringer. Angripere som finner svakheter derimot, vil høyst sannsynlig angripe koden.

Argumentene foran baserer seg på at lukket kode faktisk inneholder sårbarheter.

Men både Wheeler [3], Hoepman et al [15] og Witten et al [16] er opptatt av at angriperne uansett har tilgang til den kjørbare versjonen av koden, og at de ofte kan rekonstruere kildekoden utfra denne. De mener dessuten at å klare å holde kildekoden lukket over lang tid viser seg å være vanskelig. Og selv om koden forblir lukket, vil sårbarheter med tiden bli avslørt for et større publikum. Det eksisterer verktøy for feilsøking og for å plukke koden fra hverandre (disassemble) slik at angripere relativt raskt kan finne sårbarheter i systemene uten å ha tilgang til kildekoden. Kompilering er *ikke* en måte å kryptere kildekoden på, og er derfor ikke til hjelp for å hindre angrep. Angripere kan velge å holde kunnskapen om sårbarhetene skjult, slik at de kan utnytte dem uten at noen vet om det og uten at noen får mulighet til å patche dem. Og hvis de offentliggjør sårbarhetene, er det kun den som har produsert koden som kan rette opp sårbarhetene. Witten et al [16] mener at lukking av kildekode har en økonomisk og ikke en sikkerhetsmessig hensikt.

I krypto-verdenen har det vært vanlig praksis siden slutten av 1800-tallet at design av system eller algoritme er offentlig kjent. Den måten man sikrer systemet på er å holde krypterings-

---

<sup>30</sup> [www.stanford.edu](http://www.stanford.edu)

<sup>31</sup> [www.coverity.com](http://www.coverity.com)

<sup>32</sup> [www.symantec.com](http://www.symantec.com)

nøkkelen hemmelig. Dette prinsippet ble først nedsatt i 1883 av Auguste Kerckhoffs: ”I et vel-designet kryptografisk system er det bare nøkkelen som må holdes hemmelig, det burde ikke være noe hemmelighold av algoritmen”.<sup>33</sup> Ross Anderson [17] regner dette som starten på debatten omkring åpne vs. lukkede system.

Også Overly [4] mener at mange ikke vil trenge kildekoden for å finne sårbarheter. Et stort antall sikkerhetsproblem har sin opprinnelse i noen få typiske programmeringsfeil, f.eks. buffer-overflow.

Som Hoepman et al [15] påpeker, vil det å åpne opp kildekoden ikke endre sikkerheten til et system, fordi det å åpne opp i seg selv ikke innfører flere feil eller svakheter i koden. Men det å åpne opp kildekoden kan på kort sikt øke eksponeringen og gjøre systemet mer utsatt fordi det synliggjør eksisterende feil og svakheter – sårbarheten og truslene blir offentlig kjent. Perioden med økt sårbarhet kan imidlertid bli kort, og brukerne har mulighet til å vurdere sikkerheten ved systemene de skal velge blant. Åpen kildekode gjør det lettere for andre å evaluere systemet og bidra til forbedringer slik at systemet på lengre sikt blir bedre og sikrere.

### 3.2.5 Planting av ondsinnet kode

Perens [8] spør: Hva med Trojanske hester? Hva om noen modifierer en åpen kildekode ved å legge inn trojanske hester – og så, når programvaren har fått en viss utbredelse, utnytter denne svakheten? På denne måten vil folk kunne oppfatte åpen kildekode som mer sårbar enn f.eks. Microsoft.

I system som utvikles etter bazaar-metoden kan angripere snike inn bakdører eller lignende i kode de offentliggjør [15]. Dette er kanskje avhengig av prosjektet og styringen med hvem som får lov til å legge ut ny kode. Sikring av åpen kildekode forutsetter god kontroll med og identifikasjon av hvem det er som legger ut ny kode eller endringer [8]. De fleste åpen kildekode-prosjekt bruker i dag versjonskontrollsystem (f.eks. CVS) som sporer alle endringer [14]. Når man vet hvem det er som har lagt ut en Trojansk hest, kan man bruke straffelovgivningen mot dem [8]. Større organisasjoner, f.eks. Linux, lar ikke hvem-som-helst publisere nye versjoner. Endringer og oppdateringer gis til en liten kjernegruppe som validerer dem før de legges ut.

Proprietær programvare har imidlertid også sin andel av slike feil og sårbarheter, selv om utenforstående ikke har anledning til å legge inn kode i disse produktene. Payne [14] mener at et av de største sikkerhetsproblemene knyttet til *proprietær* programvare er at hvis noen bevisst planter ondsinnet kode i programmet, kan det være svært vanskelig å oppdage dette. Testene leverandører kjører har til hensikt å oppdage programmeringsfeil o.l., og ikke ondsinnet programvare. Det er også sjelden at andre enn den som har utviklet koden analyserer koden på et detaljert nivå, slik det ofte blir gjort med åpen kildekode. Ondsinnet programvare som blir plantet i åpen kildekode har en langt større sannsynlighet for å bli avslørt.

Uten tilgang til kildekoden stoler man ikke bare på kildekoden, men også på kompilatoren som laget den kjørbare versjonen. I 1984 skrev Ken Thompson en berømt artikkel med tittel ”Reflections on Trusting Trust” [18] der han illustrerer hvordan kompilatoren kan endre den kjørbare versjonen av et program uten at endringen vises i kildekoden. Men har man kildekoden kan man i det minste velge kompilator selv, og til og med kompilere med flere kompilatorer og sammenligne resultatet. Tilgang til kildekoden gjør at man dermed kan oppdage både tilfeldige og plantede feil.

---

<sup>33</sup> Vår oversettelse fra <http://www.schneier.com/crypto-gram-0205.html#1>

Når EU ber sine medlemsland om å fremme bruken av åpen kildekode bruker de sikkerhet som ett argument: "Guaranteeing that no backdoors are built into programmes." [3]. I 2001 ba Europaparlamentet EU-kommisjonen om å angi en standard for sikkerhetsnivået i e-post-system, og rangere som minst pålitelige de systemene som ikke har åpen kildekode<sup>34</sup>.

### 3.2.6 Mulighet for lokale tilpasninger

Hoepman et al [15] lister opp en del fordeler åpen kildekode gir med tanke på å gjøre lokale tilpasninger:

- Åpen kildekode gjør det mulig for brukeren å legge til ekstra sikkerhet.
- Åpen kildekode gjør det mulig for brukeren å redusere kompleksiteten i systemet ved å fjerne unødvendige deler av systemet. Dette kan også ha sikkerhetsmessige fordeler.

Og vi kan legge til: Åpen kildekode gjør det mye enklere for brukeren å tilpasse koden til andre system eller program den skal integreres med.

## 3.3 Sammenligning av sikkerhet og pålitelighet i lukket kontra åpen kildekode

Dare Obasanjo [12] sier at i alle mulige undersøkelser sammenlignes sikkerheten i åpen kildekode med sikkerheten i Microsoft. F.eks. sammenligner man åpne Unix-varianter med Windows. Han mener det blir som å sammenligne "hummer og kanari". Windows har en helt annen utbredelse enn Unix, både i mengde, anvendelse og brukergrupper. – Vi tror likevel at de sammenligningene som har vært gjort gir gode indikasjoner på forholdet mellom sårbarheter i lukket kontra åpen kode.

Noen av de kvantitative studiene har bl.a. konkludert med at antallet "hackede" web-sider er størst for de som har Windows som operativsystem [3]. De har også konkludert med at Linux varer lenger uten patching enn Windows, at alle operativsystem med åpen kildekode har færre sårbarheter enn Windows, at Windows har større andel *kritiske* sårbarheter enn (Red Hat) Linux, at virusangrep er langt mer framtrepende på Windows enn på noe annet operativsystem (muligens pga. den store utbredelsen av Windows, noe som gjør det mer attraktivt å lage virus for Windows for å oppnå raskere smitteeffekt), at 80 % av all spam er sendt ut fra infiserte Windows-pc'er (2004), at Mozilla fikser sine sårbarheter mye raskere enn Internet Explorer (IE) (2004) og at leverandører av åpen kildekode reagerer 60 % raskere på sårbarhetsrapporter.

Wheeler [3] viser til at i løpet av hele 2004 var det bare sju dager at IE var uten sårbarheter, IE var sårbar for angrep i 98 % av det året, mens Opera var sårbar 17 % av året og Mozilla 15 %. Typen sårbarhet som ble undersøkt var "Remote code execution". For IE var det dette året (2004) kun én sammenhengende uke i oktober at nettleseren ikke var sårbar for slike angrep.

For 2006 fant man følgende tall: IE var sårbar for angrep i 78 % av året, Mozilla i 2 %. I flere tilfelle var angrepene fra organiserte kriminelle mot sårbarhetene i IE så alvorlige og Microsoft så sen med å produsere oppdateringer at tredjeparts sikkerhetspatcher ble lansert. Mange anbefalte å installere disse oppdateringene. For flere detaljer om de kvantitative studiene viser vi til Wheeler [3].

---

<sup>34</sup> [http://www.cyber-rights.org/interception/echelon/European\\_parliament\\_resolution.htm](http://www.cyber-rights.org/interception/echelon/European_parliament_resolution.htm) - tiltak 30 - 31

Payne [14] har gjort undersøkelser av antall sikkerhetsmekanismer og sårbarheter i tre ulike system: Debian Linux, Solaris (Unix) og OpenBSD (Unix). OpenBSD hadde flest sikkerhetsmekanismer, og minst antall rapporterte sårbarheter. Dette viser at det å skjule koden ikke reduserer antall rapporterte sårbarheter. Det proprietære systemet (Solaris) hadde flest rapporterte sårbarheter. – Men utviklerne av OpenBSD sier selv, i følge Payne [14], at det ikke er det at kildekoden er åpen som gjør OpenBSD sikkert, men det faktum at de har en liten gruppe programmerere som er spesialisert på å finne og rette sikkerhetsfeil.

### 3.4 Oppsummering

Som Wheeler [11] påpeker, er spørsmålet om åpen kildekode bidrar til større sikkerhet fortsatt et viktig debatt-tema. Det er mange øyne som ser, men vil folk virkelig gjennomgå kildekoden? Han antar at det vil være få som gjennomgår kildekoden dersom kildekoden er laget for et nisjeprodukt eller nisjemarked, dersom koden har veldig få utviklere (kanskje bare en), eller dersom det f.eks. brukes et lite utbredt programmeringsspråk.

Tilgang til kildekoden gir i seg selv ingen garanti for at kvaliteten og sikkerheten blir gjennomgått på en systematisk måte. De fleste som gjennomgår koden er bare ute etter funksjonalitet, og ser ikke spesielt etter sikkerhetsfeil og sårbarheter. Det må også være de rette øyne som ser, de må vite hva de ser etter og de må vite hvordan man lager sikker kode.

Åpen kildekode gjør det *mulig* for brukere av koden å selv evaluere sikkerheten og påliteligheten i programvaren, eller få andre uavhengige aktører til å evaluere den. Tilgang til kildekoden gir dermed i prinsippet mulighet for raskere feilretting og fjerning av sårbarheter. Åpen kildekode gjør det enklere med lokale tilpasninger, men det er også viktig at rettinger distribueres raskt og ikke bare gjøres lokalt, slik at nye versjoner av kildekoden får med de samme rettingene.

Payne [14] påpeker at de som vil holde kildekoden *lukket* og de som taler for å *åpne* den egentlig bruker samme argument: det å åpne kildekoden gjør at feil og sårbarheter oppdages lettere. De første ønsker imidlertid at sårbarhetene skal forbli ukjente, mens de andre ønsker å få dem avdekket og rettet. Og Wheeler [11] fremholder at det å hemmeligholde sårbarheter og svakheter ikke gjør at de forsvinner.

Som vi ser, er det ingen entydige konklusjoner når det gjelder om åpen kildekode er mer sikker enn lukket kode.

## 4 Åpen kildekode i helsesektoren

Også i helsesektoren er det naturlig å skille mellom "horisontale system" og "vertikale system" (se avsnitt 2.3). De vertikale systemene, som vi i helsesektoren kan kalle "medisinsk programvare", vil her bestå av både store fellessystem for helsesektoren, som pasient-administrative system og pasientjournalssystem, men også av spesialapplikasjoner for de ulike fagområdene.

Programvare i helsesektoren er veldig fragmentert. For mange delsystem finnes det løsninger fra forskjellige leverandører, og mye av det medisinsk-tekniske utstyret har sin egen programvare som er tett koplet til maskinvaren.

I dette kapitlet gir vi først noen få eksempler på bruk av åpen kildekode i helsesektoren internasjonalt. Deretter presenterer vi en oppsummering av intervju og samtaler med ulike aktører i helsesektoren i Norge om bruk av og holdninger til åpen kildekode.

### 4.1 Åpen kildekode i helsesektoren – internasjonalt

I dette avsnittet gir vi først en kortfattet presentasjon av en del pågående internasjonale prosjekt med åpen kildekode i helsesektoren. Deretter presenterer vi i mer detalj to større satsninger: Et sykehus i Dublin, Irland som gikk over til å benytte åpen kildekode i stor utstrekning; og en intervjuundersøkelse fra Quebec, Canada der de valgte å *ikke* gå over til åpen kildekode i helsesektoren.

#### 4.1.1 Noen internasjonale prosjekt og løsninger

De systemene vi presenterer her er i kategorien "vertikale system", dvs. applikasjoner som er spesifikke for helsesektoren. Flesteparten av disse er system for elektronisk pasientjournal (EPJ), eller de kategoriserer seg som fulle "Clinical Management Systems" (CMS). I det følgende omtaler vi noen av systemene. En mer omfattende liste er tatt med i vedlegg C.

Det kanskje mest kjente initiativet her i Europa er **GEHR** (Good European Health Record)<sup>35</sup>. Dette var et europeisk prosjekt på 1990-tallet, som laget en arkitektur for et elektronisk journalssystem. Prosjektet har bidratt til standardiseringen av EPJ i Europa, innenfor CEN (Comité Européen de Normalisation). Dette prosjektet, samt det australske GEHR-prosjektet (Good Electronic Health Record) videreføres internasjonalt (i CEN og ISO) gjennom **OpenEHR** (<http://www.openehr.org/>).

**OpenEMed** (<http://openmed.sourceforge.net/>) var ikke ment å være et journalssystem, men en plattform for utvikling av helsesystem, kanskje spesielt for sykehus og større virksomheter, en plattform som man også kan bygge EPJ utfra. Programvaren er BSD-lisensiert. (Prosjektet var i sin tid ganske aktivt, men er nå i stillstand, og web-sidene har ikke vært oppdatert siden 2005.)

**OpenHRE** (<http://www.openhre.org/>) er et amerikansk initiativ som vil fremme programvare for utveksling av informasjon mellom pasientjournaler. (Også dette ser ut til å være et passivt prosjekt, med liten eller ingen aktivitet siden tidlig i 2006.)

I det følgende omtaler vi noen system i kategorien elektroniske pasientjournaler.

---

<sup>35</sup> <http://www.chime.ucl.ac.uk/work-areas/ehrs/GEHR/index.htm>

**MirrorMed** (<http://www.mirrormed.org/>) er et av de interessante prosjektene rettet mot primærhelsetjenesten. Systemet er lisensiert under GPL. Det er bygd på programvare fra to andre prosjekt, **FreeMED** (<http://www.freemed.org/>) og **OpenEMR** (<http://www.oemr.org/>), og ble også igangsatt av utviklere fra de to prosjektene. Alle disse prosjektene er fortsatt aktive, og OpenEMR sies å være et av de mest populære EPJ-systemene med åpen kildekode, med nærmere 35.000 nedlastninger fra SourceForge.

**OSCAR** (Open Source Clinical Applications & Resources) (<http://www.oscarcanada.org/>). Dette er et kanadisk prosjekt som omtaler sitt system som et fullt CMS-system. De har også en "personlig journal" for pasienter, tilknyttet sitt system: MyOSCAR (<http://myoscar.org/>).

**PatientOS** (<http://www.patientos.org/>). Dette er et amerikansk system under utvikling, lisensiert med GPL v.3. (Telemedisin-studenter i Tromsø har brukt dette i noen prosjekt og vurderer at det er et system som er lett å videreutvikle.)

**GNUmed** (<http://wiki.gnumed.de/bin/view/Gnumed>). Dette er et internasjonalt prosjekt som primærleger og utviklere har tatt initiativ til. Systemet er under stadig utvikling, og det prøves ut blant primærleger og fysioterapeuter, spesielt i Tyskland.

Et system som ofte blir trukket fram er **VistA** (<http://www.hardhats.org/>). Mens flere av de øvrige systemene retter seg mot primærhelsetjenesten, er dette i første rekke et system for sykehus og andre større virksomheter. Det ble utviklet av U.S. Department of Veterans Affairs for bruk i deres sykehus, poliklinikker og sykehjem. Dette er et av de eldste av denne typen system, arbeidet ble igangsatt for over tjue år siden. Den opprinnelige programvaren var lisensfri ("public domain"), men senere versjoner er lisensiert under ulike åpen kildekode-lisenser. Kommentarer tyder på at man må ha en del detaljkunnskap for å sette opp, drifte og vedlikeholde dette systemet. De bruker bl.a. et lite utbredt programmeringsspråk som heter M<sup>36</sup>. WorldVistA (<http://worldvista.org/>), ble etablert som egen organisasjon i 2002, for å støtte en større utbredelse av programmet, og systemet WorldVistA, som er lisensiert under GPL, ble lansert som et eget EPJ-system i 2007.

I tillegg finnes det en mengde system innen spesifikke fagområder, f.eks. programvare for radiologi og annet bildedannende utstyr. Det finnes også flere åpen kildekode-system som retter seg mot pasienten og pasientens tilgang til helsesektoren, såkalte PHR-system (Personal Health Record). Noen av disse systemene er tatt med i lista i vedlegg C.

#### 4.1.2 *Hvorfor åpen kildekode ved Beaumont hospital*

Beaumont hospital i Dublin, Irland [21], er et større universitetssykehus, et resultat av sammenslåing av tre mindre sykehus i 1987. I 2003 var de ca. 3000 ansatte. IT-avdelingen hadde ansvar for et heterogent miljø av utstyr og programvare, inkludert 36 servere, hvorav 22 kjørte Linux og 14 Microsoft Windows NT. Dertil kom de medisinske systemene på HP stormaskin, og de økonomiske systemene som også ble kjørt under Unix på HP. Brukerne hadde tilgang til om lag 1000 pc-er, en tredel av dem var "utdaterte".

I 2003 hadde dette sykehuset et underskudd på 17 M€ (ca 135 MNOK).

Deres største drivkraft for overgang til åpen kildekode var *kostnadsreduksjon*. De ønsket seg "null kostnad, eller så billig som mulig." De poengterer at de ikke var styrt av noen "anti-Microsoft"-ideologi. Microsoft hadde allerede kommet dem i møte ved å gi dem "akademiske

---

<sup>36</sup> <http://www.webopedia.com/TERM/M/MUMPS.html>

priser". Og samtidig som de tok i bruk åpen kildekode, valgte de Microsoft-løsninger for de medisinske systemene i en av sykehusets avdelinger.

Tilgang til kildekode var i utgangspunktet *ikke* noe poeng for Beaumont. De ser ikke for seg at de vil gjøre endringer i kildekode (selv om de ved et tilfelle hadde gjort en liten 5-linjers tilpassing i Linux og kompilert kjernen på nytt).

De tok i bruk et vidt spekter av åpen kildekode, fra usynlige infrastruktur-system som Linux, til mer synlige desktop-system som Office og e-post. I 2002 gikk de over til **Star Office** som desktop-system. Brukerne kunne velge å beholde proprietære løsninger, men de ville da ikke få support fra IT-avdelingen, de måtte drifte det selv. 8 % av brukerne valgte å gjøre det. Sykehuset bruker **Zope** som Content Management System (CMS), der de betaler for support til et lite firma som har spesialisert seg på å forhandle/supportere åpen kildekode-løsninger. De har også valgt åpne løsninger for **e-læring**.

IT-avdelingen ved Beaumont utviklet en egen løsning for å finne fram og vise digitale røntgenbilder, som de benytter i kombinasjon med HW for bildearkivering de har fått kostnadsfritt fra Sun. Samtidig brukte et annet sykehus i Irland 4.3 M€ (35 MNOK) på en kommersiell PACS-løsning, en løsning som var adskilt fra øvrige system slik at brukerne måtte ha to skjermer. Beaumont sin egenutviklede løsning var integrert i andre system slik at de kunne få opp bilde og data på samme skjerm.

Beaumont har utviklet flere applikasjoner som de nå tilbyr som åpen kildekode til andre innen helsevesenet. Eksempler på dette er et vaktlistesystem, et system for sammenligning av vevsprøver og et akuttssystem.

Beaumonts overgang til åpen kildekode var styrt av "De tre P-er": Prinsipp. Pragmatisme. Praksis.

**Prinsipp:** Få mest igjen for skattebetalernes penger. De oppnådde kostnadsbesparelser på 13 M€ i løpet av 5 år (over 100 MNOK). – Og sykehuset hadde i utgangspunktet gode rabatter (såkalte akademiske priser) på proprietær programvare.

**Pragmatisme:** Utgangspunktet var store reduksjoner i IT-budsjettet. De hadde valget mellom enten å redusere service eller å se etter alternativer. IT-ledelsen var overbevist om at åpen kildekode var verdt å satse på. Og IT-avdelingen er fornøyd i etterkant.

**Praksis:** For brukerne er det viktig at funksjonaliteten er den samme, og at grensesnittet er likt.

Av erfaringene kan nevnes at de var positivt overrasket over at åpen kildekode-løsninger ikke bare finnes "horisontalt", i infrastruktur-system (Linux, Apache), men at gode løsninger også finnes "vertikalt", som synlige applikasjoner. De var klar over risikomoment når det gjelder videreutvikling og vedlikehold av åpen kildekode. Når de leter etter nye løsninger, ser de etter graden av aktivitet på f.eks. SourceForge for å finne produkt som brukes av mange.

Hovedskepsisen deres mot å gå over til åpen kildekode var likevel at det er komfortabelt å kunne ringe et telefonnummer når man har et problem, i stedet for å lete etter en løsning på Internett.

Men en organisasjon som tar i bruk åpen kildekode, må være klar over at det er kostnader forbundet med implementasjon, tilpasning og support. De kommenterer reaksjoner fra ledelsen slik: *Man kan forsvare å betale 500.000 i support for et produkt som koster 1 million, men man vil ikke betale 500.000 i support for et produkt som er gratis.*

### 4.1.3 Hvorfor ikke åpen kildekode i helsesektoren i Quebec

I forbindelse med en større omlegging av helsesektoren i provinsen Quebec i Canada [22], ble en overgang til åpen kildekode sett på som en mulighet.

Bakgrunnen for å vurdere åpen kildekode var først og fremst økonomi: 44 % av 2007-budsjettet til provinsen Quebec gikk til helse og sosial, av dette utgjorde IT bare 1.8 %. Selv om helsemyndighetene var klar over at den planlagte organisatoriske omleggingen ville medføre behov for utstrakt bruk av informasjonsteknologi, førte ikke endringen med seg økning i IT-budsjettene. Utgiftene økte, bl.a. fordi det i 2007 ble tegnet en ny 3-års-kontrakt med en stor kommersiell leverandør (Microsoft, antakelig, men ikke navngitt) der de mista tidligere rabatter slik at lisenskostnadene gikk opp med 400 %.

Omleggingen i helsevesenet, som skulle fremme samarbeid mellom nivåene, ville også vært en naturlig bakgrunn for å vurdere åpen kildekode. Den nye strukturen ville passet enda bedre med delingsfilosofien omkring åpen kildekode.

Det ble imidlertid ingen overgang til åpen kildekode. Man ønsket å finne ut hvorfor, og derfor ble 15 IT-sjefer ("CIOs") intervjuet om hva de så som hindringer for å ta i bruk åpen kildekode.

Man konkluderte med at de viktigste barrierene mot å gå over til åpen kildekode i helsesektoren i Quebec var:

1. Mangel på interne IT-ressurser og ekspertise
2. Politisk påvirkning internt og eksternt
3. Mangel på pålitelig informasjon om åpen kildekode-produkt
4. Konservativt miljø blant IT-ledere i helsesektoren
5. Mangel på ansvarlig tredjepart (leverandør)
6. Individualistisk og konkurransepreget kultur (ikke delings-kultur)
7. Skjulte kostnader ved åpen kildekode-produkt

## 4.2 Åpen kildekode i helsesektoren i Norge

For å få et visst inntrykk av holdningen til og utbredelsen av åpen kildekode i helsesektoren i Norge, valgte vi å intervju representanter for ulike aktører i sektoren. Vi hadde utarbeidet ei liste med spørsmål som vi benyttet som et utgangspunkt for intervjuene. (Se vedlegg B.1.) Spørsmålene i lista ble ikke fulgt slavisk, men ble tilpasset det enkelte intervjuobjekt.

Vi intervjuet representanter for driftsorganisasjoner, helsenettet, leverandører av journal-system og mindre fagapplikasjoner, helsepersonell, en tjenesteleverandør og noen pilot-prosjekt. For en detaljert oversikt over hvilke organisasjoner som ble intervjuet og hvem som deltok i intervjuene, se vedlegg B.2.

June Henriksen [20] intervjuet representanter for noen av de samme typer aktører i 2005 i forbindelse med en prosjektoppgave som del av et masterstudium i informatikk ved Universitetet i Tromsø. En av hennes informanter sa at sykehus som er ute etter ny programvare *først* vil se etter funksjonalitet, dernest vil de vurdere service-avtaler (hvor fort blir problem håndtert/rettet) og leverandørens rykte (f.eks. sannsynligheten for at denne eksisterer om ti år). Et annet kriterium er den HW programvaren kjører på. Mange frykter at leverandøren av utstyr reduserer garantien dersom de velger en annen programvare enn det leverandøren anbefaler.

I intervjuene fra 2005 kom det også fram at når det oppstår problem med en programvare, vil det føles betryggende å ha noen (en leverandør) å henvende seg til for å få hjelp. Å vite at det firmaet som utviklet programvaren også er de som retter den, øker følelsen av sikkerhet.

I følge June Henriksen [20] var det så godt som null sjanse for at noen av de leverandørene innen helse som hun intervjuet, ville åpne opp programvare som allerede var lukket.

Vi ønsket bl.a. å undersøke om de samme holdningene gjelder i dag, og om det er andre moment som er viktig for aktørene når det gjelder vurderingen av åpen kildekode.

I våre intervju kjenner vi igjen mange av de samme argumentene som i intervjuene fra 2005 [20]: IT-driftspersonell vil ha den beste løsningen, uavhengig av om det er fri eller proprietær programvare, og leverandører er negative til å legge ut sine produkt som åpen kildekode, først og fremst med tanke på fortjeneste. Vårt inntrykk er likevel at det er en større aksept for åpen kildekode i sektoren nå, f.eks. på de større sykehusene.

I de følgende avsnittene oppsummerer vi synspunktene som kom fram i intervjuene. En mer utfyllende oppsummering kan leses i vedlegg B.3.

#### *4.2.1 Utbredelse av åpen kildekode i helsesektoren*

Vårt inntrykk er at helsesektoren i økende grad vurderer å ta i bruk åpen kildekode der dette synes hensiktsmessig. Hittil har aktørene i sektoren i større grad tatt i bruk åpen kildekode på serversiden (horisontale system) enn på klient- og applikasjonssiden (vertikale system). Årsakene til dette er bl.a. følgende:

- Det som er tilgjengelig av åpen kildekode for horisontale applikasjoner kan oppleves som mer stabilt enn det som finnes for vertikale applikasjoner.
- Man ikke blir bundet til å tilgjengeliggjøre for andre eventuell egenutviklet kode for applikasjoner og program som kun benyttes internt til drift i egen virksomhet, selv om man benytter åpen kildekode som er lisensiert under GPL.
- Det er stor utbredelse av Microsoft-produkt i sektoren, både horisontalt og vertikalt, og at miljøene er usikre på i hvilken grad åpen kildekode fra andre lar seg integrere mot Microsoft-produktene.
- IT-miljøene i sektoren har langt mer kompetanse på proprietær/lukket programvare (f.eks. Microsoft-produkt og Unix) enn på programvare med åpen kildekode.
- Det er usikkerhet med tanke på tilgjengelig support og garanti for feilretting ved bruk av åpen kildekode. Man har en følelse av å ha "ryggen fri" når man bruker kommersielle leverandører.

Leverandørene av applikasjoner og verktøy til helsesektoren er foreløpig nokså tilbakeholdne med hensyn til å skulle gi ut løsningene sine som åpen kildekode. Noen av de viktigste årsakene til dette er at:

- Markedet for helsespesifikke IT-løsninger er relativt begrenset i Norge, og til dels uforutsigbart. Flere av de store helseforetakene har f.eks. skiftet leverandør av journalsystem i løpet av de siste to årene. Leverandørene må endre drastisk på forretningsmodellene hvis de skal gå over til å levere åpen kildekode. Dette kan være vanskelig i et nisjemarked som helsesektoren (frykten for å gi bort "arvesølvet" er stor).

- Det er usikkerhet med tanke på hvem som har ansvar for å finne og rette feil som måtte oppstå i system hvor kunden har tilgang til kildekoden og til å legge inn egenutviklet kode eller kode fra tredjepart.
- Det er usikkerhet med hensyn til hvilken garanti leverandøren skal gi for kode fra tredjeparter, som er integrert i egne løsninger.

Fokuset hos brukerne og driftsmiljøene i sektoren synes å være på hvilken programvare som har den ønskede eller beste funksjonalitet og som er mest hensiktsmessig ut fra en totalvurdering, og ikke på hvorvidt programvaren er åpen eller ikke. Når det gjelder åpen kildekode-applikasjoner innen f.eks. bildediagnostikk (røntgen etc), er man i ferd med å teste ut løsninger som på en del områder synes å ha bedre funksjonalitet enn tilsvarende lukkede løsninger.

Samtidig er man opptatt av lisensproblematikken som ved bruk av proprietær eller lukket programvare i mange tilfelle gir uønskede bindinger og begrensninger, både økonomisk, sikkerhetsmessig og driftsmessig. For driftsorganisasjoner som leverer tjenester vil det stort sett være uproblematisk å benytte åpen kildekode, inkludert kode lisensiert under GPL.

For leverandører som tilbyr produkt og verktøy i motsetning til tjenester, kan også lisensene for åpen kildekode gi begrensninger, spesielt GPL. Andre åpen kildekode-lisenser som f.eks. Apache eller BSD, gir færre begrensninger for egenutviklet kode som er integrert med den åpne kildekoden. Leverandørene benytter i utstrakt grad biblioteksrutiner som er åpen kildekode, da disse i liten grad skaper konflikter med tanke på lisensiering av den egenutviklede koden.

I pilotprosjekt er det større åpenhet for å benytte åpen kildekode som utgangspunkt for det som skal utvikles, slik at man ikke trenger å utvikle alt selv. Pilotprosjektene utvikler som regel mindre applikasjoner, noe som kan gjøre det lettere å teste all koden i sammenheng enn ved utvikling av f.eks. et helt journalsystem.

De som har tatt i bruk åpen kildekode, opplever at support og feilretting stort sett er like god (eller dårlig), men i noen tilfelle bedre, enn ved kjøp av kode fra leverandører av proprietær eller lukket programvare.

#### *4.2.2 Fordeler og utfordringer ved bruk av åpen kildekode*

De som har tatt i bruk åpen kildekode nevner en rekke fordeler de opplever ved slik kode kontra lukket kode. Noen av dem er tatt med i lista under:

- Det er gjerne en lavere terskel for å teste ut slike program, da man ikke nødvendigvis må forplikte seg til å kjøpe programmet før man installerer det for utprøving og eksperimentering over lengre tid.
- Åpen kildekode gir som regel større frihet når det gjelder valg av maskinvare.
- Ved implementering av åpen kildekode kan man i større grad la være å installere moduler/komponenter man ikke ønsker av sikkerhetsårsaker eller andre grunner
- Man slipper til enhver tid å ha oversikt over antallet brukere og antallet lisenser for de ulike applikasjoner
- Driftsavdelinger i helsesektoren som har gått over til Linux, har veldig positive erfaringer med dette både når det gjelder stabilitet, driftssikkerhet og håndtering av lisenser.

- Hvis man har tilgang til kildekode er det mulig å gjøre endringer og feilrettinger selv i stedet for å måtte vente på at leverandøren eventuelt gjør det man ønsker, og det kan være lettere å finne ut hva som er årsakene til feil som oppstår.
- Ved å produsere en tjeneste på en plattform som er åpen kildekode kan man komme raskere på markedet enn hvis man må utvikle plattformen selv. Man får også endringene som alle andre bidragsyttere kommer med.
- Pilotprosjektene opplever at det er mye enklere å tilpasse ulike program til hverandre når koden er åpen (det kan være vanskelig å bygge videre på lukket kode).
- Ved å bygge videre på åpen kildekode slipper man å bruke ressurser på å selv utvikle kode som er utviklet minst like godt av andre.
- Mange mener at kvaliteten på programvaren ofte blir bedre hvis koden skal gis ut som åpen kildekode, fordi utviklerne har større motivasjon for å lage kode av god kvalitet i og med at navnet deres blir eksponert sammen med koden.

Noen påpekte også en del utfordringer knyttet til det å bruke åpen kildekode, som f.eks.:

- Bruk av åpen kildekode er ikke kostnadsfritt slik mange tror, men krever at man har nødvendig kompetanse til intern drift, og at man setter av nødvendige midler til installering, drift, vedlikehold og support av løsningene<sup>37</sup>.
- For mindre, nisjepregede system er det viktig å sikre seg at det eksisterer stabile miljø som supporterer og videreutvikler systemet.
- Ved integrering av åpen kildekode i egenutviklet kode er det en utfordring å avklare ansvarsforholdene knyttet til feilsituasjoner og oppgradering av den åpne koden

#### 4.2.3 Sikkerhetsmessige aspekt

Intervjuobjektene hadde også noen tanker omkring sikkerhetsmessige aspekt knyttet til det å benytte åpen kildekode. Vi oppsummerer noen av dem i lista under:

- Det at koden er åpen gjør det mulig for dem som ønsker å finne sårbarheter i koden å søke etter slike. Utro tjenere internt i en virksomhet vil også ha denne muligheten.
- Av sikkerhetsmessige grunner kan det være ønskelig å begrense funksjonaliteten i et system. Dette er som regel mye enklere å få til med åpen kildekode, da man får mulighet til å utelate de delene av systemet man ikke ønsker å benytte.
- De fleste vi intervjuet mener det ikke er noen særlige sikkerhetsmessige forskjeller på å bruke proprietær programvare kontra åpen kildekode. I noen tilfelle kan det være en fordel å bruke åpen kildekode i og med at den kan være inspisert av flere, og at feil og svakheter kan bli oppdaget og rettet raskere enn for proprietær kode dersom det står et aktivt miljø bak utviklingen av den åpne koden. Man har også mulighet til å sjekke koden selv og rette koden. Men det er viktig at man er kritisk til det man tar i bruk, og at man sjekker kvaliteten og "ryktet" til koden før man bruker den.

---

<sup>37</sup> "Åpen kildekode tiltrekker seg gjerrignarker", som en av våre informanter sa.

#### 4.2.4 *Har helsesektoren spesielle utfordringer mht. programvare?*

Vi spurte intervjuobjektene om de så noen forskjeller i det å benytte åpen kildekode i helsesektoren kontra andre sammenhenger. Følgende aspekt ble nevnt:

- Helsevesenet har mange ansatte, og bruker mye midler på lisenser for kontorstøttesystem o.l. Det vil gi store besparelser å innføre åpen kildekode for slike horisontale system i helsesektoren.
- Når det gjelder de vertikale systemene, er det gjerne større utfordringer for helsesektoren enn for en del andre sektorer. De fagspesifikke applikasjonene i helsesektoren er mer nisjepregede enn f.eks. personaladministrasjonssystem, økonomisystem, kundebehandlingsprogram etc. Dette medfører at det i utgangspunktet er færre aktører som kan bidra til utvikling og feilretting av slike system.
- Standardene for operativsystem, web-browsere etc. er veldig klart definert. Journalssystem er mye mer komplekse og mindre standardiserte. Det er derfor en større utfordring å designe og utvikle et journalsystem basert på åpen kildekode.
- Journalsystem for sykehus er svært kompliserte. Mange ulike applikasjoner skal spille sammen, og tilgangsstyringen er kompleks og dynamisk. Integrasjon mellom de ulike delsystemene er avgjørende for gode arbeidsprosesser og for riktig og oppdatert informasjon om pasienten til enhver tid. Dette stiller store krav til utviklingen av systemene, og til test- og godkjenningprosessen. Det er derfor viktig å ha en solid organisasjon som står bak utviklingen og forvaltningen av slike system og tar ansvar for at journalsystemene forholder seg til gjeldende regelverk.
- Hvis et sykehus tar inn kode utenfra etter at de har fått et ferdig testet system fra journalleverandøren, må de selv ta ansvar for dette. Det er ikke gitt at disse vil kjøre sammen. Det kan være vanskelig å finne gode nok testmiljø for å kontrollere om den nye koden fungerer tilfredsstillende sammen med det man har fått fra journalleverandøren.

#### 4.2.5 *Bruk av åpen kildekode og åpne standarder*

Både leverandører og driftsmiljø mener det er viktigere at programvaren følger åpne og godt dokumenterte standarder enn at den er åpen kildekode. De mener også at det er viktigere med standardiserte driftsmiljø og standardiserte komponenter enn å ha tilgang til kildekoden.

Et åpent grensesnitt mot komponentene man skal bruke, anses som langt viktigere enn å ha tilgang til komponentenes kildekode. Også internt i et journalsystem er det ofte nok å ha kunnskap om grensesnitt man skal forholde seg til slik at man ikke trenger gå til kildekoden for å forstå hva koden gjør.

En del aktører mener at utviklingen av helseapplikasjoner i Norge bør bevege seg mer over på internasjonale og godt beskrevne standarder, bl.a. HL7, slik at andre kan koble seg opp mot systemene uten å kjenne detaljert hvordan de er bygd opp internt. De mener også at de norske standardene ikke på langt nær oppfyller dette.

#### 4.2.6 *Aktørenes bidrag til utvikling av åpen kildekode*

Aktørene i helsesektoren bidrar i liten grad til andres utvikling av åpen kildekode. Dette skyldes bl.a. at det er vanskelig å få ressurser til slikt arbeid. En annen årsak er at den tilpasningen og videreutviklingen som blir gjort av åpen kildekode lastet ned fra andre, som regel ikke anses å ha interesse for flere da den ofte blir skreddersydd til de lokale

forholdene. Aktørene gir imidlertid tilbakemelding til utviklerne om feil de finner, og de sender i noen grad inn forslag til endring og/eller forbedring av koden.

De fleste driftsorganisasjoner utvikler noe programvare selv. Hittil har de ikke i særlig grad delt denne koden med andre. Uten økonomiske insentiver er det lite motivasjon for deling når de har brukt interne ressurser til å utvikle denne programvaren.

Ullevål universitetssykehus har imidlertid nylig besluttet at de skal lisensiere egenutviklet programvare rundt integrasjon og samhandling som fri programvare.

## 5 Trusler og mottiltak

I dette avsnittet kommenterer vi truslene og gir forslag til mottiltak. Vi har ikke gjort en fullstendig risikovurdering av de kartlagte truslene, da det ikke er mulig å si noe spesifikt om sannsynlighet for truslene uten å se dem i sammenheng med omgivelsene programvaren skal anvendes innenfor, og de tilhørende organisatoriske rutiner og prosedyrer.

Vi har i det følgende gruppert trusler som henger sammen. Til slutt nevner vi også noen direkte fordeler knyttet til bruk av programvare som er lisensiert som åpen kildekode.

Tabellen i vedlegg A oppsummerer trusler ved bruk av åpen kildekode som er kommet til uttrykk i intervju og som går igjen i litteraturen, med referanse til hvor de er hentet fra. Tabellen har en tilsvarende kolonne som framhever de positive sidene ved åpen kildekode.

### *Leverandørvhengighet/support*

- Helsesektorens strenge krav til oppetid for systemene medfører behov for tilgang til supportpersonell 24 timer i døgnet. Det er en utbredt frykt for at dette behovet ikke kan tilfredsstilles ved bruk av åpen kildekode, noe som kan medføre redusert driftsstabilitet og redusert tilgjengelighet til system og informasjon.
- Mangel på tilstrekkelig intern IT-kompetanse (personell som kjenner systemet).

Hvis man skal ta i bruk programvare som er åpen kildekode, er det viktig å sikre seg at man likevel har tilgang til nødvendig support, enten via avtale med ekstern leverandør av slik support, eller ved at man har eller bygger opp tilstrekkelig intern kompetanse på den aktuelle programvaren.

### *Uklare og mangelfulle ansvarsforhold*

- Hvis leverandøren integrerer åpen kildekode fra andre i sitt eget produkt, kan det være uklart for leverandøren hvilket ansvar de tar på seg.
- Hvis brukerne gjør endringer eller tilpasninger i programvare fra leverandør, er det ikke opplagt hvem som skal være ansvarlig for å søke etter og rette feil som måtte oppstå.

Leverandører som tar inn åpen kildekode fra andre i egne program har til syvende og sist selv ansvaret for kvaliteten av den samlede koden, og at den fungerer etter hensikten.

Leverandøren bør også ha klare avtaler med kunden når det gjelder begrensning av sitt ansvar dersom kunden legger inn eller endrer kode.

Ansvarsforholdene må i alle tilfelle tydelig klargjøres overfor kunden.

### *Uforutsigbar forvaltning av åpen kildekode*

- For programvare med åpen kildekode kan det i noen tilfelle være større usikkerhet med tanke på om koden blir oppgradert i takt med endringer i annen programvare den skal samspille med.
- Noen mener det er større usikkerhet med tanke på programvarens levetid for system med åpen kildekode enn for system fra leverandører av lukket kode. Har man tilgjengelig support og forvaltning av koden hvis opprinnelig utgiver "dør"?
- Hvis kun et fåtall utviklere står bak vil det være større risiko forbundet med å ta programvaren i bruk, pga. usikkerhet omkring support og kodens levetid.

Ved bruk av åpen kildekode er det viktig å forsikre seg om at det står et robust og stabilt miljø bak programvaren. Tendensen i dag er at det bygges opp virksomheter som spesialiserer seg på åpen kildekode og forvaltning av denne typen programvare, f.eks. LinPro i Norge og RedHat for Linux.

#### *Mange "uavhengige" utviklere*

- Bidrag fra mange utviklere kan medføre at ingen har god oversikt over helheten i programmet eller systemet, og at koden kan bli fragmentert og usammenhengende.
- Det er uvisst om noen tar ansvar for systematisk gjennomgang av koden for å avsløre feil når ingen betaler dem for å gjøre det.

Det er viktig at noen har det overordnede ansvaret for utviklingen av et system, og for å kontrollere at alle bidrag har god nok kvalitet.

#### *Tillit til leverandør*

- Pålitelig informasjon om åpen kildekode-produkt kan være vanskelig tilgjengelig. Dermed blir det vanskelig å vite om man velger produktet med best kvalitet.
- Mange opplever utrygghet ved å ta i bruk produkt fra leverandører man ikke kjenner godt.
- Det kan være vanskelig å være helt sikker på at web-siden man laster ned koden fra er den autentiske siden til utgiveren.

Organisasjoner som tilbyr kvalitetssikret informasjon om leverandører av åpen kildekode og produkt vil kunne avhjelpe dette noe.

#### *Tilgang til og eksponering av koden*

- Åpen kode gjør det mulig for "alle" å undersøke koden for å finne feil og svakheter som kan utnyttes. Det vil kunne være nok å finne én feil for å kunne angripe systemet.
- Tilgang til koden gjør det lettere å legge inn bakdører og ondsinnet kode, også for utro tjenere hos brukerorganisasjonen.
- Eksponering av koden gjør at det også er mulig for alle å sette seg inn i sikkerhetsmekanismene og forsøke å utnytte eller omgå disse.
- Tilgang til åpen kildekode gjør det enklere å lage modifiserte kopier som kan benyttes til ondsinnede handlinger.

Følgende fordeler ved åpen kildekode fremheves ofte som bidrag til å redusere disse truslene: For åpen kildekode vil det kunne være en større brukergruppe som potensielt kan oppdage feil og bidra til retting. De som ser koden kan selv evaluere kvaliteten av sikkerheten i koden, og det vil være lettere å avsløre bakdører og ondsinnet kode. Det vil også være mulig å compilere koden selv, og eventuelt sammenligne kjørbare versjoner fra ulike kompilatorer. Når man har tilgang til koden kan man endre og tilpasse koden til egne behov. Man har også større fleksibilitet når det gjelder hvilke deler av systemet man vil installere.

Sikkerheten skal ikke være basert på hemmelighold av hvilke sikkerhetsmekanismer som er implementert. Sikkerheten i en organisasjon er i stor grad avhengig av organisasjonens evne og forpliktelse til å håndtere sikkerheten – uavhengig av om det brukes åpen eller lukket kildekode [4].

Digital signatur av koden som legges ut (f.eks. ved bruk av sertifikater) kan benyttes for å avsløre uautoriserte versjoner av koden.

### *Manglende sikkerhetsfokus*

- De som utvikler kode er ikke nødvendigvis opptatt av sikkerhet, men av funksjonalitet.
- De som tester koden er ikke nødvendigvis opptatt av å teste med tanke på sikkerhet, men først og fremst med tanke på funksjonalitet.
- Det er uvisst om personer med den rette kompetansen går gjennom koden for å finne sikkerhetsfeil.

Ved bruk av åpen kildekode er det viktig å forsikre seg om at de som har utviklet koden har hatt nødvendig fokus på kvalitet og sikkerhet, og at det står en ansvarlig organisasjon bak utvikling og support. I åpen kildekode-prosjekt vil man kunne ha bedre tid til kvalitetssikring av koden fordi man ikke nødvendigvis er begrenset av "time-to-market"-presset som de kommersielle leverandørene er styrt av. Det er også grunn til å anta at motivasjonen for å lage kode av god kvalitet er stor fordi utviklerne eksponerer seg selv og ikke kan skjule seg bak et firmanavn.

### *Mangelfullt testmiljø*

- Når mange bidrar til utvikling og videreutvikling av kode, er det ikke sikkert noen har utviklet et komplett testmiljø for helheten.
- Hvis brukerne tar inn åpen kildekode i kode fra leverandør, må de også ha et testmiljø for å teste at integrasjonen fungerer etter hensikten og ikke medfører nye sårbarheter eller feil. Det er lite sannsynlig at slike testmiljø eksisterer hos brukerne selv.
- Når leverandører tar inn åpen kildekode fra andre i sine system, har de ikke nødvendigvis et testmiljø tilpasset til også å omfatte denne koden.

Hvis man skal integrere åpen kildekode fra andre, er det viktig å sørge for at man har et testmiljø som omfatter helheten i systemet.

### *Lisensrelaterte fordeler*

Det er visse sikkerhetsmessige fordeler ved åpen kildekode relatert til lisensiering.

Ved bruk av åpen kildekode unngår man uønskede bindinger og begrensninger både økonomisk, sikkerhetsmessig og driftsmessig pga. større frihet til å velge ulike kombinasjoner av programvare og utstyr.

Det er enklere å iverksette redundans i system og backup-løsninger når man ikke trenger å betale lisens for hver installasjon av programvaren. Av samme grunn får man også større mulighet for å isolere tjenester fra hverandre ved å kjøre dem på hver sin server, noe som hindrer at utløsning av sårbarheter i en tjeneste påvirker andre tjenester. Oppstart av servere og applikasjoner etter uforutsett driftsstans kan skje raskere når man ikke er avhengig av å ha lisenskode tilgjengelig.

## 6 Konklusjoner og anbefalinger

### Generelle inntrykk

Vårt inntrykk er at helsesektoren i økende grad vurderer å ta i bruk åpen kildekode der dette synes hensiktsmessig. Det kan være flere årsaker til dette. Reduserte kostnader og problem omkring håndtering av lisenser vil nok ofte være et hovedmotiv. Lisenskostnadene utgjør en stor andel av et IT-budsjett, og det er vanskelig til enhver tid å ha oversikt over antall lisenser som skal betales for de ulike systemene. Og skal man ha redundans og backup-system betyr det som oftest dobbelt sett av server-lisenser. Et annet motiv er at produkt med åpen kildekode ofte gir større fleksibilitet med tanke på tilpasning av programvaren og valg av HW. Og i noen tilfelle er funksjonaliteten bedre i produkt som har åpen kildekode enn i program med lukket kode. Våre informanter mener at stabiliteten til åpen kildekode-produkt på server-siden også har vist seg å være minst like god som for proprietære og lukkede produkt.

### Sikkerhet

Når det gjelder sikkerheten i åpen kildekode, er det ulike meninger om den. De fleste vi intervjuet mener det ikke er noen særlige sikkerhetsmessige forskjeller på å bruke proprietær programvare kontra åpen kildekode. Noen mener imidlertid at det vil være for risikabelt å gjøre koden tilgjengelig fordi dette vil gi potensielle hackere en for stor fordel. Andre, derimot, mener at det å gjøre koden tilgjengelig vil gjøre koden mer motstandsdyktig mot angrep etter relativt kort tid, i og med at "alle" kan søke etter sårbarheter og feil i koden og bidra med feilrettinger.

For programvare som benyttes av mange er vi enig i at tilgjengeliggjøring av koden vil bidra til mer og raskere feilretting. For programvare som har en snevrere målgruppe er det imidlertid mer usikkert i hvilken grad koden vil bli gjenstand for slik kvalitetssikring.

Fokus på sikkerhet i utviklingen er uavhengig av om koden gis ut som åpen kildekode eller ikke. For dem som skal ta koden i bruk er det viktig å vite om utviklerne har hatt fokus på sikkerhet.

Journalssystem og andre fagapplikasjoner i helsesektoren er underlagt strenge krav til funksjonalitet og sikkerhet. Ved utvikling av slike system er det helt nødvendig at utvikleren legger til rette for at systemet gjør det mulig for brukerne å etterleve gjeldende regelverk. Dersom slike fagapplikasjoner skal utvikles som åpen kildekode, vil det derfor være avgjørende at det står en organisasjon bak som kjenner disse kravene og sørger for at nødvendig funksjonalitet blir implementert. En annen mulighet er at helsesektoren, f.eks. i regi av Nasjonal IKT, danner et bestillerforum som etterspør den nødvendige funksjonalitet fra utviklerne, og kanskje også betaler for slik utvikling.

### Utbredelse i helsesektoren

Sektoren er i økende grad åpen for å vurdere åpen kildekode-løsninger for horisontale system. Men jo mer spesialiserte og helsespesifikke komponentene blir, jo mindre sannsynlig er det at det finnes åpne løsninger.

Leverandørene av fagspesifikke applikasjoner til helsesektoren (vertikale system) er lite villig til å gi ut koden som åpen, bl.a. fordi det vil medføre en forretningsmessig utfordring. Det vil også kunne by på utfordringer med tanke på helhetlig testing av systemene dersom kundene integrerer egenutviklet eller andres kode med leverandørens kode.

Driftsavdelingene ved sykehusene er mer åpen for å tilgjengeliggjøre egenutviklet kode for de andre sykehusene slik at de kan dra nytte av deres arbeid. Men det kan være en utfordring å få ressurser til nyutvikling og vedlikehold, og til forvaltning av slik kode på vegne av flere brukerorganisasjoner (HF). Ullevål universitetssykehus har imidlertid besluttet (høsten 2008) at de skal lisensiere egenutviklet programvare rundt integrasjon og samhandling som fri programvare.

Tilgjengelighet til standarder er viktig for utnyttelse av åpen kildekode. Det vil fortsatt være slik at helsesektoren vil ha et fundament av kommersielle systemer som moduler av åpen kildekode vil måtte tilpasses [25]. For å få dette til er det viktig at både de proprietære, lukkede produktene og produkt med åpen kildekode baserer seg på åpne, tilgjengelige standarder.

Det synes som om HL7-baserte standarder er i ferd med å få en stadig større utbredelse i sektoren. Dette vil gjøre informasjonsutveksling enklere både internt i sykehus og mellom fagsystemer i ulike organisasjoner.

### **Tiltak og anbefalinger**

Hvis man skal integrere åpen kildekode fra andre i egen programvare, er det viktig å avklare hvem som har ansvar for å detektere og rette feil. Da er det også viktig å sørge for at man har et testmiljø som omfatter helheten i systemet. Leverandører som tar inn åpen kildekode fra andre i egne programmer har til syvende og sist selv ansvaret for kvaliteten av den samlede koden, og at den fungerer etter hensikten. De har det overordnede ansvaret for utviklingen av systemet.

Tilbakemeldingen fra mange av våre intervjuobjekt når det gjelder bruk av åpen kildekode er at de er utrygge på om det finnes et robust og stabilt miljø bak programvaren, et miljø som står inne for kvalitet og sikkerhet og som kan garantere for fortsatt vedlikehold og support.

Det kan være en utfordring å få full oversikt over programvare som andre har laget, selv med kildekoden tilgjengelig. Enten må man ha en leverandør/organisasjon som tar ansvaret for kvalitet og sikkerhet, eller så må man selv bygge opp tilstrekkelig intern kompetanse på programvaren.

Tendensen i dag er at det bygges opp virksomheter som spesialiserer seg på åpen kildekode og forvaltning av denne typen programvare, og som tar seg betalt for å gjøre denne jobben. Det er dette som av enkelte kalles "andre generasjon åpen kildekode", på engelsk kalt Second Generation Open Source eller OSS 2.0.

Organisasjoner som tilbyr kvalitetssikret informasjon om åpen kildekode (produkt og miljø) vil kunne avhjelpe den usikkerheten som mange føler når det gjelder å ta i bruk slik programvare. I et noe nisjepreget miljø som helsesektoren er det gjerne myndighetene som må sørge for at noen vil ta på seg ansvaret for å vedlikeholde og videreutvikle åpen kildekode som skal brukes. I dette ligger også et ansvar for å sikre at programvaren tilrettelegger for at brukerne kan oppfylle lovpålagte reguleringer.

Intervjuene våre viser at både leverandørene og sektoren selv mangler insentiver når det gjelder å ta i bruk og gi ut åpen kildekode.

Mange i helsesektoren vil kanskje si seg enig med McDonald et al [25] når de konkluderer med hva som skal til for å øke utbredelsen av åpen kildekode i helsesektoren:

- Det er naturlig at all programvare som utvikles med offentlig finansiering/støtte legges ut som åpen kildekode.
- Man må kunne kreve at programvare utviklet med offentlig støtte følger internasjonale standarder.

- Man må ha en mekanisme for å informere helsesktoren om tilgjengelige produkt med åpen kildekode.



## Forkortelser

AMIA	American Medical Informatics Association
API	Application Program Interface
BSD	Berkeley Software Distribution
CEN	Comité Européen de Normalisation (den europeiske standardiseringsorganisasjon)
CIO	Chief Information Officer (IT-sjef)
CMS	Content Management System / Clinical Management System
CT	Computertomografi (metode for medisinsk bildediagnostikk)
CVS	Concurrent Versioning System (versjonskontrollsystem)
DICOM	Digital Imaging and Communications in Medicine
DNS	Domain Name System (for mapping mellom navn og IP-adresser)
EDI	Electronic Data Interchange
EPJ	Elektronisk pasientjournal
EU	European Union
EUPL	European Union Public License
FAD	Fornyings- og administrasjonsdepartementet
FLOSS	Free/Libre Open Source Software
FOSS	Free Open Source Software
FS	Free Software (fri programvare)
FSF	Free Software Foundation
GEHR	Good European Health Record (europeisk prosjekt)
GEHR	Good Electronic Health Record (australsk prosjekt)
GNU	(GNU's Not Unix) Unix-lignende operativsystem
GPL	GNU General Public License
GUI	Graphical User Interface
HF	Helseforetak
HL7	Health Level Seven (non-profit internasjonal organisasjon som utvikler standarder for utveksling av helsedata)
HP	Hewlett-Packard
HW	hardware (maskinvare)
IBM	International Business Machines
IE	(Microsoft) Internet Explorer
IKT	Informasjons- og kommunikasjonsteknologi
IP	Internet protocol

IPR	Intellectual Property Right
ISO	International Organization for Standardization (den internasjonale standardiseringsorganisasjonen)
IT	Informasjonsteknologi
LGPL	Library GPL (Lesser GPL)
LMS	Learning Management System
MIT	Massachusetts Institute of Technology
MPL	Mozilla Public License
MR	Magnetisk resonans (tomografi, metode for medisinsk bildediagnostikk)
MTBF	Mean Time Between Failures
NPL	Netscape Public License
NST	Nasjonalt senter for telemedisin
OS	Operativsystem
OSI	Open Source Initiative
OSS	Open Source Software
PACS	Picture archiving and communication system (for medisinsk bildediagnostikk)
PET	Positronemisjonstomografi (teknikk for medisinsk bildediagnostikk)
PHP	PHP: Hypertext Preprocessor (programmeringsspråk/scriptspråk)
PHR	Personal Health Record
PKI	Public Key Infrastructure (Asymmetrisk krypteringsmetode basert på to nøkler)
RIS	Røntgen-informasjonsystem
SW	software (programvare)
TTL	Tromsø Telemedicine Laboratory
UNN	Universitetssykehuset Nord-Norge HF

## Ordliste

Copyleft	Fri bruk, i motsetning til "copyright", men med den betingelse at alle andre også får de samme frie rettighetene
Copyright	Opphavsrett
Freeware	Gratis programvare (uavhengig av om kildekoden er tilgjengelig)
Public domain software	Programvare uten opphavsrett (lisensfri)
Shared source	Microsofts betegnelse på programvare som (utvalgte) kunder får tilgang til kildekoden til
Shareware	Programvare som er gratis i en begrenset tidsperiode

## Litteraturliste

1. Utlysningen: [http://www.regjeringen.no/nb/dep/fad/Tema/IT-politikk\\_eNorge/-Prosjektstotte-for-fri-programvare-i-20.html?id=482319&epslanguage=NO](http://www.regjeringen.no/nb/dep/fad/Tema/IT-politikk_eNorge/-Prosjektstotte-for-fri-programvare-i-20.html?id=482319&epslanguage=NO) (sist sett 21.10.08)
2. Tildelingen: [http://www.regjeringen.no/nb/dep/fad/Tema/IT-politikk\\_eNorge/Disse-prosjektene-mottar-stotte-.html?id=493598](http://www.regjeringen.no/nb/dep/fad/Tema/IT-politikk_eNorge/Disse-prosjektene-mottar-stotte-.html?id=493598) (sist sett 21.10.08)
3. David A. Wheeler: Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the numbers. [http://www.dwheeler.com/oss\\_fs\\_why.html](http://www.dwheeler.com/oss_fs_why.html) - April 2007 (sist sett 21.10.08)
4. Michael Overly: The open source handbook. Pike & Fisher Inc, 2003
5. The Free Software definition <http://www.gnu.org/philosophy/free-sw.html> (sist sett 10.09.08)
6. Eric S. Raymond: The Cathedral and the Bazaar <http://catb.org/~esr/writings/cathedral-bazaar/> - 2000 (sist sett 21.10.08)
7. Eric S. Raymond: The Magic Cauldron <http://catb.org/~esr/writings/magic-cauldron/magic-cauldron.html> - June 1999 (sist sett 21.10.08)
8. Bruce Perens: The Open Source Definition <http://perens.com/OSD.html> January 1999 (sist sett 21.10.08) (Artikkelen er med i boka: "Open sources: Voices from the Open Source Revolution". O'Reilly, 1<sup>st</sup> edition, January 1999)
9. Richard Stallman: Why "Open Source" misses the point of Free Software <http://www.gnu.org/philosophy/open-source-misses-the-point.html> (sist sett 21.10.08)
10. Richard Stallman: Why "Free Software" is better than "Open Source". <http://www.gnu.org/philosophy/free-software-for-freedom.html> (sist sett 21.10.08)
11. David A. Wheeler: Secure programming for Linux and Unix HOWTO <http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html> v.3.010, 3 March 2003 (sist sett 21.10.08)
12. Dare Obasanjo: The Myth of Open Source Revisited v2.0. 13 March 2002 <http://www.developer.com/open/article.php/990711> (sist sett 21.10.08)
13. John Viega: Open Source Security: Still a Myth. 16 Sept. 2004, [http://www.oreillynet.com/pub/a/security/2004/09/16/open\\_source\\_security\\_myths.html](http://www.oreillynet.com/pub/a/security/2004/09/16/open_source_security_myths.html) (sist sett 21.10.08)
14. Christian Payne: On the security of open source software, Information Systems Journal Vol.12 Iss.1 2002.
15. Jaap-Henk Hoepman, Bart Jacobs: Increased security through open source. Communications of the ACM, Vol. 50, Issue 1. January 2007, pp. 79 - 83
16. Brian Witten, Carl Landwehr, Michael Caloyannides: Does Open Source Improve System Security? IEEE Software, Volume 18, Issue 5, Sep/Oct 2001 pp 57-61
17. Ross Anderson: Security in Open versus Closed Systems – The Dance of Boltzmann, Coase and Moore. <http://www.cl.cam.ac.uk/~rja14/Papers/toulouse.pdf> Konferanse: Open Source Software Economics, Law and Policy 2002 (sist sett 21.10.08)

18. Ken Thompson: Reflections on Trusting Trust. Comm. of the ACM, Volume 27, No.8, August 1984.
19. Coverity: Coverity Venture with U.S. Department of Homeland Security Resolves Quality Issues and Potential Security Vulnerabilities in 11 Major Open-Source Projects. 8 January 2008.  
[http://www.coverity.com/html/press\\_story54\\_01\\_08\\_08.html](http://www.coverity.com/html/press_story54_01_08_08.html) (sist sett 21.10.08)
20. June Henriksen: Free Medical Software – Ifl, Universitetet i Tromsø, 23. januar 2006
21. Brian Fitzgerald, Tony Kenny (2003): Open Source can improve the Health of the Bank Balance – the Beaumont Hospital Experience. University of Limerick, Ireland.  
<http://www.netproject.com/docs/Beaumont.pdf> (sist sett 23.10.08)
22. Guy Paré, Michael D. Wybo, Charles Delannoy: Barriers to Open Source Software Adoption in Quebec's Health Care Organizations. Journal of Medical Systems, May 22, 2008. <http://www.springerlink.com/content/puj11873w2698u70/> (sist sett 23.10.08)
23. Karl Øyri, Peter J. Murray: osni.info – Using free/libre/open source software to build a virtual international community for open source nursing informatics. International Journal of Medical Informatics, vol. 74, 2005 (pp. 937-945)
24. Michael Goulde, Eric Brown: Open Source Software: A Primer for Health Care Leaders. California HealthCare Foundation, ihealth reports, March 2006  
<http://www.chcf.org/documents/healthit/OpenSourcePrimer.pdf> (sist sett 21.10.08)
25. Clement J. McDonald, G. Schadow, M. Barnes, P. Dexter, J. M. Overhage, B. Mamlin, J. M. McCoy: Open source software in medical informatics – why, how and what. International Journal of Medical Informatics, vol. 69, 2003 (pp. 175-184)
26. Richard T. Watson, Marie-Claude Boudreau, Paul T. York, Martina E. Greiner, Donald Wynn Jr.: The Business of Open Source. CACM, April 2008, Vol.51, no.4, pp. 41-46
27. Brian Fitzgerald: The Transformation of Open Source Software. MIS Quarterly, Vol.30, No.3, 2006
28. ISO/IEC FCD 27000: Information technology – Security techniques – Information security management systems – Overview and vocabulary. 2008-06-06.
29. LOV-2000-04-14-31 – Lov 14. april 2000 nr. 31 om behandling av personopplysninger (Personopplysningsloven).  
<http://www.lovdatab.no/all/hl-20000414-031.html> (sist sett 18.09.08)
30. FOR-2000-12-15-1265 – Forskrift om behandling av personopplysninger (Personopplysningsforskriften)  
<http://www.lovdatab.no/for/sf/fa/fa-20001215-1265.html> (sist sett 18.09.08)
31. LOV-2001-05-18-24 – Lov 18. mai 2001 nr. 24 om helseregistre og behandling av helseopplysninger (Helseregisterloven).  
<http://www.lovdatab.no/all/hl-20010518-024.html> (sist sett 18.09.08)
32. LOV-1999-07-02-64 – Lov 2. juli 1999 nr. 64 om helsepersonell m.v. (Helsepersonelloven). <http://www.lovdatab.no/all/hl-19990702-064.html> (sist sett 18.09.08)

33. LOV-1961-05-12-02 – Lov 12. mai 1961 nr. 02 om opphavsrett til åndsverk m.v. (Åndsverkloven). <http://www.lovdata.no/all/hl-19610512-002.html> (sist sett 18.09.08)
34. Torger Kielland: Copyleft. En analyse av rekkevidden av gjensidighetsvilkår i åpne programvarelisenser i norsk rett. Complex nr. 7/2005. Institutt for rettsinformatikk, Oslo, november 2005.  
<http://www.jus.uio.no/ifp/markedsrett/publikasjoner/copyleft.pdf> (sist sett 18.09.08)

## Vedlegg A: Tabell over sikkerhetstrusler og -fordeler

Tabellen oppsummerer sikkerhetstrusler ved bruk av åpen kildekode, slik de er kommet til uttrykk i intervjuene vi har gjort og slik de går igjen i litteraturen, med referanse til hvor de er hentet fra og hvor de er beskrevet i rapporten. I tabellen har vi også tatt med en kolonne som framhever de positive sidene ved åpen kildekode.

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
<b>Lisenskostnader og andre kostnader</b>			
	Redundans og backup-løsninger er lettere å få til når man ikke må betale lisens for hver installasjon	Intro. kap.4	
	Ingen ekstra lisenskostnader ved å spre programvaren på flere servere. Det kan av sikkerhetsgrunner være nyttig å isolere tjenester fra hverandre.	B.2.5 (Driftsorg.)	
	Man slipper å forholde seg til lisenskoder ved restart av maskiner eller applikasjoner som har hatt driftsstans.	B.2.3 (Driftsorg.)	
Tar ikke høyde for at det også følger kostnader med åpen kildekode, til drift, vedlikehold, videreutvikling, etc. Tror man at slik programvare er gratis, er man mindre kritisk til kvalitet og stabilitet		B.2.4	
	Det er lavere terskel for å teste ut program med åpen kildekode, da man ikke nødvendigvis må forplikte seg til å kjøpe programmet før man installerer det.	B.2.3	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
	Man unngår problemet med å til enhver tid skulle ha oversikt over antall brukerlisenser man benytter.	Avsn. 4.2.2	
<b>Trygghet med en leverandør i ryggen</b>			
Utrygghet i ikke å ha noen å ringe til 24 timer i døgnet når problem oppstår, Manglende stabilitet på leverandørsiden.		Avsn. 4.1.2 (Beaumont) Intro avsn. 4.2 B.2.1 (Rikshospitalet) B.2.4, B.2.5	Å vite at de som utviklet programvaren også er de som retter den, øker følelsen av sikkerhet. Det er mulig å kjøpe supportavtaler også for åpen kildekode. Det er ikke stor forskjell på åpen og lukket programvare når det gjelder support. (B.2.2)
Kan ikke være 100 % sikker å at man laster ned kode fra en autentisk side.		B.2.5 (Pilotprosj.)	Sertifikater og digital signatur kan være en løsning.
<b>Kunnskap/kjennskap</b>			
Mangel på interne IT-ressurser og ekspertise hvis problem oppstår		Avsn. 4.1.3 (Quebec)	
Mangel på personell med (Linux-) kompetanse		B.2.1 (Helse Nord IKT)	Men eget personell tilegner seg nødvendig kompetanse raskt.
Mangel på pålitelig informasjon om åpen kildekodeprodukt (dvs. usikker på om man velger det med best kvalitet)		Avsn. 4.1.3 (Quebec)	
Tidkrevende å sette seg inn i kode fra andre for å forstå den fullt ut. Udfordring å vedlikeholde kode som andre har skrevet.		B.2.6 (Leverandørene)	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
Usikker på om man bruker koden riktig.	Det følger ofte med eksempelkode.	B.2.5	
Utryggheten i å ta i bruk noe man ikke kjenner.		B.2.1 (Driftsmiljøene)	
Vanskelig å sette seg inn i de ulike lisensene og forstå konsekvensen av å bruke dem		B.2.4 (Leverandørene)	
<b>Kvalitet</b>			
	Bedre tid til å kvalitetssikre koden, til testing og til å utvikle sikkerhetsfunksjonalitet.	B.2.3 Payne [14]	Drives ikke av "time-to-market"
Dårlig testing av kode som kunden tar inn i annen programvare, kundene mangler et helhetlig testmiljø.		B.2.6 (Leverandørene) Payne [14]	Når helhetlig testmiljø blir tilgjengelig for åpen kildekode-miljøene blir det bedre.
Mangelfull fokus på sikkerhet			
Stabiliteten kan variere		B.2.4	
Bidrag fra mange utviklere fører til manglende oversikt over helheten		Overly [4]	
<b>Raskere rettinger. Egne endringer og tilpasninger.</b>			
	Raskere utvikling, raskere feilretting. Feil oppdages raskere.	B.2.2, B.2.3 B.2.5 (Pilotprosj.) Witten [16]	
	Oppgraderinger varsles gjennom varslingslister o.l.	B.2.2	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
	Mulig å gjøre feilrettinger selv. Kan gjøre egne feilrettinger og tette sikkerhetshull mens man venter på patcher. Mulig å også få patcher fra andre enn leverandøren.	B.2.1, B.2.3, B.2.5 (Driftsorg.) (Helsenettet) Hopeman [15]	
	Større mulighet for å få videreutviklet programvaren etter egne behov og til egne system. Lett å få til egne endringer.	B.2.3, B.2.5 (Driftsorg.) Payne [14] Hoepman [15]	
	Kan la være å installere moduler/komponenter man ikke ønsker, av sikkerhetsårsaker eller av andre grunner.	B.2.3 (Driftsorg.) B.2.5	Mindre kompleksitet, lettere å holde oversikt og kontroll over programvaren.
Tilgang til koden kan gjøre det mulig for "utro tjenere" hos kunde/bruker å modifisere koden.		B.2.5 (Leverandører)	
Lettere å plante ondsinnet kode	Utvikler eksponerer seg selv, kan ikke skjule seg bak et firmanavn. Det gjør det vanskeligere for utviklere av åpen kildekode å legge inn bakdører eller ondsinnet kode. Lettere å avsløre ondsinnet kode	B.2.5 Payne [14]	
	Mulig å kompilere koden selv	Thompson [18]	
	Skriver enklere og klarere kode pga. offentliggjøring av utviklers navn	Hoepman [15]	
<b>"Many eyeballs" – fordel eller trussel?</b>			
	Større brukergruppe som lettere oppdager feil. "Many eyeballs – all bugs are shallow."	B.2.3 Raymond [6]	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
Alle som ser på koden kan finne feil og svakheter for å utnytte disse.	Alle kan også finne feil og svakheter for å rette disse.	B.2.3	Man trenger ikke kildekoden for å finne feil og svakheter. Man kan gjøre ascii-dump av programmet eller rekonstruere kildekode utfra kjørbare versjon. Nødvendig informasjon for å bryte seg inn i et program finnes også i den kjørbare binærkoden, "disassemblers" og "decompilers" kan raskt trekke ut det nødvendige fra den eksekverbare versjonen
	Ved offentliggjøring av sårbarheter er det mange som kan bidra til feilretting	Witten [16]	
Enklere å lage ondsinnede kopier			Men hvis noen bevisst planter ondsinnet kode i <i>proprietære</i> program, kan det være svært vanskelig å oppdage. Leverandører tester for programmeringsfeil o.l., og ikke ondsinnet programvare.
Det synliggjøres hvordan sikkerheten er implementert. Angripere kan lettere oppdage svakheter/sårbarheter og alvorlige logiske feil	Tilgjengelig kode viser om sikkerhetsløsningene er bedre. Åpning av kode gjør brukerne mer informert om sikkerheten i systemet	B.2.5 Payne [14] Hoepman [15]	
	Uavhengig bedømmelse av sårbarheter	Hoepman [15]	
Manglende systematisk gjennomgang av koden for å finne feil		Viega [13] Anderson [17]	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
Ingen garanti for at kvalifiserte personer undersøker koden for å rette feil.		Payne [14]	
De som ønsker å utnytte eventuelle svakheter i koden har fri tilgang til å utforske den.	At kode blir mye "hacket" (angrepet) er et kvalitetsstempel: Det medfører tilsvarende mye retting og forbedring av koden.	B.2.5	
Ujevn kamp? Hackere trenger bare å finne én feil for å angripe systemet, mens de som skal ta vare på sikkerheta må finne og rette <i>alle</i> feilene og svakhetene.		Hoepman [15]	
Ved mindre system: Ikke mange nok som tester ut koden og søker etter feil.		B.2.4, B.2.6	
Nisjepreget: Helse-IT er et lite miljø. Enda mer nisjepreget: Tilpasset norske forhold.		B.2.4, B.2.6	
Små prosjekt med få utviklere, kanskje bare én. Bruker kanskje et sjeldent programmeringsspråk → ikke mange som <i>kan</i> se på koden, som forstår den.			Dette kan også være tilfelle for proprietær programvare, men der vil det vanligvis være et system for kvalitetssikring
<b>Ansvar</b>			
Mangel på ansvarlig tredjepart.		Avsn. 4.1.3 (Quebec)	

Sikkerhetstrusler	Fordeler	Kilder	Kommentarer
<p>Uklare ansvarsforhold ved feilsituasjoner som oppstår når man tar inn åpen kildekode i leverandørens programvare, hvem har i tilfelle ansvar for å detektere og rette feil?</p> <p>Vanskelig og tidkrevende å gå gjennom koden systematisk.</p>		<p>B.2.1, B.2.6 (Leverandørene) B.2.5</p>	
<p>Programvare som "dør": Man vet aldri hvor lenge det vil stå en organisasjon bak og om koden blir vedlikeholdt og oppgradert.</p> <p>Ofte står det bare ei lita gruppe eller enkeltpersoner bak, mange prosjekt er små og blir ikke vedlikeholdt.</p> <p>Manglende ansvar for å forvalte programvaren videre.</p> <p>Vanskelig å få en forpliktelse om ny leveranse, om ny versjon, oppgradering etc.</p>		<p>B.2.4 (Leverandørene)</p>	<p>Programvare som "dør" er også en trussel for lukket kode. Dersom koden er tilgjengelig er det i det minst en mulighet for at noen kan ta ansvaret for den videre.</p>
<p>Åpen kildekode blir ikke vedlikeholdt i takt med endringer i programvare den skal ha et grensesnitt mot.</p>		<p>B.2.4 (Leverandørene)</p>	

## Vedlegg B: Intervju

Vi har valgt å ikke skrive ut intervjuene i detalj, og heller ikke sitere eksplisitt hvem som har sagt hva. I kapittel 4 har vi gjengitt mer generelt de svarene vi fikk fra ulike typer aktører. I dette vedlegget gjengir vi spørsmålene som var grunnlag for intervjuene/samtalene, samt en liste over hvem vi har snakket med.

### B.1 Intervjuguide

#### Bruk

Bruker dere åpen kildekode utviklet av andre?

Hvis ja, hvilken type programvare bruker dere? Usynlige horisontale infrastrukturensystem (som f.eks. OS), omgivelser for utvikling, biblioteker, desktop-applikasjoner og/eller annet?

Bygger dere i tilfelle videre på kode som andre har utviklet, ev. inkluderer dere kode i form av biblioteksrutiner i egenutviklet kode?

Bidrar dere til andres utviklingsprosjekt?

Lager dere egne/nye program som dere gir ut som åpen kildekode?

Erfaringer med bruk av åpen kildekode?

#### Anskaffelse, oppgradering, vedlikehold

Hvordan finner dere/får dere tak i den åpne kildekoden som dere bruker?

Hva med oppgradering og vedlikehold av slik programvare?

Og hva med support?

#### Holdninger

Hvilke fordeler ser du for deg ved bruk av åpen kildekode?

Hvilke ulemper (trusler) ser du for deg ved bruk av åpen kildekode?

Ser du noen sikkerhetsmessige ulemper eller fordeler ved å bruke åpen kildekode?

Er det andre moment du mener det er viktig å ta hensyn til når man vurderer å ta i bruk åpen kildekode?

Er det forskjell på å bruke åpen kildekode i helsesektoren i forhold til i andre sammenhenger?

### B.2 Hvem som ble intervjuet

Prosjektmedarbeiderne Eva Henriksen og Eva Skipenes gjennomførte alle intervjuene. De fleste intervjuene ble gjennomført i fysiske møter. Et fåtall foregikk pr. telefon. Vi har gruppert intervjuene i kategoriene helsevesenet, driftsorganisasjoner (inkl. helsenettet), leverandører, forskere/utviklere/pilotprosjekt, og tjenesteleverandører. Tabellen gir en oversikt over hvem som ble intervjuet (organisasjon og person), og når intervjuet ble gjort. Vi har også tatt med en link til organisasjonens web-sider. *(Alle nettsider er sjekket 09.10.08.)*

Organisasjon	Beskrivelse	Personer	Dato
<b>Helsevesenet</b>			
Nukleærmedisinsk avdeling, UNN HF	<a href="http://www.unn.no/category9492.html">http://www.unn.no/category9492.html</a>	Erik Traasdahl (overlege og sivilingeniør)	09.09.08
<b>Driftsorganisasjoner</b>			
Helse Nord IKT	Totalleverandør av IKT-tjenester til alle helseforetak i Helse Nord. <a href="http://www.helse-nord.no/category12573.html">http://www.helse-nord.no/category12573.html</a>	Tor André Skjelbakken, Terje Bless	07.05.08 08.05.08
HeMIT	Regional IT-enhet med eierskap og ansvar for sentrale servere, felles programvare og felles infrastruktur i Helse Midt-Norge. <a href="http://www.hemit.no/templates/StandardMaster_77743.aspx">http://www.hemit.no/templates/StandardMaster_77743.aspx</a>	Per Olav Skjesol, Bård Grødem	20.06.08
Rikshospitalet HF, IT-avdelingen	IT-avdelingen er en serviceavdeling som betjener hele sykehuset med tjenester og kompetanse. <a href="http://www.rikshospitalet.no">http://www.rikshospitalet.no</a>	Sissel Jor	03.06.08
Norsk Helsenett, driftsavdelingen	Norsk Helsenett AS er opprettet for å ivareta behovet for et sikkert og enhetlig kommunikasjonsnettverk mellom aktører i norsk helse- og omsorgssektor. <a href="http://www.nhn.no/">http://www.nhn.no/</a>	Asbjørn Andersen, Rune Hætta, Roger Jørgensen	13.05.08
<b>Leverandører</b>			
Visma Unique AS	Leverer virksomhetskritiske løsninger til offentlig sektor. Løsningene forenkler administrative prosesser og arbeidsprosesser knyttet til tjenesteproduksjon og leveres primært til kommuner, fylkeskommuner og til helsesektoren. <a href="http://visma.no/index.asp">http://visma.no/index.asp</a>	Henk Braak	02.06.08
CSAM International AS	Klinisk informasjonsselskap som tilbyr en teknologiplattform som gir helsepersonell tilgang til all relevant klinisk informasjon i og mellom sykehus i forbindelse med behandling av pasienter. <a href="http://www.csam.no/CSAM/index.htm">http://www.csam.no/CSAM/index.htm</a>	Håkon Haugtomt, Kjetil Sanders	03.06.08
RisCo AS	Leverandør av åpne prosessstøttesystem innenfor radiologi <a href="http://www.risco.no/">http://www.risco.no/</a>	Svein Harald Utgård	14.05.08
Well Diagnostiscs	Well tilbyr tjenester som bidrar til samhandling i helsesektoren <a href="http://well.no/">http://well.no/</a> (Well er kjøpt opp av DIPS ASA, og i september 2008 også sammenslått med dem organisatorisk.)	Yngve Nyheim	21.05.08

Organisasjon	Beskrivelse	Personer	Dato
DIPS ASA	Største leverandør av system for elektronisk pasientjournal til sykehusene i Norge. <a href="http://www2.dips.no/">http://www2.dips.no/</a>	Sigurd From	17.06.08
Infodoc as	En betydelig leverandør av datasystem for helsesektoren i Norge. Produktene spenner fra journal og timebok for en liten solopraksis til allsidige løsninger for institusjoner og store helsesentre. <a href="http://www.infodoc.no/">http://www.infodoc.no/</a>	Endre Dyrøy	06.08.08
<b>Pilotprosjekt (forskere/utviklere)</b>			
PasientLink NST	Prosjektet utviklet og prøvde ut sikker kommunikasjon mellom pasient og fastlege over Internett i 2002-2003. <a href="http://www.telemet.no/pasientlink.7457.no.html">http://www.telemet.no/pasientlink.7457.no.html</a>	Per Egil Kummervold	07.05.08
Diabetes/Lifestyles TTL NST	Develop self-help ICT-supported tools for motivation both to change and maintain a healthy lifestyle for the individual, enhanced by health sensor monitoring, sensor data analyzes, information presentation in a self-help learning environment, in peer network learning environment and in health professional supported applications. <a href="http://www.telemet.no/the-diabetes-ict-health-motivation-project-2008-2010.4467424-51255.html">http://www.telemet.no/the-diabetes-ict-health-motivation-project-2008-2010.4467424-51255.html</a>	Ragnhild Varmedal, Eirik Årsand	13.05.08
MinHelsestasjon TTL (Norut og NST)	Personal healthcare technology and services for elderly chronically ill. The project has designed and developed a prototype of a system for home based training, self-management and following-up of patients/chronically ill over broadband networks. <a href="http://www.telemet.no/myhealthservice-personal-healthcare-technology-and-services-for-elderly-chronically-ill.448999-51255.html">http://www.telemet.no/myhealthservice-personal-healthcare-technology-and-services-for-elderly-chronically-ill.448999-51255.html</a>	Lars K. Vognild, Luiz Fernandez Luque og Njål Borch, Norut. Tatjana Burkow og Trine Krogstad, NST, Anders Baardsgaard, Norsk Helsenet	15.05.08
Melanom-prosjektet TTL NST	Målet med prosjektet er deteksjon av maligne melanomer basert på føflekkbilder. Prosjektet utvikler analysemetoder som er egnet til fullt automatiserte og skalerbare differensielle system for statistisk inferens, med formålet diagnostisk beslutningstaking. <a href="http://www.telemet.no/deteksjon-av-maligne-melanomaer-basert-paa-foeflekkbilder.4487574-51252.html">http://www.telemet.no/deteksjon-av-maligne-melanomaer-basert-paa-foeflekkbilder.4487574-51252.html</a>	Kevin Thon Vedad Hadziavdic	17.06.08

Organisasjon	Beskrivelse	Personer	Dato
SNOW-prosjektet TTL NST	Symptombasert helseovervåking i Helse Nord. Prosjektet ønsker å undersøke om deling av epidemiologiske data mellom primærleger endrer klinisk praksis med hensyn til testing for, diagnostisering og behandling av smittsomme sykdommer. <a href="http://www.telemet.no/symptombasert-sykdomsovervaakning-i-helse-nord.448019-72584.html">http://www.telemet.no/symptombasert-sykdomsovervaakning-i-helse-nord.448019-72584.html</a>	Johan Gustav Bellika, Per Atle Bakkevold, Lars Ilebrekke	18.09.08
<b>Tjenestetilbydere</b>			
Helsekompetanse.no NST	Nasjonal læringsportal for helse-Norge. Her kan man utvikle egne nettbaserte kompetansetilbud, eller delta i kurs som er utviklet. <a href="http://www.helsekompetanse.no/">http://www.helsekompetanse.no/</a>	Vegard A. Johansen	07.05.08

## B.3 Sammendrag av synspunkt i intervjuene

### B.3.1 Utbredelse av åpen kildekode i helsevesenet

Vårt inntrykk er at helsevesenet i liten grad benytter åpen kildekode på applikasjonssiden, verken når det gjelder kontorstøttesystem eller fagapplikasjoner. På serversiden, derimot, har man i varierende grad tatt i bruke åpen kildekode.

Våre intervjuobjekt kommer hovedsakelig fra driftsorganisasjoner og leverandører innen helsesektoren, i tillegg til forskere og utviklere i pilotprosjekt. Vi har i liten grad intervjuet helsepersonell, da vi ikke har kjennskap til mange som både er helsepersonell og har tilstrekkelig kunnskap rundt problematikken knyttet til åpen kildekode for helseapplikasjoner, eller som kjenner til de applikasjonene som er tilgjengelig som åpen kildekode.

Det finnes imidlertid noen pådrivere blant helsepersonell, blant annet innen bildediagnostikk/radiologi, og sikkert også innen andre områder. Som nevnt tidligere (avsnitt 4.1.2), har de ved Beaumont Hospital i Dublin tatt i bruk et vidt spekter av system med åpen kildekode. I tillegg har de selv utviklet en åpen kildekode-løsning for å finne fram og vise digitale røntgenbilder. Når det gjelder programvare for medisinsk bildebehandling har man den fordel at det finnes en åpen standard – DICOM, som de fleste leverandører forholder seg til. Dermed ligger det godt til rette for å lage nye applikasjoner eller tilleggsapplikasjoner i åpen kildekode. Det finnes bl.a. et åpent kildekodeprosjekt som utvikler et avansert bildeframvisningsprogram for Mac OS X, som kommer i samme kategori som det proprietære systemet Agfa IMPAX. Systemet kalles **OsiriX**<sup>38</sup>. (For mer informasjon se vedlegg C.)

Sykehusene har tradisjon for å kjøpe programvare fra kjente kommersielle leverandører. Noen mener at **driftsmiljøene** innen helsevesenet er redd for å prøve nye ting. Når de aller fleste bruker Microsoft Office og Windows, er det lett å tenke at alt annet også må være basert på Microsoft-produkt for at det skal virke i det samme driftsmiljøet. Ved å velge ett

<sup>38</sup> <http://www.osirix-viewer.com/>

produkt fra en leverandør, tvinges man dessuten ofte til å velge flere produkt fra samme leverandør. Noen av helseforetakene bruker f.eks. Microsoft terminalserver fordi denne viste seg å være best egnet til formålet. De andre alternativene hadde problem med printerløsning etc. Men når man bruker Microsoft terminalserver må man også ha Microsofts AD-server (Directory).

Et vanlig argument for *ikke* å bruke åpen kildekode er at det er tryggere å bruke noe man kjenner. Når driftsavdelingene utvikler egen kode, bygger de som regel ikke på åpen kildekode, bl.a. pga. behovet for samspill med andre system. De bruker som regel Microsoft sine utviklingsverktøy for egen utvikling. Gjenkjennbarhet er også et poeng. Det blir enklere for brukerne hvis nye tjenester og applikasjoner har det samme grensesnittet som andre system de kjenner. I tillegg er det viktig at løsningene tilfredsstiller gjeldende sikkerhetskrav, og at man har leverandører å tilkalle når det oppstår problem.

Det ansettes stort sett heller ikke personer med Linux-kompetanse i driftsavdelingene i helsesektoren. En bekymring i Helse Nord IKT ved overgang til Linux var at man ikke hadde tilstrekkelig personell med Linux-kompetanse. Erfaringen man gjorde seg, var imidlertid at personell med god kompetanse på Microsoft-produktene raskt tilegnet seg nødvendig kompetanse på Linux selv om de ikke hadde tid til å gå på kurs.

I Helse Nord IKT bruker driftsavdelingen en god del åpen kildekode-servere og -verktøy. De er bl.a. i ferd med å gå over til Linux for kjøring av DIPS-databasen (databasen til journalsystemet), da Linux har vist seg å være bedre egnet til dette enn Microsoft-produktene de har brukt hittil. Selve applikasjonen (DIPS) er Windows-basert.

Helse Nord's felles publiseringssystem for web kjører også på Linux. Mange av komponentene i dette systemet er implementert i PHP (åpen kildekode), men selve applikasjonen er proprietær. En del samhandlingssystem driftes også på Linux. Helse Nord bruker også åpen kildekode driftsverktøy for statistikk og overvåkning av ytelse, belastning og oppetid (lisensiert under GPL). I samarbeid med et av helseforetakene tester de ut åpen kildekode-løsninger for PACS, både bildearkiveringsfunksjonen og bildevisningsfunksjonen, inkludert noe RIS-funksjonalitet. Agfa har dyre innebygde lisensavtaler for programvare fra andre leverandører, og man har liten frihet med tanke på oppgraderinger etc. med løsningen fra Agfa. Bildeframvisningsprogrammet basert på åpen kildekode (OsiriX) har bedre funksjonalitet enn den proprietære løsningen, og den åpne kildekode-løsningen for arkivering og uthenting av medisinske bilder (**Dcm4Che**<sup>39</sup>) har dessuten mye bedre tilgangskontroll og logging på brukernivå enn den proprietære løsningen som er i bruk.

Det samme helseforetaket har også utviklet programvare for en termografilab basert på åpen kildekode DICOM-komponenter. Dette systemet er i daglig bruk ved helseforetaket.

I motsetning til driftsavdelingen i Helse Nord IKT, forholder deres utviklingsavdeling seg stort sett til Microsoft-produkt. De regner med at Linux gradvis vil ta over, da det viser seg at Linux gir bedre ytelse og tilgjengelighet.

Helse Midt-Norge har stort sett bare Microsoft- eller Unix-servere eller stormaskiner. De har noe "shareware" på verktøysida, men synes det er enklere å drifte en felles plattform enn å forholde seg til mange forskjellige. Holdningen er å bruke det som er mest hensiktsmessig økonomisk og kvalitetsmessig.

På Rikshospitalet HF benytter de i liten grad åpen kildekode, men kunne ønsket å gjøre det mer, bl.a. pga. store kostnader forbundet med proprietære system. Men for systemene som

---

<sup>39</sup> <http://www.dcm4che.org/>

brukes i pasientbehandlingen anses det som svært viktig å ha en leverandør som kan tilkalles for å rette feil 24 timer i døgnet. De bruker mye Suse-Linux fra Novell. De ønsker å teste ut mer åpen kildekode, f.eks. Open Office, i første omgang blant forskerne som også er ansatt på universitetet i Oslo, men mangler foreløpig finansiering for dette.

Norsk Helsenett har en grunnholdning om at det skal brukes åpen kildekode der det er mulig, bl.a. fordi det da er lett å få til egne endringer. Serverparken kjører primært på Linux (Ubuntu og RedHat) og FreeBSD. For tjenestene som kjører på disse plattformene, f.eks. e-post, EDI, DNS og proxy'er for internett, brukes åpen kildekode på serversiden. Klientene hos brukerne er proprietære. De bruker ellers Apache og lighttpd for web-servere, og rammeverket Rails for web-utvikling.

**Leverandørene** av helseapplikasjoner lever av å selge applikasjoner og verktøy, og ikke av å selge tjenester. Lisensproblematikken blir da en utfordring med hensyn til å skulle ta inn åpen kildekode fra andre i egenutviklet kode. En god del av den aktuelle typen kode er lisensiert under GPL, med de konsekvenser det medfører ved integrering med egenprodusert kode. Driftsorganisasjoner som leverer tjenester kan integrere kode lisensiert under GPL med egenutviklet kode, og fordi programvaren brukes internt trenger de ikke å utgi den egenutviklede koden som åpen kode. Leverandører som selger produkt (applikasjoner og verktøy) derimot, tvinges til å legge ut egne bedriftshemmeligheter ("arvesølvet") som åpen kode som konkurrenter kan få gratis tilgang til, dersom de integrerer kode lisensiert under GPL i egenprodusert kode. De er skeptiske til å gi ut koden, da de er redde for at dette vil underminere eksistensgrunnlaget deres. Leverandørene benytter heller åpen kildekode som er lisensiert under Apache, BSD eller lignende lisensstyper dersom de tar inn åpen kildekode i egen kode.

En annen utfordring med helsesektoren er de store kravene de har til integrasjon og individuelle tilpasninger av løsningene. Utstyr sykehusene kjøper har i stadig større grad basert seg på et HL7-grensesnitt for interagering mot andre system. Dette gir utfordringer ved behov for interagering med de systemene som *ikke* har et HL7-grensesnitt. (HL7 er en internasjonal/amerikansk organisasjon som utvikler standarder for datautveksling og samhandling i helsesektoren.) Ved i større grad å bruke HL7-standarder i de norske fagapplikasjonene, vil muligheten for å ta i bruk åpen kildekode fra internasjonale miljø i helsesektoren også bli større.

En annen grunn til at leverandørene er skeptiske til å bygge videre på eller inkludere åpen kildekode i egen programvare, er at dette fort kan medføre uklare ansvarsforhold ved feilsituasjoner: Ligger feilen i den åpne kildekoden, og i tilfelle hvor, eller i leverandørens egen kode – og hvem har i tilfelle ansvar for å detektere og rette feilen? Noen tar likevel inn små komponenter med åpen kildekode hvor de kan gå gjennom koden og skaffe seg god oversikt over hva den gjør.

De fleste leverandørene benytter en del biblioteksrutiner som er åpen kildekode, f.eks. OpenSSL, for krypteringsformål. Dette dreier seg om godt uttestede løsninger, og løsninger hvor man har god oversikt over funksjonaliteten. En del av de mindre leverandørene benytter også operativsystem og databasesystem som er åpen kildekode.

Mange **forskere og utviklere** som er engasjert i pilotprosjekt er Linux-brukere, og de bruker gjerne utviklingsverktøy som er åpen kildekode. De bruker også åpen kildekode web-browsere, Apache Tomcat og biblioteksrutiner. For krypteringsformål brukes kryptobibliotek som er åpen kildekode.

Noen inkluderer det meste av det de bruker av åpen kildekode i form av biblioteksrutiner. Andre tilpasser applikasjoner med åpen kildekode til eget bruk og for integrering mot andre system (customization of code).

De som utvikler løsninger for mobiltelefoner gjør dette på Windows Mobile, da det ikke finnes gode nok alternativer med åpen kildekode. De aller fleste mobiltelefoner har proprietære operativsystem. Noen har et åpent programmeringsgrensesnitt slik at man kan programmere på mobiltelefonene. Det har kommet mobiltelefoner med Linux OS, men disse har ikke slått an i markedet ennå og er derfor lite tilgjengelig. Programvaren for mobiltelefoner er sterkt kontrollert av mobiltelefonprodusentene. Windows Mobile gir den største friheten for programmerere i dag (bedre enn Symbian OS). Ved utvikling av nye løsninger for mobiltelefoner må man benytte telefoner som fungerer på det mobilnettet man har og der markedet er. Men åpen kildekode-alternativene for *tilleggsprogramvare* er gjerne bedre enn de proprietære, også innen mobiltelefoni.

Ett av pilotprosjektene valgte likevel å bruke åpen kildekode for utvikling av en løsning for mobile enheter fordi den gav mye større fleksibilitet, selv om det ble mye mer arbeid. De "lukkede" og mer kommersielt utbredte plattformene har mange tilhørende program og utviklingsverktøy. Det er derfor lettere å raskt få opp en demonstrator ved å benytte kommersielle utviklingsverktøy for å designe brukergrensesnittene og koble programvare sammen.

### *B.3.2 Support av åpen kildekode*

Det er mulig å kjøpe supportavtaler også for produkt hvor kildekoden er åpen. Man kan også søke på nettet etter andres erfaringer, og være medlem av varslingslister for endringer og oppgraderinger. De som benytter åpen kildekode opplever ikke veldig stor forskjell i support fra leverandører av proprietær programvare og åpen kildekode. Den beste supporten får de i en del tilfelle fra åpen kildekode-miljøet på internett. Feilretting gjøres ofte raskere for åpen kildekode enn proprietær kode. De kjøper også i en del tilfelle oppgraderinger og endringer fra leverandører av åpen kildekode.

Utviklerne som benytter åpen kildekode, opplever det som positivt at det ofte følger med eksempler på hvordan man skal bruke koden, f.eks. hvordan man skal lage plug-ins.

Mange søkte på nettet (Google, SourceForge, etc.) for å finne relevant åpen kildekode, eller de fikk tips fra andre som kjente til programvaren fra før av. Leverandørene er ute etter stabile versjoner av åpen kildekode som i mindre grad trenger oppgraderinger. De bruker ikke de nyeste (og mer ustabile) versjonene av koden.

Oppgraderinger varsles gjerne gjennom varslingslister. Noe programvare sier selv fra når det foreligger oppgraderinger. Andre ganger ser man at nye oppdateringer eller lignende er lagt ut på nettsidene til de som leverer koden. Ellers leter man etter slikt bare når man støter på problem selv. Hvis kode/moduler er integrert i annen kode på en god måte, er det enkelt å oppgradere med nye versjoner. I noen tilfelle må man gjøre oppgradering og vedlikehold selv.

### *B.3.3 Fordeler med å bruke åpen kildekode*

De aller fleste sier at åpen kildekode gir større fleksibilitet. I det ligger det mange argument:

En fordel med åpen kildekode er at det gjerne er en lavere terskel for å teste ut slike program, da man ikke nødvendigvis må forplikte seg til å kjøpe programmet før man installerer det. Åpen kildekode medfører ofte raskere utvikling av programvaren og raskere

retting av feil. Man får også større mulighet for å få videreutviklet systemet etter egne behov. En av våre informanter tror programvaren blir bedre ved denne utviklingsmåten, i og med at man får en større brukergruppe som kan finne feil. De fleste antar at bruk av åpen kildekode blir billigere fordi man slipper lisenskostnadene. Kostnadene ved åpen kildekode er ofte knyttet til support og oppgraderings- og videreutviklingsavtaler.

**Driftsorganisasjonene** mener det kan være mye å spare på å kunne få velge hvilken HW de skal kjøre applikasjonene på. Dette er stort sett enklere med åpen kildekode enn med proprietær programvare. I tillegg kan man som regel velge hvilke moduler/komponenter man vil installere fra den åpne kildekoden, slik at man kan klare seg med mindre HW-ressurser. Man kan også la være å installere moduler/komponenter man ikke ønsker, av sikkerhetsårsaker eller av andre grunner.

Driftsorganisasjonene har svært positive erfaringer med overgang til Linux, bl.a. når det gjelder stabilitet, driftssikkerhet og håndtering av lisenser. En annen stor fordel med åpen kildekode er at man slipper å holde oversikt over antall lisenser man har betalt for og antall personer som benytter den aktuelle programvaren. Man slipper også å forholde seg til lisens-koder ved restart av maskiner eller program som har hatt driftsstans.

Hvis man har tilgang til kildekoden, er det mulig å gjøre endringer og feilrettinger selv i stedet for å måtte vente på at leverandøren eventuelt gjør det man ønsker, og det kan være lettere å finne ut hva som er årsakene til feil som oppstår.

Det er mange ulike meninger om fordeler og ulemper ved bruk av åpen kildekode. En fordel kan være at utviklerne må signere koden med sitt eget navn og på den måten eksponere seg og arbeidet sitt for hele verden – i stedet for å "skjule seg" bak et leverandørnavn. Dermed blir det mer om å gjøre å lage kode av god kvalitet, selv om man ikke har noen garanti for at dette skjer.

En annen fordel er at de som utvikler åpen kildekode ikke nødvendigvis drives av "time-to-market", og derfor har bedre tid til å kvalitetssikre koden.

Ved å produsere en tjeneste på en plattform som er åpen kildekode, kan man komme raskere på markedet enn hvis man må utvikle plattformen selv. Man får også endringene som alle andre bidragsytere kommer med.

**Pilotprosjektene** har svært positiv erfaring med å benytte åpen kildekode. De opplever at det er mye enklere å tilpasse ulike program til hverandre når koden er åpen (det kan være vanskelig å bygge videre på lukket kode.). En annen stor fordel er at man slipper å bruke ressurser på selv å utvikle kode som er utviklet minst like godt av andre.

Hos tjenesteleverandøren vi intervjuet ble prisen ut til brukerne drastisk redusert da de gikk over til åpen kildekode, fordi de slipper å betale dyre årlige lisenser for programvaren.

### *B.3.4 utfordringer med bruk av åpen kildekode*

Bruk av åpen kildekode krever gjerne at man har en viss kompetanse i egen organisasjon til å vedlikeholde systemet. Mange forbinder åpen kildekode med gratis programvare<sup>40</sup>. Dette kan bidra til at de som vurderer å ta det i bruk, ikke i stor nok grad tar høyde for at det er kostnader knyttet til å benytte slik programvare også, i form av egne ressurser til drift, vedlikehold og videreutvikling, og til support- og driftsavtaler med eksterne. Det at man tror

---

<sup>40</sup> "Åpen kildekode tiltrekker seg gjerrigknarker", som en av våre informanter sa.

bruk av slik programvare er gratis, kan kanskje også føre til at man er litt mindre kritisk når man vurderer kvaliteten og stabiliteten av programvaren.

Man er også avhengig av vitale miljø som kan holde programmet levende og videreutvikle det. For mindre system kan det kanskje være større risiko for feil og sårbarheter, da det ikke nødvendigvis er mange nok som er interessert i å teste ut koden og søke etter feil.

Når det gjelder bruk av åpen kildekode ute blant brukerne, er det en del andre utfordringer som melder seg. Det er stor enighet om at Microsoft i dag er best hvis man trenger sentralisert drift. Det vil være svært vanskelig, om ikke umulig, å drifte et stort antall pc-er hvis man benytter f.eks. Mac OS eller Linux. Linux er mye vanskeligere å drifte/administrere sentralt enn Microsoft.

En annen type utfordring er at helsevesenet har høye krav til stabilitet, driftskvalitet og sikkerhet. Åpen kildekode er ikke alltid utviklet med tanke på dette, stabiliteten kan variere. Man må være sikker på at man har veltestede system som også er stabile ved oppgraderinger. Man må óg ha stabilitet på leverandørsida. Plassering av ansvar blir også en utfordring. Noen må ha ansvar for å forvalte programvaren videre, bl.a. ved å ta vare på tilbakemeldinger og legge ut nye løsninger/versjoner. Hvis man benytter åpen kildekode kan man ikke skyldes på noen hvis noe går galt. Man må selv ta ansvar for egne data, og man må kunne mer for å være sikker på at programvaren fungerer slik man ønsker.

I dag har ikke helsevesenet økonomisk insentiv til å utvikle åpen kildekode som skal deles med de andre aktørene i sektoren. Det ligger ikke noen gevinst i det for dem som ønsker å ta initiativ til å utvikle programvare som kan være felles.

Det er også utfordringer rundt vedlikehold av denne type programvare. På internett er det et stort "community" som tar vare på dette for mer generell programvare, men i et lite miljø som helse-IT, er det mye verre å få til slike mekanismer. Systemene innen helsevesenet i Norge er i stor grad tilpasset norske forhold, både når det gjelder reguleringer og språk. Hvis man får system som i større grad tar i bruk medisinskfaglige støttesystem, bør man kanskje også se til utenlandske leverandører.

**Leverandørene** mener det er en ulempe at man ikke vil kunne gi kunden de samme garantier når det gjelder åpen kildekode som blir tatt inn i egenprodusert kode. Man vet aldri hvor lenge det vil stå en organisasjon bak og om koden blir vedlikeholdt og oppgradert. Det kan være vanskelig å få en forpliktelse om ny leveranse, om ny versjon, oppgradering etc. ved bruk av åpen kildekode. Ofte opplever man at det bare står ei lita gruppe eller enkeltpersoner bak produktene som gis ut som åpen kildekode, og det er vanskelig å få nødvendig support. Man blir henvist til å videreutvikle og rette ting selv.

Åpen kildekode bør ikke brukes ukritisk. Mange av åpen kildekode-prosjektene er små og blir ikke vedlikeholdt. Å ta inn programvare som "dør", kan være et større problem med åpen kildekode enn ved proprietær programvare. Hvis åpen kode ikke blir oppdatert lenger, har det ofte en grunn. Da oppstår det gjerne nye og bedre program, og det er lurt å gå over til de nye. Det foregår en naturlig utvelgelse av det som er bra. Ved å velge program som har vært i bruk lenge, vil man kunne sjekke hvor stabile de er og hvor fornøyd andre er med dem.

Man bør også velge programvare som har mer enn én utvikler. Det kommer også an på hvor kritisk komponenten er. Et bibliotek som skal styre en enkelt funksjon, bør lett kunne byttes ut med andre tilsvarende komponenter hvis f.eks. vedlikehold opphører, dvs. man bør vurdere om det finnes flere alternativer.

Noen frykter at åpen kildekode ikke blir vedlikeholdt i takt med endringer i (nye versjoner av) lukket programvare som den skal ha et grensesnitt mot. Dette forekommer imidlertid også for proprietær/lukket kode.

Ved bruk av åpen kildekode-bibliotek kan man ofte velge mellom mange ulike biblioteks-rutiner. Å vite hva som er best å velge, kan være en utfordring. Når man bruker åpen kildekode må man hele tida være "på hugget" og følge med på hva som skjer.

Det er komplisert for en som ikke er programmerer å ta i bruk åpen kildekode, og det kan være vanskelig å sette seg inn i de ulike lisensene og forstå konsekvensen av å bruke dem.

### *B.3.5 Sikkerhetsmessige aspekt*

En sikkerhetsmessig utfordring ved bruk av åpen kildekode er at de som ønsker å utnytte eventuelle svakheter i koden, har fri tilgang til å utforske den.

**Driftsorganisasjonene** mener at det viktigste er at sikkerhetsfunksjonaliteten har god kvalitet. Driftskulturen er mer avgjørende enn om koden er åpen eller ikke; måten man setter opp tjenesten, oppsett av programvaren, konfigurering etc. er i utgangspunktet mest avgjørende. Da er det viktig å velge programvare som har de egenskapene man mener er nødvendig ut fra et sikkerhetsaspekt.

I en del av applikasjonene synliggjøres sikkerheten, dvs. hvordan sikkerheten er implementert. Å legge ut slik kode kan innebære en sikkerhetsrisiko. Da må man heller tilstrebe å legge sikkerheten utenpå koden.

Åpen kildekode gir driftsfolkene mulighet til å finne og tette sikkerhetshull mens de venter på oppgraderinger fra leverandørene. Proprietære leverandører krever ofte å ha så stor kontroll på applikasjonene eller utstyret at det er vanskelig å gjøre sikkerhetsmessige oppgraderinger selv.

Med åpen kildekode kan man i mye større grad begrense hvilke muligheter brukerne får, bl.a. av sikkerhetsgrunner. Man kan fjerne eller la være å installere unødvendige komponenter.

Sporbarheter av endringer er dramatisk mye bedre i den nye delen av Linux-kjernen. Dette gjør det også enklere å stole på systemene og å gjøre "audit" av dem.

En fordel med åpen kildekode på servernivå er at det ikke blir noen ekstra lisenskostnader ved å kjøre koden på mange servere. Før eller senere vil man få innbrudd, og for å hindre at innbrudd i en tjeneste også påvirker andre tjenester, kan det være nyttig å isolere tjenestene fra hverandre. Proprietære program er ofte mer sammenbundet og det er vanskelig å skille ut egne tjenester.

I prinsippet bør det være sikrere å bruke åpne løsninger, fordi hull i slike system kan oppdages raskt og rettes raskt, da man kan sende tilbakemeldinger direkte til utviklerne. I lukkede system oppdages hullene mer tilfeldig og blir ikke nødvendigvis rapportert og rettet.

Når det gjelder helsevesenet, og spesielt sykehusene, er det ønskelig å velge leverandører som er gjennomprøvde og har en organisasjon bak seg som kan tette sikkerhetshull når de oppstår.

Ved integrering av åpen kildekode i egne applikasjoner kan det være en risiko at man bruker komponentene feil, slik at man viser feil skjermbilde eller data om pasientene, eller får et uforutsett resultat.

Noen mener det kan være enklere å lage sikkerhetshull i åpen kode enn i lukket kode hvis man ønsker det. En potensiell risiko er at kode som er levert som åpen kode, kan endres av utro tjenere hos kunden selv. Dette stiller store krav til kundens driftsorganisasjon. Det er nå gått sport i å legge inn skjulte bomber i kode på en slik måte at det ikke er så lett å oppdage dem, f.eks. "easter-eggs" (som er mer godartede "bomber"). Hvis noen har lagt inn ondsinnet kode, hvem skal få skylden? "Leverandøren", kunden eller hvem som helst? Og hvis man tar inn komponenter med åpen kildekode i egen kode, er det vanskelig å garantere for den koden man har tatt inn. Det kan være for tidkrevende å gå detaljert gjennom hele koden. Andre mener det er langt mer risikabelt for utviklere å legge inn bakdører eller ondsinnet kode i åpne løsninger, da utviklerens navn følger med koden.

Firma som skal leve av det, gjør hva de kan for at deres (proprietære) kode skal være sikker.

Et vellykket åpen kildekode-prosjekt som blir mye utbredt blir også mye "hacket". Det kan i seg selv være et kvalitetsstempel, for dette medfører at det blir gjort tilsvarende mye retting og forbedring. Det kan være like sikkert å stole på dette som å stole på at en proprietær leverandør har testet godt nok. Dette gjelder velprøvde prosjekt som er godt utbredt. Da er åpen kildekode bedre.

Personene vi intervjuet fra **pilotprosjektene** mener det ikke er noen særlige sikkerhetsmessige forskjeller på å bruke proprietær programvare kontra åpen kildekode. I noen tilfelle kan det være en fordel å bruke åpen kildekode i og med at den kan være inspisert av flere, og at feil og svakheter kan bli oppdaget og rettet raskere enn for proprietær kode. Man har også mulighet til å sjekke koden selv. Men det som er viktigere er hvor kritisk man er til det man tar i bruk, og at man sjekker kvaliteten og "ryktet" til koden før man bruker den. En annen utfordring er at man ikke alltid kan være helt sikker på at siden man laster ned koden fra er leverandørens eller prosjektets autentiske side.

### *B.3.6 Har helsesektoren spesielle utfordringer mht. programvare?*

Grunnlaget for åpen kildekode-prosjekt er muligens mindre i helsevesenet enn i en del andre sektorer fordi det er færre brukere av de mer spesialiserte programmene; de er gjerne mer nisjepreget enn program for andre sektorer (f.eks. programvare for økonomistyring, kundebehandling, personalbehandling, etc). For mer generelle program bør det ligge godt til rette for bruk av åpen kildekode i helsevesenet i og med at man vil få storbruksfordeler.

Standardene for operativsystem, web-browsere etc. er veldig klart definert. Journalsystem er mye mer komplekse og mindre "standardiserte". Det er derfor en større utfordring å lage en "entydig" standard for et åpen kildekode-journalsystem.

Journalsystem for sykehus har mange ulike applikasjoner som skal spille sammen, og tilgangsstyringen er svært kompleks og dynamisk. Integrasjon mellom de ulike delsystemene er avgjørende for gode arbeidsprosesser og for riktig og oppdatert informasjon om pasienten til enhver tid. Dette stiller store krav til utviklingen av systemene, og til test- og godkjenningprosessen.

Utviklingsmetoden for dagens journalsystem er i stor grad "testdrevet utvikling", dvs. at programutviklingen skal være basert på testing, og på kode man vet kjører slik den skal. Leverandørene bygger etter hvert opp et miljø av tester som skal kjøres etter en viss rutine. Det er ikke så mange av kundene som har slike helhetlige testmiljø. Mens programutvikling tidligere i stor grad innebar å skrive kode, omfatter utvikling nå alt fra kravspesifisering til testing, og det meste kan gjøres ved bruk av utviklingsverktøy.

Hvis et sykehus tar inn kode utenfra etter at de har fått et ferdig testet system fra journalleverandøren, må de selv ta ansvar for dette. Det er ikke gitt at disse programmene vil fungere sammen. Det kan være vanskelig å finne gode nok testmiljø for å kontrollere om den nye koden fungerer tilfredsstillende sammen med det man har fått fra journalleverandøren.

En annen problemstilling som kan oppstå, er i det tilfelle journalleverandøren legger ut kildekoden og andre lager endringer. Hvordan skal leverandøren forholde seg til disse endringene i et vedlikeholdsperspektiv, og hvem skal ta ansvar for eventuelle feil som oppstår pga. endringene?

Mange utviklingsverktøy kommer fra miljøene som driver med åpen kildekode. Men når man skal utvikle kommersielt, er det greit å basere seg på kommersielle verktøy som også omfatter hele utviklingsprosessen. Tar man inn kodesnutter utenfra, må disse tilpasses til utviklingsmiljøet.

Microsoft frigjør noe programvare som åpen kildekode, som de ikke tar noe ansvar for. En av journalleverandørene innen sykehussektoren har tatt inn en slik kodebit i et prosjekt, men må selv vedlikeholde denne koden. Det krever mye tid å sette seg inn i slik kode for å forstå fullt ut hva den gjør. Det er en utfordring å vedlikeholde kode som andre har skrevet. Hvis man tar en kodesnutt fra andre som det ikke går an å lage enkle tester for, vil det være vanskelig å integrere koden i resten programvaren.

Systemdesign innebærer også å definere hvem som har ansvaret for et system og de ulike delene av det over tid. Hvordan skal de ulike delene oppdateres, og hvem har ansvar for hva? Denne utfordringen blir ikke enklere hvis man tar inn kode som andre har skrevet. Enten må man stole på de som har laget koden, eller man må ta ansvaret selv og sette seg grundig inn i koden. Det siste koster mye. De fleste mener derfor at kode som har stor anvendelse og brukes av mange (og dermed er grundigere testet), er lettere å basere seg på enn kode for veldig spesialiserte oppgaver.

En annen utfordring for det norske helsevesenet er at Norge er et lite land med relativt få aktører og et uforutsigbart marked, med til dels uforutsigbare økonomiske rammer. Microsoft er veldig dominerende i helsesektoren, f.eks. i forhold til i bank- og finanssektoren. Dette gjør at det kan være en høy terskel å ta i bruk løsninger basert på åpen kildekode.

Leverandørene må endre drastisk på forretningsmodellene hvis de skal gå over til å levere åpen kildekode. Ingen tør å forlate den måten de jobber på. Hvis noen turte å være først ute med å etablere et åpen kildekode journalsystem, antar de at resten av markedet ville kommet etter.

### *B.3.7 Bruk av åpen kildekode og åpne standarder*

Det er en generell oppfatning både blant **driftsorganisasjonene** og **leverandørene** at det er viktigere med standardiserte driftsmiljø og komponenter og at koden følger åpne standarder, enn å ha tilgang til kildekoden. Leverandørene mener det er mer aktuelt å tilgjengeliggjøre grensesnitt som andre kan utnytte, enn å tilgjengeliggjøre selve koden. Også internt i et journalsystem er det ofte nok å ha kunnskap om grensesnitt man skal forholde seg til. Kildekoden blir stort sett bare brukt av de som jobber med denne delen av koden. DIPS ASA, f.eks., er opptatt av at kodesnutter skal være innkapslet og veldefinert slik at andre ikke trenger gå til kildekoden for å forstå hva koden gjør.

Leverandørene er i noen grad opptatt av at løsningene deres skal kunne kjøres på ulike plattformer, men samtidig kan det være kostbart å tilpasse og vedlikeholde løsningene for

flere plattformer. Kundene etterspør i liten grad åpen kildekode, da de fleste i helsevesenet i stor grad forholder seg til produkt fra Microsoft.

Hvis man skal lage en løsning for samvirke mellom helseforetakene, må man velge åpne løsninger og åpne standarder uten proprietære tillegg. Noen oppfatter at det er svært lite gehør for dette i sektoren.

### *B.3.8 Aktørenes bidrag til utvikling av åpen kildekode*

**Driftsorganisasjonene** bidrar i noen grad til andres utvikling av åpen kildekode ved å sende inn forslag til endringer eller påpeke feil i programvare de benytter. Det er imidlertid vanskelig for ansatte i disse organisasjonene å få satt av tid til å bidra i særlig grad til slik utvikling.

For Norsk Helsenett er det lite aktuelt å legge ut kode de utvikler selv, da den i for stor grad er spesialtilpasset Norsk Helsenetts arkitektur og det norske helsevesens behov.

**Leverandørene** bidrar i svært liten grad til andres utvikling av åpen kildekode, men de gir i noen tilfelle tilbakemelding om ting som ikke fungerer bra nok.

Leverandørene har hittil ikke lagt ut egne program som åpen kildekode. En av de mindre leverandørene vil gi utvalgte partnere tilgang til koden til en del GUI-komponenter, men ikke til kjernekomponentene. Andre/kunder kan legge sin applikasjon oppå kjernen. Det er ikke åpen kildekode i denne kjernen, men et åpent utviklingsgrensesnitt mot den.

**Pilotprosjektene** bidrar i liten grad til andres utviklingsprosjekt, da de stort sett utvikler piloter og ikke stabile løsninger.

Pilotprosjektene gir i liten grad ut kode de utvikler som åpen kildekode. Dette skyldes for en stor del at det ikke er ressurser i slike prosjekt til å gjøre koden så stabil at den kan gis ut. Det er gjerne heller ikke ressurser tilgjengelig for å følge opp vedlikehold og videreutvikling av koden. Men vi har likevel noen eksempler på at dette gjøres:

- Ett av prosjektene la programvaren ut som åpen kildekode under en BSD-lisens. Årsaken til at denne lisenstypen ble valgt, var at man håpet at noen ville videreutvikle koden til et kommersielt produkt. Hvis målet er at koden skal brukes mest mulig er dette en grei lisens. Man antok at bruk av GPL ville skremt mulige aktører fra å videreutvikle koden og integrere den med egne løsninger. Hvis man hadde valgt å holde koden lukket, ville man måtte satt den ut på anbud for å få andre til å videreutvikle den. Men da kunne man risikert at leverandører som hadde en konkurrerende løsning ville kjøpt opp programvaren for å kvitte seg med konkurransen.
- Også et annet prosjekt har planer om å legge ut egenutviklet kode under en BSD-lisens fordi det skal integreres nært med et kommersielt produkt. Men dette setter begrensninger med hensyn til hvilke åpen kildekode-program de kan bygge videre på (det ekskluderer bruk av kode lisensiert under GPL).
- Et tredje prosjekt legger ut under GPL noen matematiske bibliotek de utvikler.

NST tilbyr en nasjonal læringsportal for helse-Norge kalt **Helsekompetanse.no**. Tjenesten er bygd på åpen kildekode-programvare fra Drupal<sup>41</sup> og ATutor<sup>42</sup>. Produktet er kursene, ikke programvaren. Brukerne forholder seg ikke til programvaren på Helsekompetanse.no, men til tjenestene som tilbys. NST har skrevet noe kode til Helsekompetanse.no selv, men først og

---

<sup>41</sup> <http://drupal.org/>

<sup>42</sup> <http://www.atutor.ca/>

fremst beskrevet ønsket funksjonalitet overfor miljøet som utvikler programvaren de bruker. I senere versjoner har endringene vært inkludert. NST har krevd at endringene skal gis ut under GPL, selv om de har vært i form av tilleggsmoduler. Man valgte en åpen kildekode-løsning for Helsekompetanse.no fordi dette medførte at det var lettere å få support, få inn rettinger, og gjøre rettinger selv. Når man opplever noe rart eller en feil, kan man gå inn i koden og undersøke hva som er feil og lage en feilretting som sendes inn.

NST forholder seg til leverandørene av løsningene som om de var kommersielle, bortsett fra at NST ikke betaler for lisenser. Programvaren er gratis, men man betaler for spesifikke endringer og oppgraderinger og for utvikling av ny funksjonalitet. Feilrettinger betaler man ikke for. NST kjøper oppgradering og endring fra leverandør. ATutor kommer med ny versjon hvert halvår, men gir ut patcher/rettinger etter behov.

NST hadde problem med den forrige programvaren de benyttet og søkte på nettet etter andre som kunne levere det de ønsket. Det finnes sider som lister hva som finnes av åpen kildekode innen ulike kategorier.

Hvis man ønsker å legge ut kode fra pilotprosjekt som åpen kildekode, men ønsker å få tilbake andres endringer for å kunne bruke de selv senere, burde man legge koden ut under GPL.

Selv om man benytter seg av kode som er BSD-lisensiert, er det fornuftig av de som gjør endringer å legge tilbake oppgradert kode for å hindre for mye videreutvikling av koden i ulike retninger ("forking"). Da har man mulighet for å få nye versjoner der egne tillegg allerede er tatt inn.

## Vedlegg C: Opplisting av noen åpen kildekode-prosjekt innen helsesektoren

Dette vedlegget lister opp noen av de mange programvaresystemene for helsesektoren som er lisensiert som åpen kildekode. Noen av disse er kort omtalt i avsnitt 4.1.1 i rapporten. Vi har ikke prøvd ut noen av disse systemene, og vet heller ikke så mye om utbredelsen av dem. *(Alle linkene er sjekket 7.-8. oktober 2008.)*

Flere av initiativene dreier seg mer om arkitektur og rammeverk for EPJ og kommunikasjon, ikke ferdige system. Det er tilfelle for GEHR, OpenEHR, OpenHRE og OpenEMed:

- **GEHR**-prosjektet (<http://www.chime.ucl.ac.uk/work-areas/ehrs/GEHR/index.htm>) i Europa. Dette arbeidet, sammen med det australske GEHR-prosjektet, videreføres internasjonalt (i CEN og ISO) gjennom **OpenEHR** (<http://www.openehr.org/>).
- **OpenEMed** (<http://openmed.sourceforge.net/>) er en plattform for utvikling av helsesystem, kanskje spesielt for sykehus og større virksomheter. Programvaren er BSD-lisensiert. *Web-sidene har ikke vært oppdatert siden 2005.*
- **OpenHRE** (<http://www.openhre.org/>) er et amerikansk initiativ med programvare for utveksling av informasjon mellom pasientjournaler. *(Også dette ser ut til å være et passivt prosjekt, med liten eller ingen aktivitet siden tidlig i 2006.)*

De fleste prosjektene har som mål å utvikle en ferdig applikasjon for helsesektoren, svært ofte et elektronisk pasientjournalssystem (EPJ) eller et mer omfattende system som EPJ inngår i (CMS – Clinical Management System).

Noen system er i første rekke rettet mot sykehus og større virksomheter. Det gjelder bl.a. for:

- **VistA** (Veteran Health Information Systems and Technology Architecture, <http://www.hardhats.org/>) ble utviklet av U.S. Department of Veterans Affairs for bruk i deres sykehus, poliklinikker og sykehjem. En egen organisasjon, WorldVistA (<http://worldvista.org/>), etablert i 2002, for å støtte en større utbredelse av programmet. De lanserte WorldVistA som et EPJ-system lisensiert under GPL i 2007.

De fleste prosjektene retter seg mot primærhelsetjenesten og poliklinikker:

- **FreeMED** (<http://www.freemed.org/>), lisensiert under LGPL, gratis nedlasting, men kommersiell support er tilgjengelig.
- **OpenEMR** (<http://www.oemr.org/>) er et av de mest populære EPJ-system basert på åpen kildekode med ca 35.000 nedlastinger fra SourceForge. Lisensiert under GPL.
- **MirrorMed** (<http://www.mirrormed.org/>), system lisensiert under GPL og bygd på programvaren fra FreeMED og OpenEMR.
- **OSCAR** (Open Source Clinical Applications & Resources, <http://www.oscarcanada.org/>) er et kanadisk system utviklet for primærhelsetjenesten. De regner seg som et fullt CMS-system, og de har også en "personlig journal" tilknyttet sitt system: MyOSCAR (<http://myoscar.org/>).
- **PatientOS** (<http://www.patientos.org/>). Telemed-master-studentene her har brukt dette, mener det er lett å videreutvikle på/fra.

- **GNUmed** (<http://wiki.gnumed.de/bin/view/Gnumed>), internasjonalt prosjekt som primærleger og utviklere har tatt initiativ til. Lisensiert under GPL. Prøves ut blant primærleger og fysioterapeuter.
- **tkFP** (tk Family Practice, <http://tkfp.sourceforge.net/>). *Web-sidene har ikke vært oppdatert siden 2005.*
- **TORCH** (<http://sourceforge.net/projects/op-torch/>). *Web-sidene har ikke vært oppdatert siden 2004.*

Flere system retter seg spesielt mot pasienten og pasientens tilgang til og kommunikasjon med helsesektoren. Flere av disse regner seg som såkalte PHR-system (Personal Health Record):

- **Indivo** (<http://www.indivohealth.org/>) har sitt utspring fra Children's Hospital i Boston. Lisensiert under GPL . De tilbyr en løsning for en personlig kontrollert journal som kan ta inn pasientinformasjon fra mange kilder, tilsvarende som det Google<sup>43</sup> og Microsoft<sup>44</sup> tenker seg i sine løsninger.
- **Tolven** (<http://www.tolven.org/>) er først og fremst et system for "personlig journal" der pasienten kan bestemme hvilken informasjon som skal legges inn og hvem som skal få tilgang. De har også delsystem for lagring av pasientinformasjon og for tilgang til informasjonen for helsearbeidere omkring pasienten. Lisensiert under GPL.
- **MyOSCAR** (<http://myoscar.org/>) er en "personlig journal" for pasienter, i tilknytning til journalsystemet OSCAR.
- **PasientLink** (<http://www.telemed.no/pasientlink>) tilbyr sikker kommunikasjon mellom pasient og primærlege. BSD-lisensiert programvare utviklet av NST. Er senere tatt inn i et kommersielt produkt, men den åpne kildekode er fortsatt tilgjengelig og er brukt i en rekke mindre (student-)prosjekt.

Det finnes et stort antall system for radiologi og annet billedannende utstyr. Her tar vi bare med de som allerede er nevnt i rapporten:

- **OsiriX** (<http://www.osirix-viewer.com/>). Bildeframvisningsprogram for Mac OS X, basert på DICOM-standard. Lisensiert under GPL . Programmet kan hente fram alle typer medisinske bilder fra bildearkiv, f.eks. sykehusets sentrale PACS-arkiv, og vise dem i snitt, flerdimensjonalt og fra ulike vinkler.
- **Dcm4Che** (<http://www.dcm4che.org/>). DICOM-basert programvare for håndtering (arkivering og uthenting) av medisinske bilder. Trippel-lisensiert under MPL, GPL v.2 og LGPL.

Lister over prosjekt og system med åpen kildekode finnes på flere steder, bl.a. disse:

- **Wikipedia** ([http://en.wikipedia.org/wiki/List\\_of\\_open\\_source\\_healthcare\\_software](http://en.wikipedia.org/wiki/List_of_open_source_healthcare_software)) har en imponerende liste over programvare for helsesektoren med åpen kildekode. Listen er delt inn i ulike kategorier som bl.a. sykdomsovervåkning, elektroniske pasientjournaler, system for billedannende utstyr (PACS o.l.) og medisinske informasjonssystem.
- **GPL medicine** (<http://ehr.gplmedicine.org/>) har en liste over noen EPJ-system basert på åpen kildekode, samt review av disse gjort av AMIA (American Medical Informatics

<sup>43</sup> <http://www.google.com/intl/en-US/health/about/>

<sup>44</sup> <http://www.healthvault.com/>

Association) i 2008. De grupperer systemene henholdsvis i system for primærtjenesten, system for sykehus og PHR-system (personal health record: personlig pasientjournal).

- **tkFP** sitt nettsted (fra 2005) har en lang liste med linker til andre elektroniske journalsystem: <http://tkfp.sourceforge.net/links.html>
- **LinuxMedNews** (<http://linuxmednews.com/1087177825/addPostingForm>) har en omtale (fra 2004) av noen EHR-system som er åpen kildekode.
- **Red Hat Healthcare and Life Sciences** (<http://www.redhat.com/solutions/healthcare/>) lister opp noen åpen kildekode-løsninger for helsesektoren som de samarbeider med.