

# Maintaining Large Software

Welcome to the real world

Erwan Lemonnier  
StorProg 2010

# whoami

- Erwan Lemonnier - [erwan@cpan.org](mailto:erwan@cpan.org)
  - 8 years as a free software developer for CPAN
  - 3 years in an IT-security startup,  
100 000 lc, 3-7 developers
  - 5 years at Premiepensionsmyndigheten,  
500 000 lc, 5-20 developers
  - >1 year at Glue Finance AB,  
130 000 lc, 3-6 developers

# Topic

In practice, being a developer usually means maintaining code, together with many other developers.

This requires specific tools and skills

# Today's menu

- Software development in the real world
- The tools of the trade
  - Development process
  - Bug Tracker
  - Version Control
  - Documentation
  - (coding)
  - Code review
  - Testing

# The real-world



# The real-world

- Most software positions are maintenance positions
- Typically:
  - You become responsible for a load of code written by someone else
  - You have to fix bugs, implement new features
  - Without breaking anything
  - While coordinating with other developers

# An example: Pluto



# An example: Pluto

- The core system of the Swedish Premiépension
- Holds one saving account per person
- Performs all fund transactions  
(buy/sell/switch/dividend/merge/split...)
- Implements the mathematical insurance model
- Manages pension payments
- Etc...

# Pluto in numbers (2009)

- 320 000 lines of Perl code (.pl, .pm, .t)
- 68 000 lines of SQL script
- 27 000 lines of shell script
- 26 000 lines of HTML (Mason)
- 230 database tables
- Largest table has 500 000 000 entries
- 750 gigabytes of data in an Oracle database
- Managing 250 000 000 000 SEK (23 billion euros)
- 5.5 million users
- Up to 30 developers over 7 years

# Pluto's challenges

- Huge volumes of data
- No bugs allowed
  - A bug triggered by 1 person in a million will occur 5 times!
- High availability: running 24/7
- Tight development deadlines
- Politicians!

# Pluto's design: programs

- A batch system
- Many (>100) small programs doing simple things
- Perl programs wrapped in shell scripts
- Simple interfaces between them:
  - Inbox model
  - Database tables
  - A daily run schedule

# Pluto's design: database

- Always add, never remove
- The database schema evolves in small steps (migrations)
- Keep a history log of every change
- Trace back every change to a program

# Pluto's surroundings

- APIs towards other systems
- A large set of reporting/data mining tools
- Some administrator interfaces (web based)
- Etc...

# Your mission is...

- So you start at Pensionsmyndigheten
- You are in charge of Pluto, together with 5 others
- Your first task: “The government has voted a new law. Pluto must support a new kind of funds: a generation fund.”
- Go ahead!

# Problems:

- Understand the change specifications
- Understand Pluto
- Translate the specs into actual code changes
- Plan work for the team
- Code/Test
- Meet deadlines
- Don't break anything!

**O NO WTF!**



**Let's give it a try**

# What would you do?



# Strategy:

- Step 1: get the code
- Step 2: run the tests
- Step 3: read the doc
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit (goto step 4 until done)

# Wait wait wait...

We just defined a development process!

*wikipedia:*

“A software development process is a structure imposed on the development of a software product”

*examples:*

waterfall, spiral model, agile development...

# Development process

- There are many possible ones
- It is a crucial part of any project!
- Not always explicit
- It usually has flaws
- It is not always followed...
- A social challenge as much as a technical one

# Example: Pluto

- Pluto's development process:
  - Pilot study
  - Project planning
  - Change and test specifications
  - Coding
  - Testing
  - Live tests
  - Deployment

# Development process

So learn your project's development process and its tools before you start messing around :-)

Or agree on one with your team...

# mmm...

- Step 1: get the code <== YOU ARE HERE
- Step 2: run the tests
- Step 3: read the doc
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit

**Wait!!!**



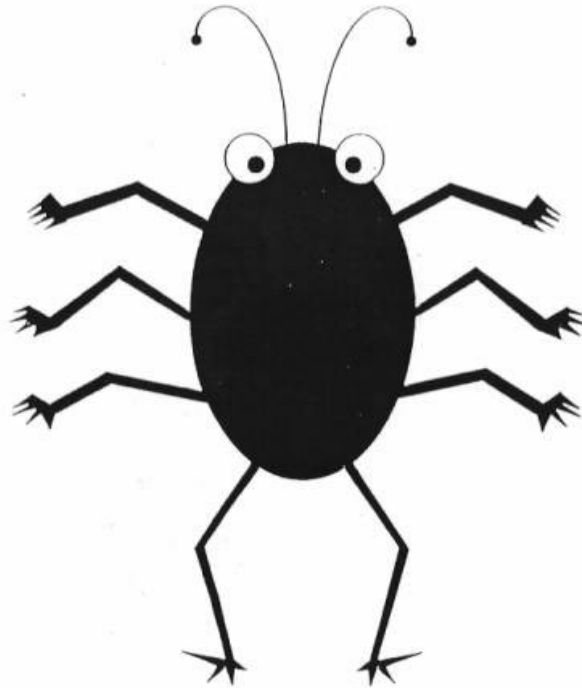
# Wait!!

We are not writing a system from scratch.

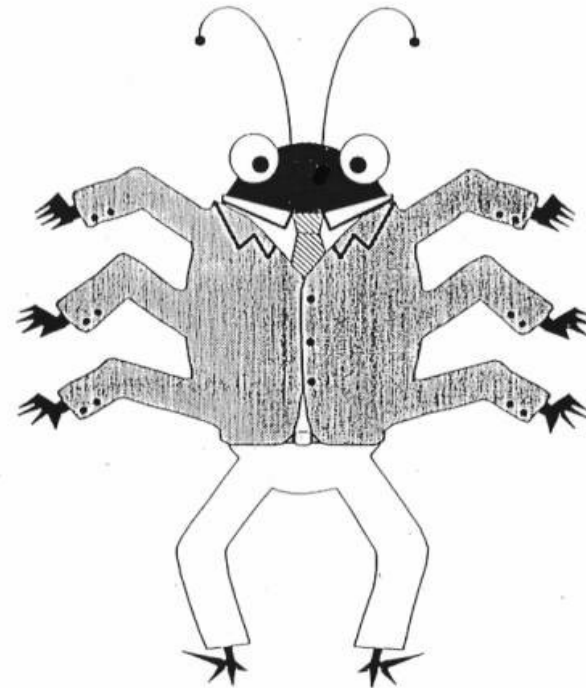
We are modifying an already existing one.

So before even getting the code, we need to keep track of the functionality change at a project level.

# Bug Tracking System

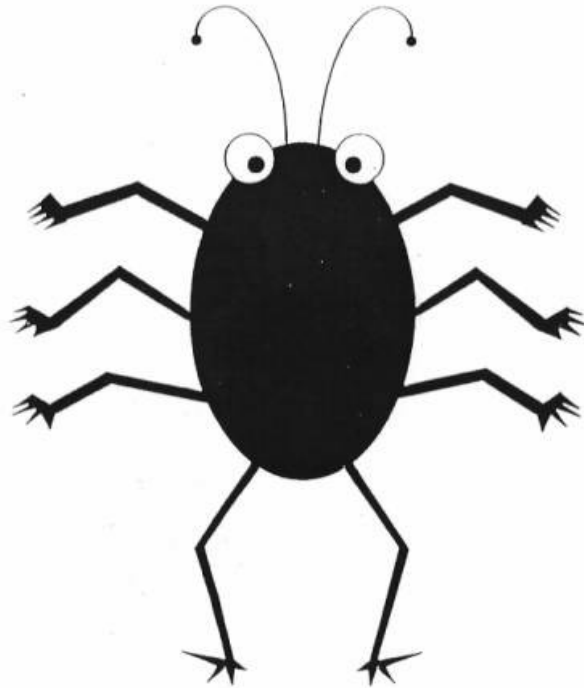


**BUG**

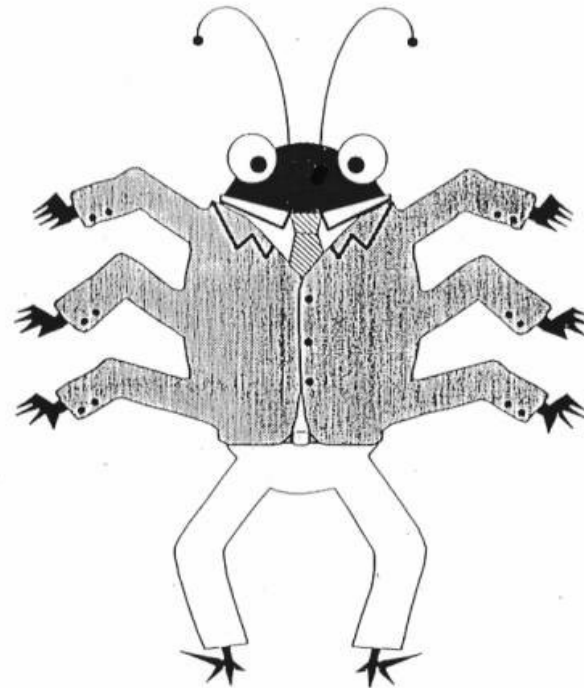


**FEATURE**

# Who has used one?



**BUG**



**FEATURE**

# Bug/Ticket/Feature tracker

- RT, bugzilla, redmine, trac, github...
- Usually have:
  - bug/feature/issue list
  - comments/attachments/links
  - Each ticket moves between states
    - (hence enforcing a development process)

# RT at a glance

- Home
- Distributions
- Search Tickets
- Tools
- Preferences

Home

Please report any issues with rt.cpan.org to [rt-cpan-admin@bestpractical.com](mailto:rt-cpan-admin@bestpractical.com).

### Bookmarked Tickets

Edit

### 10 highest priority tickets I own

Edit

#	Subject	Priority	Queue	Status
36912	<a href="#">Missing test plan error in latest Hook::Filter</a>	0	Hook-Filter	open

### Bugs in My Distributions

Edit

#	Subject Queue	Severity Priority	Last Updated By Owner	Last Updated Created
19785	<a href="#">IO::Handle error in t/lib//MockDB/DBI.pm</a> Class-DBI-AutoIncrement	0	DAVIDRW Nobody	4 years ago 4 years ago
36912	<a href="#">Missing test plan error in latest Hook::Filter</a> Hook-Filter	0	david@davidfavor.com ERWAN	2 years ago 2 years ago
35865	<a href="#">Test::More broke Hook::Filter</a> Hook-Filter	Normal 0	ANDK Nobody	2 years ago 2 years ago
49623	<a href="#">Finance::Math::IRR differ from Excel's XIRR</a> Finance-Math-IRR	0	kcheung@adamstreetpartners.com Nobody	12 months ago 12 months ago
41108	<a href="#">tough to trap errors in taint mode</a> IPC-Open3-Simple	Normal 0	MARKLE Nobody	2 years ago 2 years ago
19788	<a href="#">backwards compatible tests</a> Class-DBI-AutoIncrement	0	DAVIDRW Nobody	4 years ago 4 years ago

### 10 newest unowned tickets

Edit

### Quick ticket creation

Subject:

Queue:  Owner:

Requestors:

Content:

Create

### Quick search

Edit

Queue	new	open	stalled
<a href="#">Class-DBI-AutoIncrement</a>	2	0	0
<a href="#">DBIx-QueryByName</a>	0	0	0
<a href="#">Finance-Math-IRR</a>	1	0	0
<a href="#">Finance-SE-PPM</a>	0	0	0
<a href="#">Hook-Filter</a>	1	1	0
<a href="#">IPC-Open3-Simple</a>	1	0	0
<a href="#">Jifty-DBI-Record-AutoIncremented</a>	0	0	0
<a href="#">later</a>	0	0	0
<a href="#">Log-Localized</a>	0	0	0
<a href="#">Math-Polynom</a>	0	0	0
<a href="#">Python-Decorator</a>	0	0	0
<a href="#">Sub-Contract</a>	0	0	0
<a href="#">XML-IDMEF</a>	0	0	0

### Refresh

Don't refresh this page.

Go!

# Back to Pluto

- In Pluto's case:
- We register a new ticket: “implement law XYZ”
- The ticket will go through states mirroring the development process:
  - Pilot study
  - Project planning
  - Change and test specifications
  - Coding
  - Testing
  - Live tests
  - Deployment

# mmmm...

- Step 1: get the code <== YOU ARE HERE
- Step 2: run the tests
- Step 3: read the doc
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit

Getting the code...

# Meet the VCS

# Survey

Who has used one?

Which one?

# The VCS jungle

- cvs, svn, svk, git, darcs, mercurial, bitkeeper
- Different workflows:
  - central repository vs distributed
  - atomic commits vs not
  - branches or not
  - different merge algorithms
  - etc.

# Example: checking out

```
# cvs
```

```
cvs -d :pserver:bob@dev.bleh.com:/cvs login
```

```
cvs checkout bigbrother
```

```
# svn
```

```
svn co -username=bob https://dev.bleh.com/svn
```

```
# git
```

```
git clone git://perl5.git.perl.org/perl.git
```

# Example: commit

```
# cvs/svn
```

```
(cvs|svn) commit file1 file2 dir1
```

```
# git
```

```
git fetch dev
```

```
git merge master..dev/master
```

```
git add file1 file2 dir1
```

```
git commit
```

```
git push ssh://bob@perl5.git.perl.org/perl.git  
master:master
```

# VCS in practice

- You will use an editor with vcs support (eclipse, emacs, textmate...)
- Your project will use only a subset of the vcs features
- The VCS might be integrated with:
  - bug/ticket tracker
  - continuous integration server

# mmmm...

- Step 1: get the code
- Step 2: run the tests <== YOU ARE HERE
- Step 3: read the doc
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit

# The art of testing



# Survey

Who has run automated tests?

Who has written some?

# Why regression testing?

- Manual testing is almost worthless
  - Time consuming
  - Error prone
  - Depend on knowledge transfer
- Maintaining software without regression tests is like skydiving without a parachute
- Regression testing an art is, Mastering it you MUST!

# Regression testing

- Many frameworks for testing
  - unittests
  - TAP
- Usually language specific
- Many levels of testing
- If you inherit a project without tests, write some before you change anything!

# Example: perl

```
erwan@foo:~/perl-5.8.9$ ./Configure
(...)
erwan@foo:~/perl-5.8.9$ make
(...)
erwan@foo:~/perl-5.8.9$ make test
(...)
t/pod/plainer.....ok
t/pod/pod2usage.....ok
t/pod/pod2usage2.....ok
t/pod/poderrs.....ok
t/pod/podselect.....ok
t/pod/special_seqs.....ok
t/pod/twice.....ok
t/x2p/s2p.....ok
All tests successful.
u=1.33  s=2.49  cu=149.17  cs=42.89  scripts=1106
tests=133451
```

# Semi test-driven development

- Develop code in small steps
- So the code can run at every step
- Run it within a test file
- Add new tests to validate its behavior
- Run the whole test suite to see nothing got broken

=> You know at every step that your code runs and is correct!

# Advanced regression testing

- Continuous Integration:
  - Build/test continuously
  - Or after each commit
  - Fast and targeted feedback!
- Test coverage

# mmm...

- Step 1: get the code
- Step 2: run the tests
- Step 3: read the doc <== YOU ARE HERE
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit

# Documentation

“Real programmers don't comment their code. If it was hard to write, it should be hard to understand and harder to modify.” *anonymous*

# Documentation

- Is there a documentation?
- What kind?
  - Inline in the code?
  - API specs?
  - Software design (high level)?
  - End-user manual?
  - Wiki? Mail archive? Forum?
- Each has a different target
- Each is important!

# Real world documentation

- Documentation is often a rush work at the end of the project
- The documentation is seldom maintained and becomes outdated

*What would you do?*

# Real world documentation

- Talk to the other developers:
  - In the flesh
  - IRC/mailing lists
- Read changelogs (file, vcs log, bug tracker)
- Read the source
- Start a wiki

# Maintaining documentation

- Problems:
  - Keep doc and code synchronized
  - Documentation coverage
- Solutions:
  - Auto-generate some doc
  - Regression tests for doc coverage
  - Make it part of the development process

# mmm...

- Step 1: get the code
- Step 2: run the tests
- Step 3: read the doc
- Step 4: (the difficult part) <== YOU ARE HERE
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit

**Before coding...**

# Respect the project's structure

- Directory structure
- File naming conventions
- README?
- Changelog?
- Commit messages

# Example: a perl module

```
$ cd ~/projects/MyNewModule
$ find *
Changes
lib
lib/My/New/Module.pm
Makefile.PL
MANIFEST
t
t/fork.t
t/pod.t
t/pod-coverage.t
t/query.t
```

# Respect the code standard

- Might not be explicit
- If none, create one

# Talk to the gurus

- Always discuss your changes with the experienced developers
- This will save 80% of your time

# mmm...

- Step 1: get the code
- Step 2: read the doc
- Step 3: run the tests
- Step 4: (the difficult part)
  - Hack some code  $\Leftarrow$  YOU ARE HERE
  - Test it
  - Get it reviewed
  - Commit

**Coding aaahhhh :-)**



# mmm...

- Step 1: get the code
- Step 2: read the doc
- Step 3: run the tests
- Step 4: (the difficult part)
  - Hack some code
  - Test it <== YOU ARE HERE
  - Get it reviewed
  - Commit

# mmm...

- Step 1: get the code
- Step 2: read the doc
- Step 3: run the tests
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed <== YOU ARE HERE
  - Commit

# Code Review



**Do Not Be Afraid.**



# Code Review

- Ask for it
- Ask a guru
- You will learn A LOT!
- Review others code too
- Creates the social fabric of a dev team

# mmmm...

- Step 1: get the code
- Step 2: read the doc
- Step 3: run the tests
- Step 4: (the difficult part)
  - Hack some code
  - Test it
  - Get it reviewed
  - Commit  $\Leftarrow$  YOU ARE HERE

So...

Now, what did we forget?

# Today's menu

- Software development in the real world
- The tools of the trade
  - Development process
  - Bug Tracker
  - Version Control
  - Documentation
  - (coding)
  - Code review
  - Testing

# Questions?



**Thank you!**



**Glue Finance has work for you!**