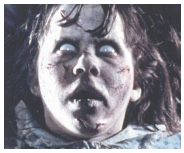
XRadar

Open source-verktøy for kode- og kvalitetsanalyse

Kjetil Jørgensen-Dahl, NOS Clearing ASA
Rodin Lie, NOS Clearing ASA

Kristoffer Kvam, Telenor asa

Teknisk gjeld



Although immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, [...] Shipping first time code is like going into debt.
A little debt speeds development so long as it is paid back promptly with a rewrite. [...] The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.

Ward Cunningham, OOPSLA, March 26, 1992

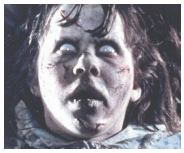
Hvordan ser teknisk gjeld ut?

- Duplisering
- Dårlig navngiving
- Testproblemer
- Høy kopling
- Lav kohesjon / problemer med ortogonalitet / ett ansvar
- Unødvendig kompleksitet
- Brudd på konvensjoner (idiomer) og standarder
- Endringer må reflekteres mange steder
- Store/lange metoder, klasser, API'er
- ...

Refrigerator code / toilet code ☺

- ***Refrigerator Code***:
  - *It's code that you're so proud of that you want to take it home and hang it on the refrigerator, right alongside of your children's drawings.*
- ***Toilet Code***:
  - *It's code that's so mediocre that when somebody encounters it, they just want to flush it down the toilet.*

Hvordan motarbeide teknisk gjeld? (I)



- Google's approach(?)
  - Starte fra scratch
  - Superutviklere
  - En smidig prosess med stor frihet
  - Strenge krav til koden
  - Godt med review/parutvikling
  - Lite tidspress (leve av noe annet?)
  - Masse belønning for *a job well done*
- Svært sjelden mulig...
  - Har som regel en arv å ta med seg
  - Superutviklere er det underskudd på (=koster mye)
  - Kunden, brukeren, forretningssiden presser på
  - Microsoft Vista?

Hvordan motarbeide teknisk gjeld? (II)

- Mer realistisk – mange pågående tiltak:
  - Kompetansebygging
    - Idioms, Smells, Patterns, ...
  - Kulturbygging
    - Positiv feedback, - *Refactor mercilessly!*, - *No broken windows!*, ...
  - Metodikk/teknikk
    - Test-Driven Development, Pair Programming, prioritering, ...
  - Verktøy
    - IDEs, Refactoring, Inspection, Reporting, ...
  - Rammevilkår
    - Forankring hos ledelsen, Tid/ressurser til å investere, ...
  - ...
- Tenk entropi – *man må hele tiden tilføre energi for at systemet ikke skal bevege seg mot kaos*

Inspeksjons- og metrikkverktøy (ett tiltak)

RADAR

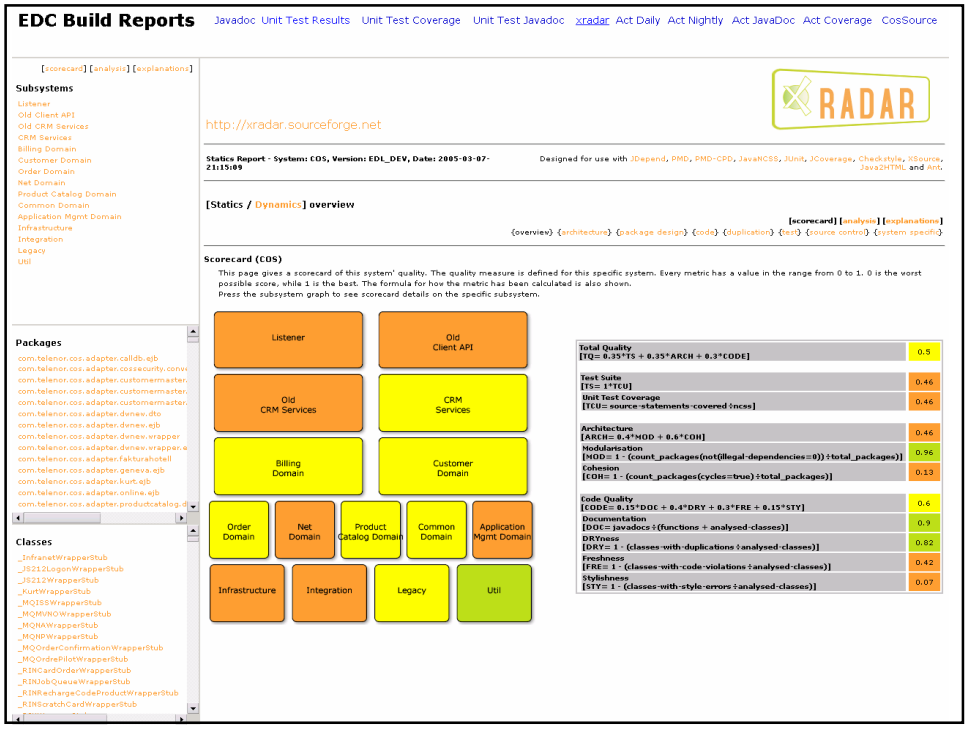
- Finnes det informasjon man kan trekke ut av koden som vil fortelle noe om systemets kvalitet?
- Hypotesen er at det finnes en del indikatorer:
  - Testdekning (andel av koden dekket av tester)
  - Dokumentasjonskompletthet (Javadoc-mangler)
  - Avvik fra kodestandard og standard idiomer
  - Kompleksitetsmetrikker
  - Metrikker for måling av kopling
  - ...

Dersom hypotesen holder ...

RADAR

... og man har et slikt verktøy:

- Kan påvise områder med potensiale for forbedring
- Benchmarking
  - mot andre prosjekt
  - mellom subsystemer/moduler
  - mot seg selv – over tid
- En rekke situasjoner
  - i egne prosjekter
  - i vurdering av open source
  - ved ”audit” og ”due dilligence”
  - når man overtar eller går inn i kode fra andre kilder
  - når man setter bort utviklingen til noen andre



Opprinnelig et beslutnings- og oppfølgingsverktøy for Pareto-prosjektet

RADAR

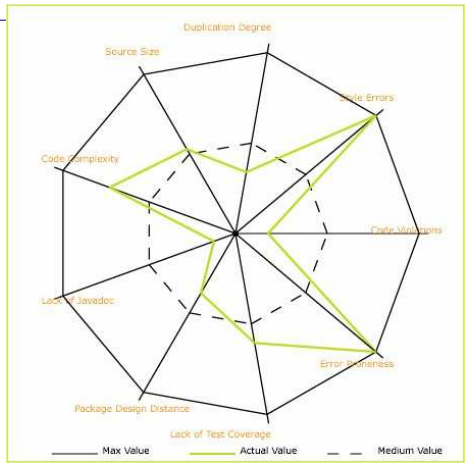
- Strukturering av arkitekturen
- Identifisere problematisk kode (80/20)
- Kontrollere utvikling under og etter prosjektet
- Validere suksess!
- Etablere et skreddersydd perspektiv for viktige roller i systemets forvaltning

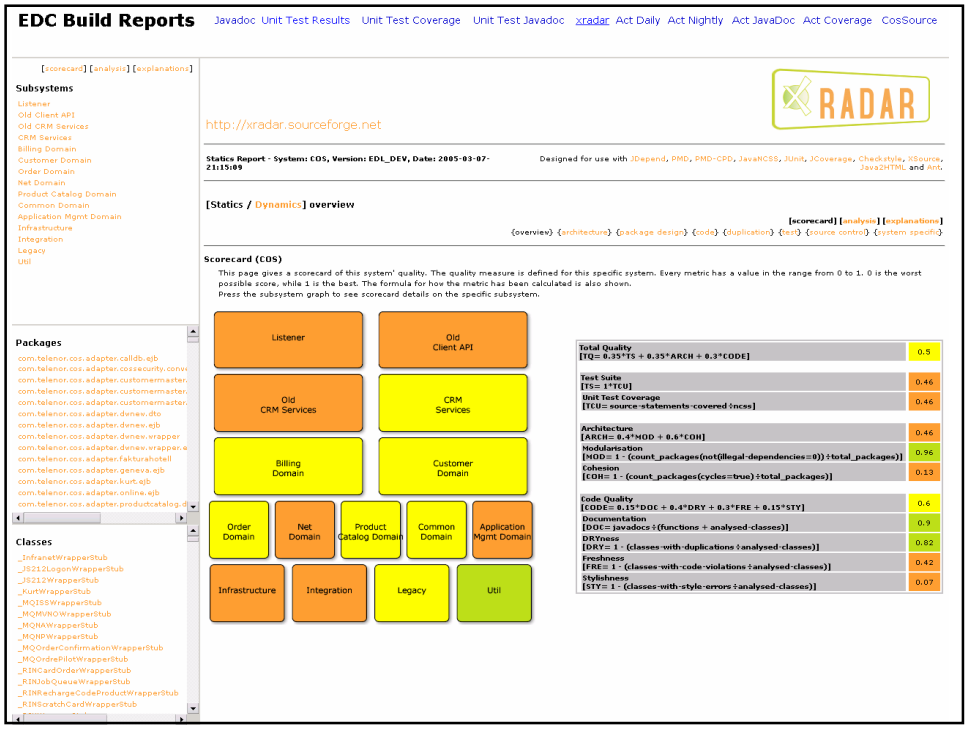
10

NOS

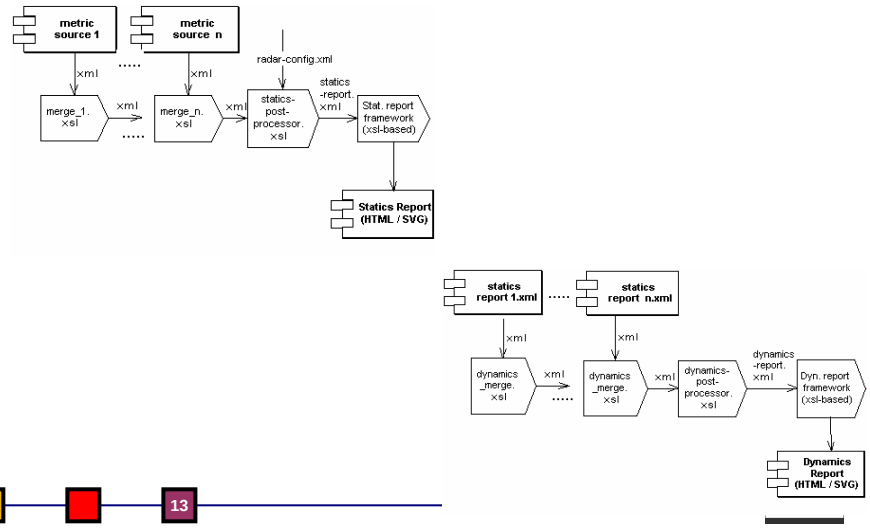
Bruk av XRadar

- Gir utviklerne en standardisert QA på den koden de implementerer
- Oversikt over kodebasen fra system via subsystem, pakke, klasse og metodenivå - og helt ned til enkeltlinjer.
- Gir kontroll over det som leveres inn i systemet
- Leder mot smartere strategiske investeringer
- Definerer systemets interne kvalitet





XRadar Architecture



13

Dagens kilder og verktøy

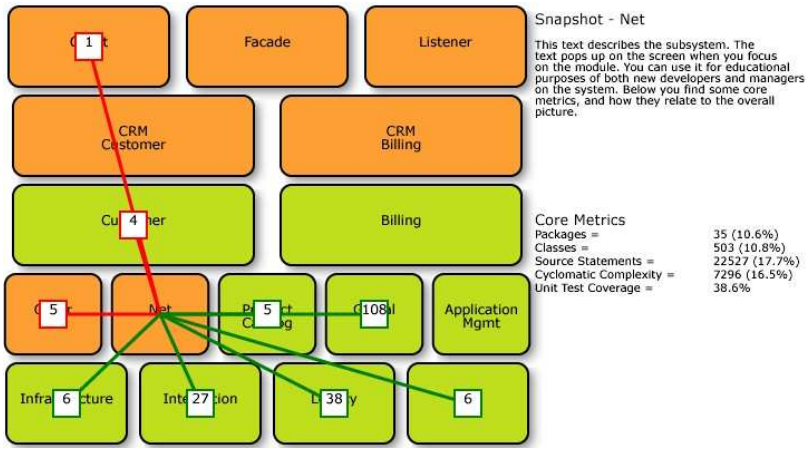
- Koden (kilde og kompilert), via…
  - PMD
  - PMD-CPD
  - Checkstyle
  - JDepend
  - JavaNCSS
  - JUnit
  - JCoverage/Cobertura
  - …
- Konfigurasjon
  - Subsystemdefinisjon
  - Releaser
- Konfigurasjonsstyrings-verktøy/versjonsstyrings-system
  - Aktivitet (feilretting)

14

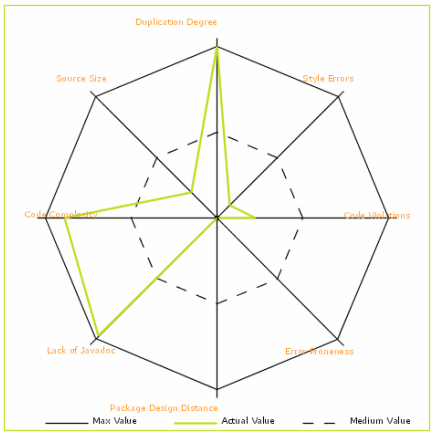
Eksempler på praktisk bruk

- Kontroll av ulovlige avhengigheter
- Spidergraf på pakke-nivå
- "Svartelister"
- Avvik fra kodestandard
- Lokalisere kodeproblemer (anti-patterns)

”Spaghetti”-motvirkning – på subsystemnivå



Spidergraf på pakke-nivå



- Tilgjengelig fra web og plug-in

"Svartelister"

Method Analysis - XRadar Test

[overview] [class analysis] [complex methods analysis] [changed methods analysis]

In the analysis below, the methods are ranked by number of changes to method multiplied by the method's cyclomatic complexity. In many systems, this number will be an indicator on methods that should be refactored. The reason for this is that methods with high complexity will have a tendency to be hard to maintain.

Rank	Changes	Source Statements	Cyclomatic Complexity	Package	Class	Method
1	2	32	12	org.xradar.test.f	F1	F1()
2	1	32	12	org.xradar.test.a	A1	testMethod29()
3	1	32	12	org.xradar.test.c	C1	testMethod28()
4	1	32	12	org.xradar.test.c	C1	testMethod29()
5	1	25	6	org.xradar.test.d	D1	methodPartyCopiedToD2()

"Svartelister" II

Methods Changed Since Last Version Analysis - XRadar Test

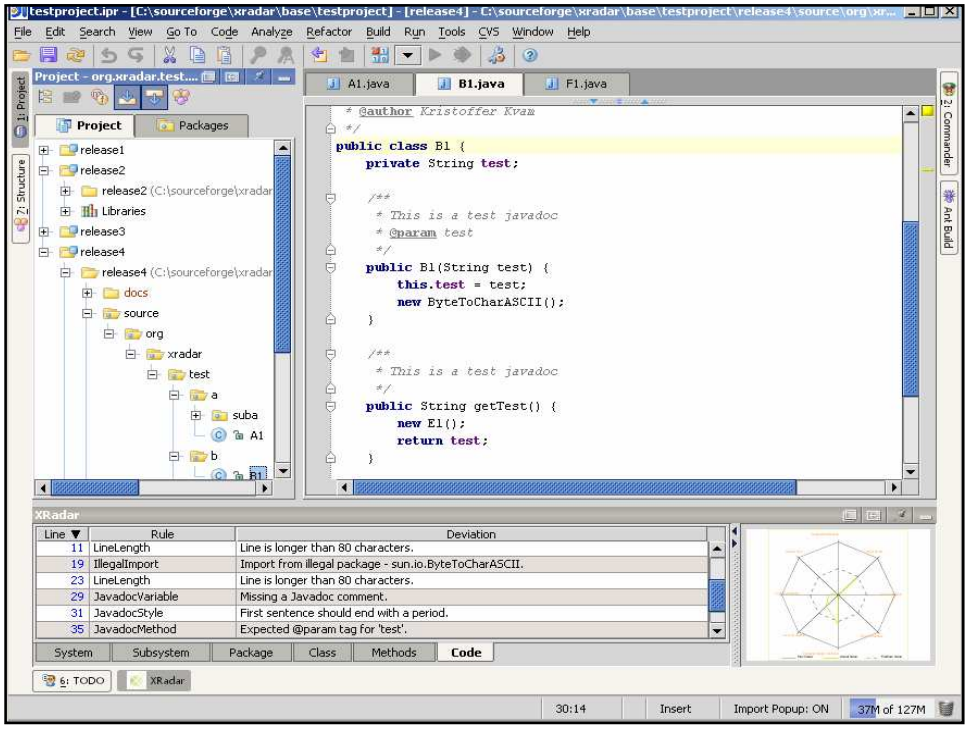
[overview] [class analysis] [complex methods analysis] [changed methods analysis]

In the analysis below, the methods are ranked the lack of test coverage of the methods multiplied by their complexity. This analysis shows the methods that have been changed since the last build. Use this report to identify complex methods that are that are not being tested when added or changed.

Rank	Changes	Source Statements	Cyclomatic Complexity	Package	Class	Method
1	1	32	12	org.xradar.test.a	A1	testMethod29()
2	1	9	3	org.xradar.test.a	A1	methodWithParameters(boolean,boolean)
3	2	3	1	org.xradar.test.b	B1	getTest()
4	1	1	1	org.xradar.test.c	C1	testMethod1()
5	1	1	1	org.xradar.test.c	C1	testMethod2()
6	1	1	1	org.xradar.test.c	C1	testMethod3()

Avvik fra kodestandard

- Mangler og feil
- Kilder: pmd, checkstyle, findbugs(?)
- Enkelt med plug-in



Lokalisere andre kodeproblemer - *"Smells"/Anti-patterns*

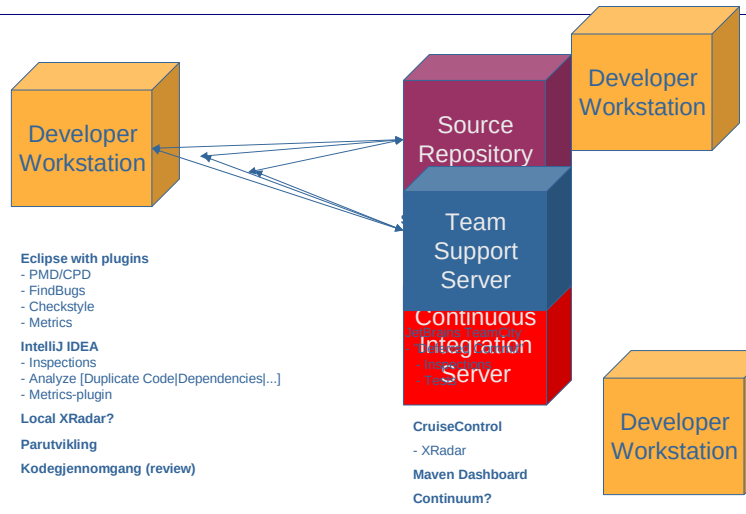
RADAR

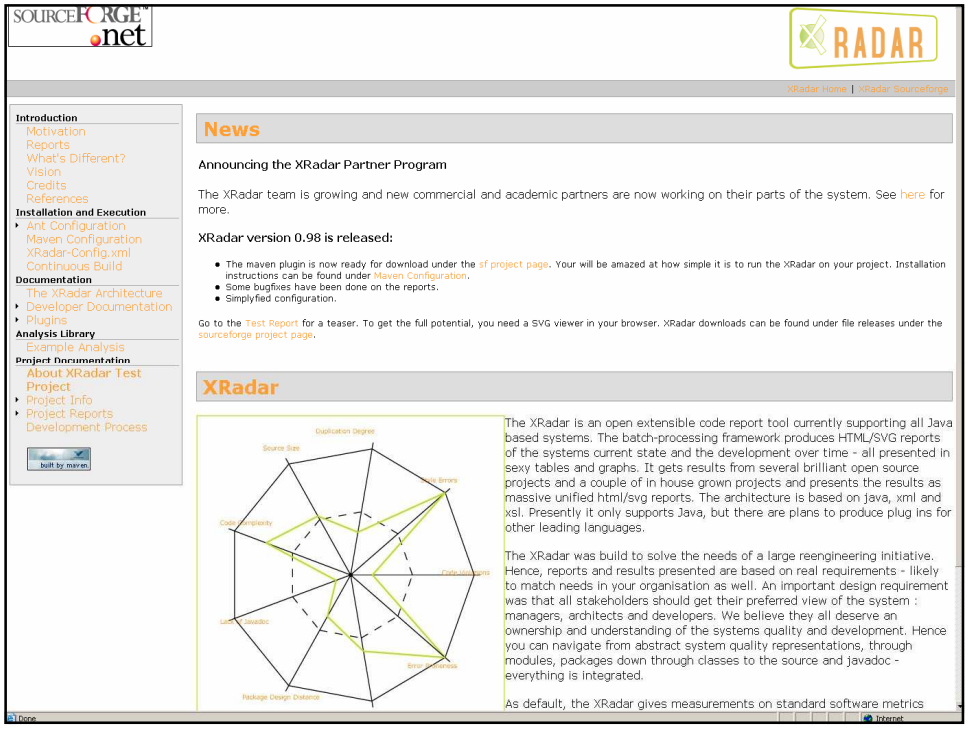
- Duplisering (copy&paste)
- Store klasser (blobs)
- Spaghetti på klassenivå
  - Høy kompleksitet
  - Sykliske avhengigheter
  - etc
- Redundant kode ("lava flow")

Noen utfordringer?

- Tilbakemeldingstid
  - Tar tid på store prosjekt som COS (1 time) (og minutter på små prosjekt)
- ”Information overflow”
  - Trenger mer skreddersydde ”views”
- Effektiv presentasjon i plugins
  - Mye info – liten plass
- Regime for innsjekket kode
  - Bør kjøres *før* innsjekking
- Har mange ideer … men kunne gjerne hatt enda mer ressurser…

Hvordan komme i gang?





RADAR

XRadar – oppsummering

- Rapport- og analyseverktøy for Java-løsninger
- Styringsverktøy
  - Utviklere, arkitekter og IT-ledere
  - Målbildeoppnåelse og systemkvalitet
- Detekterer automatisk problemområder og trender i systemer
- Internasjonal oppmerksomhet, open source (på Sourceforge siden 2004) under BSD-lisens

XRadar

Presentasjoner

- XP2004
- javaBin-møte (juni 2004)
- For Simula og Norsk Regnesentral (2005)
- En rekke konsulentselskap (2004,2005)
- OOPSLA (2005)
- JavaZone (2005)
- Fag på IFI… (2005/2007)
- +++

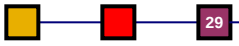
Bra interesse

- Flere forskningsprogram er relatert til XRadar
- Over hundre nedlastninger hver måned

Spørsmål?

Kristoffer Kvam (kristoffer.kvam@telenor.com)
Kjetil Jørgensen-Dahl (kjd@nos.no)

http://xradar.sourceforge.net/



NOS