software pilots
TRIFORK.

Write Less Code!
Trends in Programming Style

Kresten Krab Thorup, Trifork
krab@trifork.com

Today's Thesis

A. A large code base is hard to understand.

B. You are likely to introduce bugs in a hard-to-understand piece of code.

implication?

Large code bases are likely buggy.

Which Solution?

- Write Less Code?
- Write Code that can be Understood?

[hint: I tricked you to come today]

Defects per KLOC

- Industry Average: "about 15 - 50 errors per KLOC."
- Microsoft Applications: "about 10 - 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per KLOC in released product (Moore 1992)."
- "Harlan Mills pioneered 'cleanroom development', a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product" (Cobb and Mills 1990).

Check this

```
#!perl -pl
s!.!y$IVCXL426(-:$XLMCDIVX$dfor$$_.=5x$&*8%29628;$$$_=$_!egfor-4e3..y/iul-}/-$+ /%s''$';*_=eval
```

```
#!perl -pl
s!.!y$IVCXL426(-:$XLMCDIVX
$dfor$$_.=5x$&*8%29628;$$$_=
$_!egfor-4e3..y/iul-}/-$+ /
%s''$';*_=eval
```

perlmonks.org

And this...

```
(0/:l)(_+_)
```

http://www.google.com/search?q="(0/:l)(_%2B_)"

Which really means...

```
(0/:list)(_+_)
```


Which really means...

```
list./:(0)(_+_)
```

Which really means...

```
list.foldLeft(0)(_+_)
```


Which really means...

```
list.foldLeft(0)(x,y => x+y)
```


Which really means...

still convinced?

Writing fewer lines of code is not in itself a means to improve quality

... understandable code is

software pilots
TRIFORK.

REBOOT

software pilots
TRIFORK.

Write Code that can be Understood!
Trends in Programming Style

Kresten Krab Thorup, Trifork

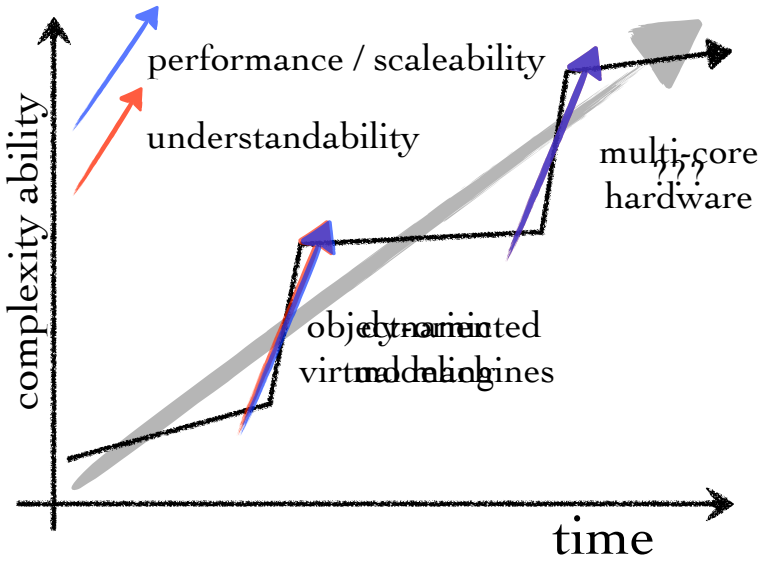
I'm no expert

I'm on a out to figure out how to

improve software quality

by means of

improving software understandability



Trends on Understandability

- Software Craftsmanship
- Clean Code [Bob Martin], Implementation Patterns [Kent Beck]
- Shared Code Ownership / Code Review
- Automated Build & Test
- Domain Specific Languages
- Domain Driven Design - DDD / Modeling

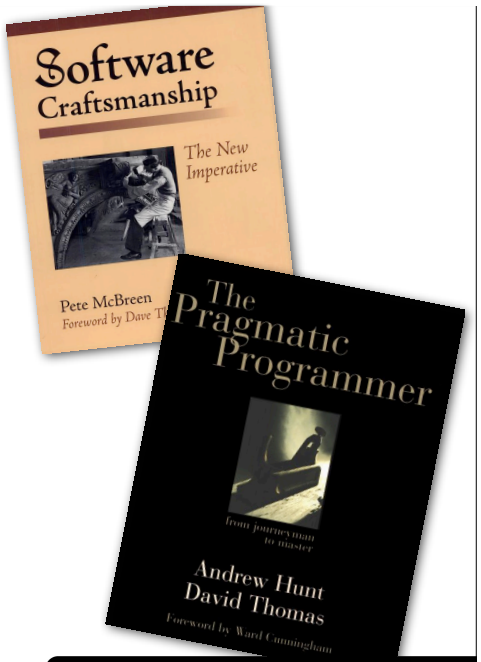
Meeting Richard Stallman

Software Craftmanship

- Being a "Software Professional"

  Apprentice ⇒
  Journeyman ⇒
  Master

- Taking responsibility, Learning-by-doing, long time customer relationships, ...



Implementation Patterns / Clean Code

- Simple, statement-level little rules for how to structure code.
- Gives you some vocabulary to talk about code quality at the detail level.

"Implementation Patterns"

If's are evil... ALLOW THEM WHEN

Condition can be expressed without **AND**, **OR**, operators

```
if (myContainer.hasItems()) {
   // ...
}
```


"Implementation Patterns"

If's are evil... ALLOW THEM WHEN

Condition is about a local field

```
if (this.isVisible) {
   // ...
}
```


"Implementation Patterns"

If's are evil... ALLOW THEM WHEN

Condition is is a non-float comparison, i.e.

```
if (myCertificates.count() < 3) {
   // ...
}
```

"Implementation Patterns"

If's are evil... BE CAUTIOUS

When there is more than one AND/OR operators, i.e.

```
if (isVisible && !isParent) {
   // ...
}
```


"Implementation Patterns"

If's are evil... BE CAUTIOUS

When there is negative logic

```
if (!isSenior) {
   // ...
}
```

positive statements are easier to understand

"Implementatin Patterns"

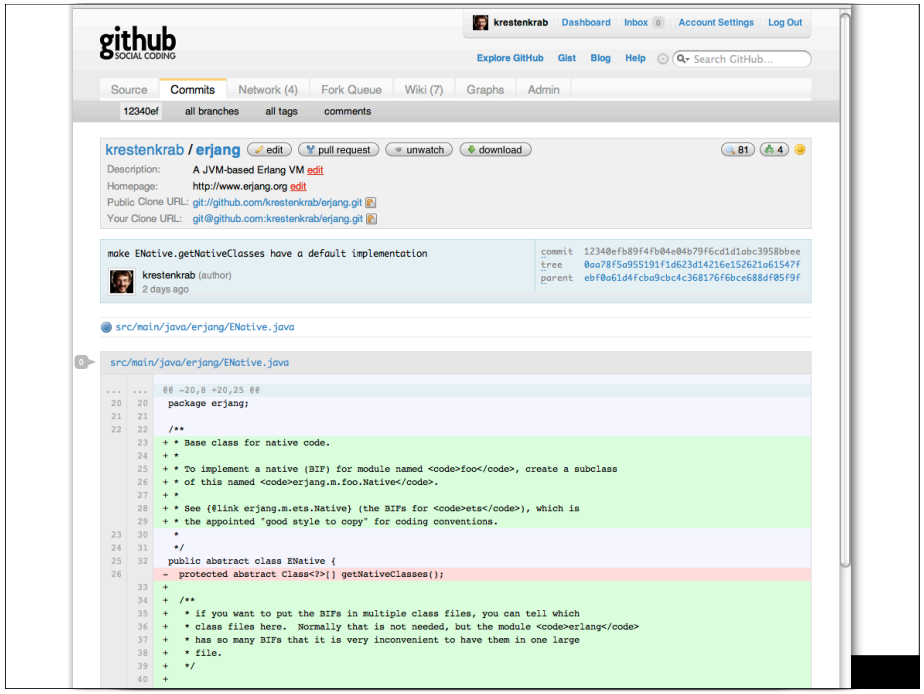
Good to read, because they give you an awareness of "what good code is".

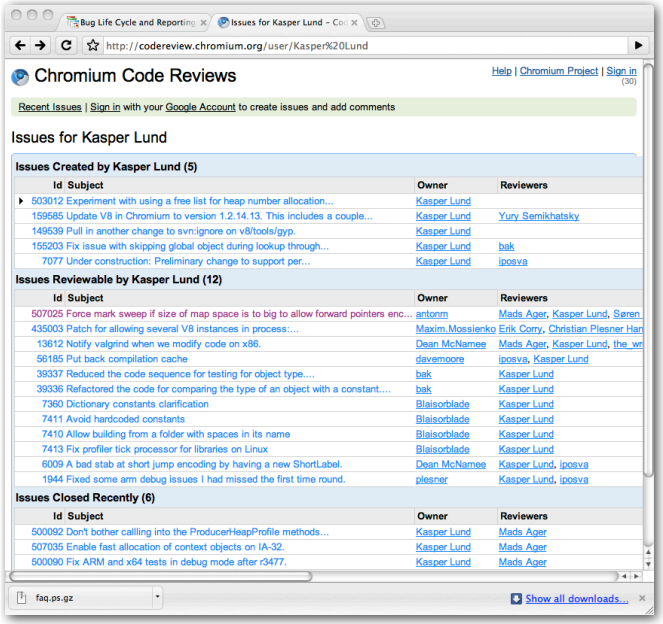
More than a "coding convention"

It's an invitation to have your own opinion!

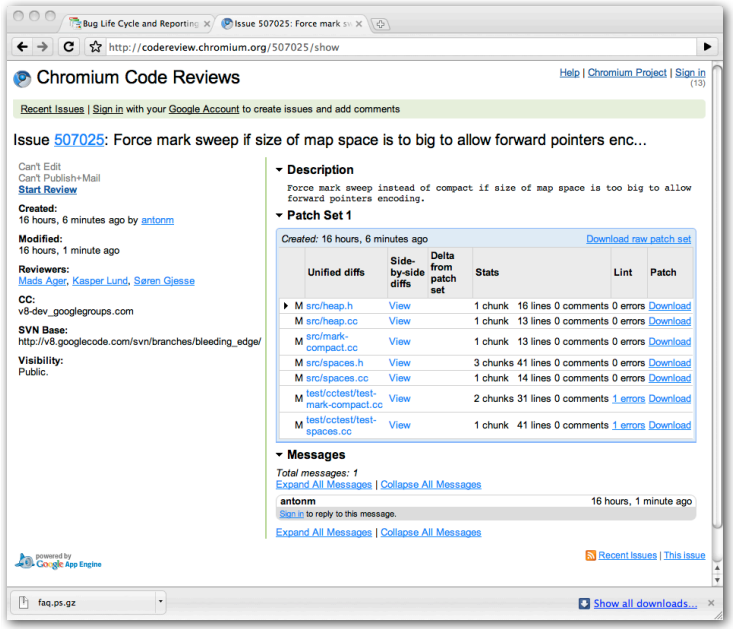
Sharing Code

- We have a lot to learn from open source projects> Distributed Collaboration
  - In the late 80's the GNU project developed the style of cooperation that is main stream today.
- XP/practices for "pairing" when writing production code.

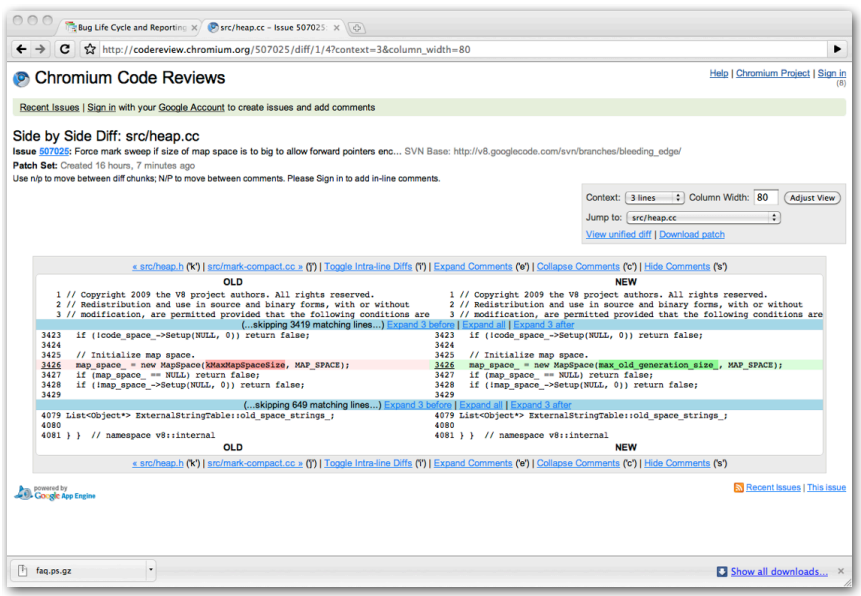




TRIFORK.



TRIFORK.



TRIFORK.

Using Version Control

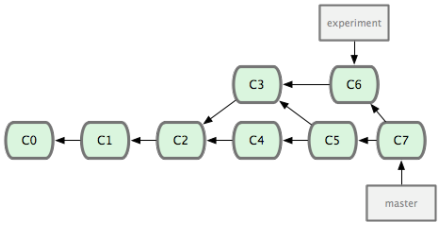
- First we used file-oriented version control
- Then repository-oriented CVS, SVN, ...
- The new kids in town:
  GIT, Mercurial & Darcs

GIT [github] **& Mercurial** [google code]

No central repository

Edit history is a DAG of deltas

Each revision identified by strong hash of all edits included.

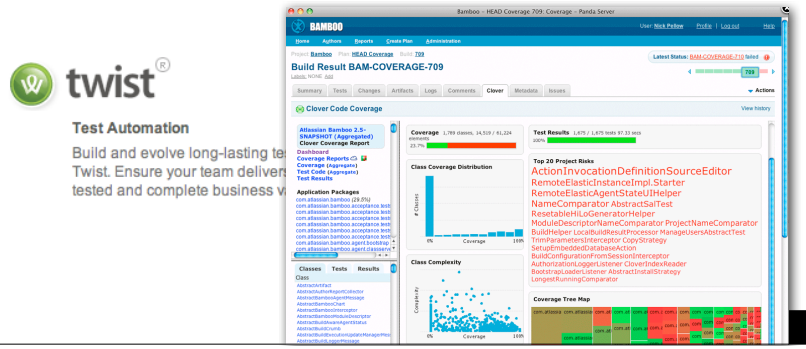


Automated Build

- Atlassian, Australian maker of developer tools, measures project performance by timing
  - **Compile-run cycle [short]**
  - **Checkout-build-deploy [from scratch]**
- Imagine what this does to make a codebase accessible to a new developer joining the project.

Continuous Integration

- Upon every "commit" to the source code repository, run automated build and test.

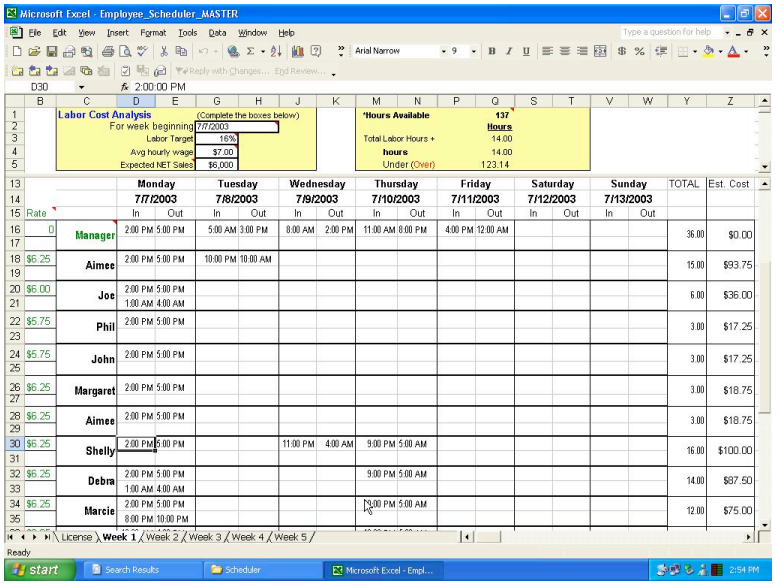


Continuous Delivery

- Deploy to production upon Commit
- The Developer's responsibility
- Bugs are much cheaper to fix right away
  - Suitable for online systems;
  - Need infrastructure to automatically track issues
  - Jez Humble's new book.

Domain Err

- The worst kind of "bug" is if your program does not do what the customer wants!
- Ideal: Make your program, such that the customer can read it; or even better: so he can write it.
  - End-user programming
  - Domain Specific Languages
  - Domain Driven Design
  - Short development cycles ⇒ feedback



resolver systems ltd

End-User Participation

- Domain Specific Languages
  - User's typically write business-logic or test cases in specialized languages
- Intentional "Domain Workbench"
  - Excel on steroids, still being proven

Technical "Domain Specific Languages"

- **Rails** is a domain specific language for the "domain" of creating database backed web applications.
- **FIT** is a test framework, that allows end-users to write tests/specifications in a tabular format in Excel or Microsoft Word.
- In dynamic languages (Ruby, Smalltalk, Javascript, Groovy, **Ioke**, ...) it is quite easy to create your own.

Expert "Domain Specific Languages"

- **ERP** systems (SAP, Maconomy, Navision) typically embody a language specifically for "application programming".
- **Trifork Athene** uses a DSL for doctors to describe procedures for determining diseases.
- There needs to be a good number of usages, before a DSL makes sense. One use case is seldom enough.

DSLs: create your own

- Internal DSLs
  - Use the language you already use (Java, C#, Ruby) to express things more concisely
- External DSLs
  - Write a parser/graphical editor specifically for your domain.

DDD

- Focus on understanding the customer's domain.
- Design [code] guide lines for how to make this successful.
- OOA&D popularized by an american.