

Pluto

or

How to make Perl juggle with billions

Erwan Lemonnier / Niclas Lundborg
Premiepensionsmyndigheten
DYPL 2009

Introducing Pluto

- The pension crisis
- A solution: let users choose themselves in which funds to save
- Users take risk but achieve higher profits
- Premiépensionmyndigheten (PPM)

Introducing Pluto

- Pluto is:
- PPM's financial platform
- holds one saving account per person
- performs all fund transactions
(buy/sell/switch/dividend/merge/split...)
- implements the mathematical insurance model
- manages pension payments
- and more!

Pluto's story

- A large IT-consultant firm tried and failed
- Pluto was a backup prototype
- Started small, grew one feature at a time:
 - First money in the system (Summer 2000)
 - First fund selection (Autumn 2000)
 - First payment (Mid 2001)

Challenges

- Huge volumes of data
- No bugs allowed
 - one bug triggered once for every 1 million persons means at least 5 bugs a day! (Probably more)
- High availability: running 24/7
- Tight development deadlines
- Politicians!
- User requirements

Pluto in numbers (2006-10)

- Loads of code:
 - 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 architecture



Pluto's architecture 1/3

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

Pluto's architecture 2/3

- A huge Oracle database
- Always add, never remove
- The database schema evolves in small steps
- Keep a history log of every change
 - Each program running gets a unique ID, to which every change refers
 - Every program logs its actions to the database
 - Every database change is made by a program
 - To update a row, add a new one with a higher sequence ID

Pluto's architecture 3/3

- Design decisions:
 - Perl
 - Originally: functional design, no object orientation
 - No multithreading
 - No use of floats or other number formats with decimals!

How was it built?



The beginning

- One strong leader
- Lots of consultants (developers)
- Focus on few areas at a time
- Do only what we need right now, the rest we take another day
- When we had spare time, we guessed what new demands there would be and acted accordingly

Code standard

- C / Pascal like code!
- No fancy Perl coding!
- No smart programming!
- Comment where needed!
- KISS (Keep It Simple Stupid)
- Standard SQL

Paranoid coding

- Defensive programming: trust no one
- Assert the impossible
- Crash, don't trash
- Log everything
- Contract programming to compensate for weak typing

Paranoid coding

```
contract('do_sell_current_holdings')
  ->in(\&is_person,\&is_date)
  ->out(\&is_state)
  ->enable;

sub do_sell_current_holdings {
  my ($person,$date)

  ...

  if($operation eq "BUD_") {
    @tasks=("GSBB","DACE");
  } elsif($operation eq "SF0A") {
    @tasks=("BSFA");
  } else {
    LogExit("4708: Operation not supported: $operation");
  }

  ...

  return $state;
}
```

Peering

- All code changes are reviewed and tested by another developer
- Check that:
 - the bug is solved
 - the code does not do anything illegal
 - tests pass
 - code standard is followed

Pluto a few years ago...

- A stable production system
- BUT!
- A procedural system (no object orientation)
- Lots of cargo code
- Lots of hidden dependencies
- Lack of automated regression tests

Pluto today

- Pluto is undergoing continuous refactoring:
 - *Modulizing* the system
 - Hunt down redundancy
 - Gather code into Perl modules (classes)
 - Use CPAN
- Write unit tests for all new code
- Follow code standard
- Use object orientation where it fits
- Try new programming techniques...

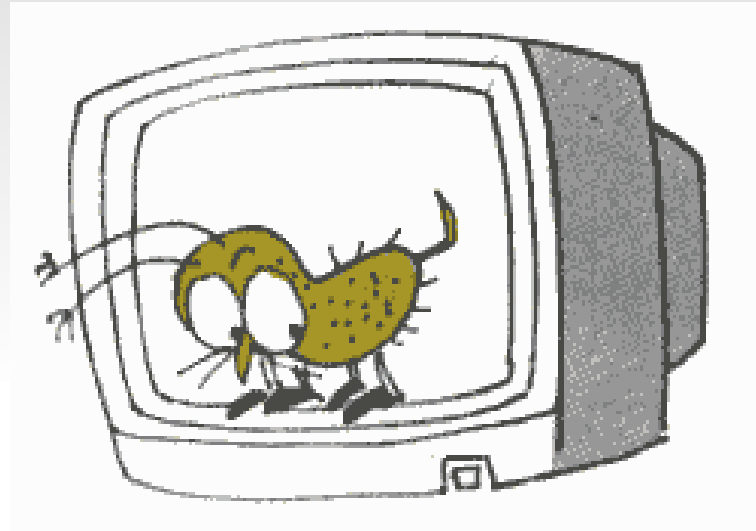
Performance

- Code profilers
- Cache profiler
- SQL profiler:
 - Pluto had a SQL profiler when none existed for Perl
- All executing programs are timed
- Multiprocessing (fork) at critical places
- Optimised database (indexes, smart SQL queries)

The bleeding edge

- Automated testing
- Contract programming (kind of)
- Open-sourcing
- Smart logging

Testing



Testing 1/4

- Test-driven development (kind of)
- Using Perl test files (*.t) and the TAP protocol
- Fast growth:
 - as of 2004-04: less than 10 test files, 0 unitests
 - as of 2006-10: 260 test files, 5000 unitests
 - as of 2007-04: 350 test files, 10000 unitests
- Tests are automated
- Run through every night in every branch

Testing 2/4

- Different sorts of automated tests:
 - test a sub
 - test a program/batch against a specific database
 - test a server
 - test sequences of batches
- Code coverage

Testing 3/4: tools

- Most tests involve a database filled with proper data
 - We have a tool to extract all data related to a person from one database and dump it into a file...
 - ...and a tool to inject the data from this file into an other database
 - Test sequences do that all the time
 - Useful for reproducing bugs
- A tool to record and replay sequences of batches and simulate the daily run schedule
- Various simulators mocking systems to which Pluto is connected

Testing 4/4

- Once the code is released, the test group takes over:
 - 15 persons
 - plenty of test environments with relevant data
 - lists of requirements to validate
- Black box testing at various levels:
 - application test
 - integration test
 - acceptance test
 - pre-production test

Contracts



Contract Programming

- Perl is a weakly typed language
- We compensate that by using contracts
- ie dynamically compiled subroutine wrappers that validate input arguments and results
- This is in fact much more accurate than strong typing:
 - You get type control AND code assertion, all at once

**Smart logging,
open-sourcing
and more**

Smart logging

- Code is full with debug output, off by default
- Debug output is added to the code during early development, and stays there.
- We define a way to selectively turn on debug logging in specific parts of the code, or based on specific conditions
- Like an intelligent filter on top of debug logging

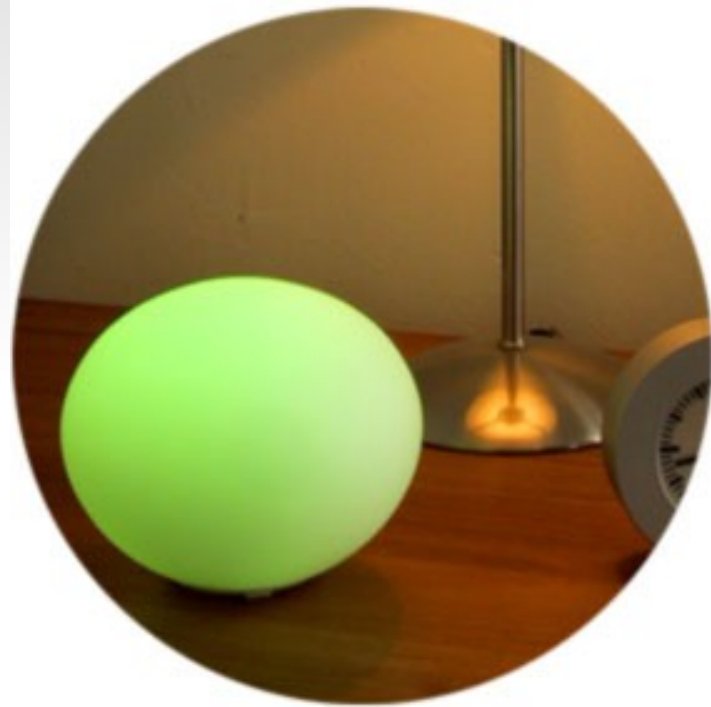
Open-sourcing

- Myndighet = code is public
- Part of our code base is generic enough to be used by others
- We put it on sourceforge/CPAN
 - It gets tested by others
 - CPAN testers
 - Improvements are suggested
 - The code gets better

More stuff

- SCRUM
- Continuous integration
- And...

Ambient orb, mmm....



Reflexions on dynamic languages

Advantages of using Perl

- High level language
- Regular expressions
- Extremely fast development timeline
- Integrates well with UNIX/Linux and Oracle
- Clean code (with code standard!)
- Advanced language features (closures, runtime hacks...) enables more flexible refactoring
- CPAN

Drawbacks of Perl

- Ugly syntax
 - versus code standard
- Weakly typed
 - versus contracts
- Sometimes slow
 - but not the slowest link (database)
- Integrates weakly with Java
- No serious obstacles...

Pluto's future

- Major functions gets extracted out and moved to other systems
- A corporate policy of using Java as main implementation language
- Hard to recruit experienced Perl developers
- Likely that Pluto will slowly get re-implemented using a non-dynamic language

Questions?

