

O'REILLY®

# Software Architecture

ENGINEERING THE FUTURE OF SOFTWARE



[bronstee.com](http://bronstee.com)  
the source site

[softwarearchitecturecon.com](http://softwarearchitecturecon.com)

#oreillysacon



# Taming the Cowboys

or:

## How to Save a Doomed Project by Refactoring it



O'Reilly Software Architecture Conference  
March 19, 2015, Boston, MA  
Ghica van Emde Boas  
[emdeboas@bronstee.com](mailto:emdeboas@bronstee.com)

# Content

- The project
- Why was it a good project
- Why was it bad software
- Why not redevelop
- Refactoring challenges
- Questions and discussion





# Why me?

- It was my gray hair and my knowledge of PHP and MySQL, which brought me into this project.
- There are not too many of those, the project manager said, who was asking for my help to review the project.
- My CV:
  - 30 years with IBM, Relational Database development, Large Java Frameworks, Software Architect.
  - Founded Bronstee.com, Consultant for IBM SanFrancisco (Java Framework).
  - Employed by: QAD, Backbase.
  - Wrote books about PHP, MySQL (in Dutch), and the Backbase JavaScript framework.



# The School for which the Software was Developed

- A vocational college in the southern part of the Netherlands.
- The college is a merger of 23 schools.
- > 20,000 students in total, budget €120,000,000
- You can learn to become a baker, hairdresser, carpenter, cook, plumber, and so on.
- In 3 of these schools they had introduced new web-based software.



# The Project: Planning & Scores (P&S)

- Software where a web-interface allowed students and teachers to view or enter the information relevant to them about:
  - Attendance of the students,
  - Their schedules,
  - Notes from teachers,
  - Their grades and study progress,
  - And so on.
- The college wanted to roll out the software to the other schools and add more functionality.
- The software: MySQL database, PHP backend.



# Let's look at some Functionality

>> *Registration of Attendance*

>> *Study Results*

**Geroosterd**      **Ingepland**      **18 Juni 2010**

8:00

8:30

9:00 MSK 09:00-10:00 8:56

9:30

10:00 TSV (Theorie) 10:00-10:30 Workshop 10:00-11:15 inschrijvingen: 24 / 24

10:30

11:00 TSV (Theorie) 10:45-11:15

11:30 MSK 11:15-12:45 12:02

12:00

12:30

Agenda   Personalia   Persoonskenmerken   Werkervaringen   POP/PAP   Notities   Scores   Rapportages   Toetsen

Beoordelingen   Voortgang

Helpdesk

Selecteer onderwijsproduct

Onderwijsproduct (naam/code)	Stp.	Resultaat	Datum	
Engels 2	HM-ENG 2	3		Selecteer
✓ Engels 2a	HM-ENG 2a	Behaald	02-08-2010	Selecteer
Engels 2b	HM-ENG 2b			Selecteer
✓ Examenopdracht 22	HM-EX 22	1	Voldoende 03-08-2010	Selecteer
✗ Examenopdracht 23	HM-EX 23	2	3,00 03-08-2010	Selecteer
Examenopdracht 25	HM-EX 25	1		Selecteer

**Zoeken**

Gebeurtenis

Type

Selecteer ...

Groep

Uv-S09-03

Wissen   Zoeken 2

**Legenda**

- Geroosterde lessen
- Vervangende lessen
- Workshop
- Mondeling 3
- Coachingsgesprek
- Toets / Herkansingstoets
- Assessment
- Meesterproef
- Externe activiteit
- Student gebeurtenis

**Geroosterd**      **Ingepland**      **4 Juni 2010**

8:00

8:30

9:00 MSK 09:00-10:00 8:51

9:30

10:00 TSV (Theorie) 10:00-10:30 Workshop 10:00-11:15 inschrijvingen: 24 / 24

10:30

11:00 TSV (Theorie) 10:45-11:15 Coachingsgesprek (BPV) 10:30-11:30 inschrijvingen: 24 / 5

11:30 MSK 11:15-12:45 12:55

12:00

12:30

13:00

13:30

14:00

14:30

15:00

15:30

16:00

Ma Di Wo Do Vr Za Zo

31 1 2 3 4 5 6

7 8 9 10 11 12 13

14 15 16 17 18 19 20

21 22 23 24 25 26 27

28 29 30 1 2 3 4

5 6 7 8 9 10 11

Juli 2010

Ma Di Wo Do Vr Za Zo

28 29 30 1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30 31 1

2 3 4 5 6 7 8

Augustus 2010

Ma Di Wo Do Vr Za Zo

28 27 28 29 30 31 1

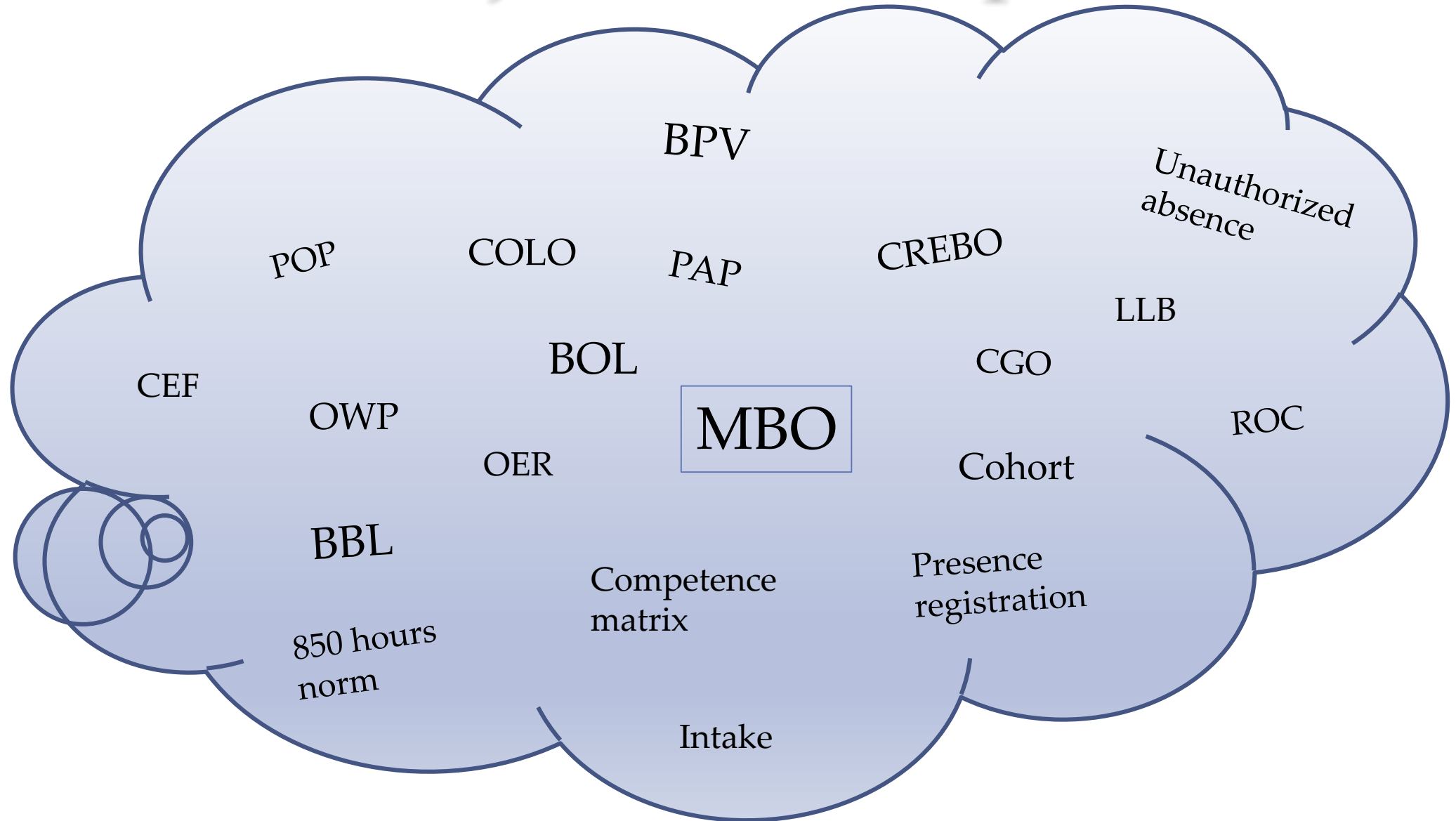
2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

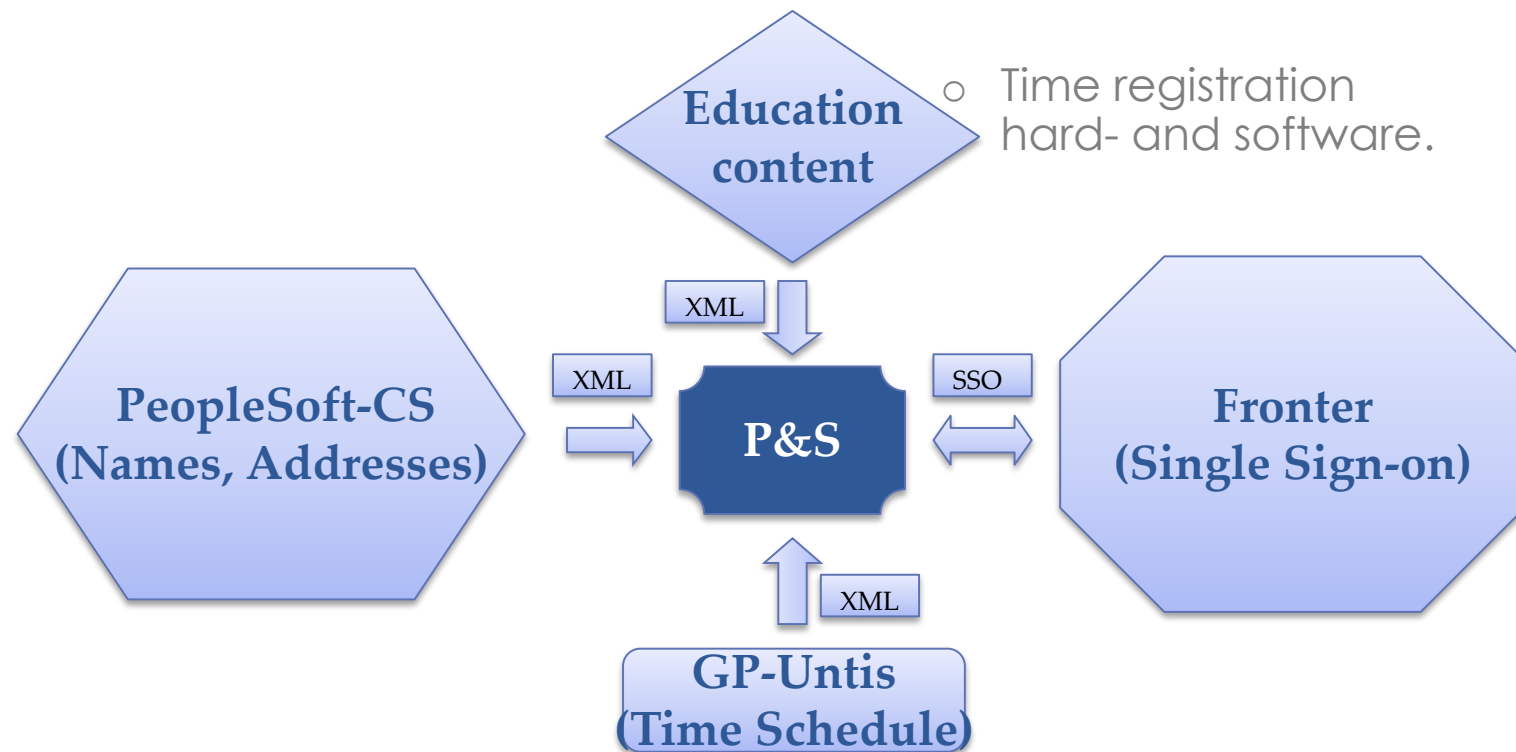
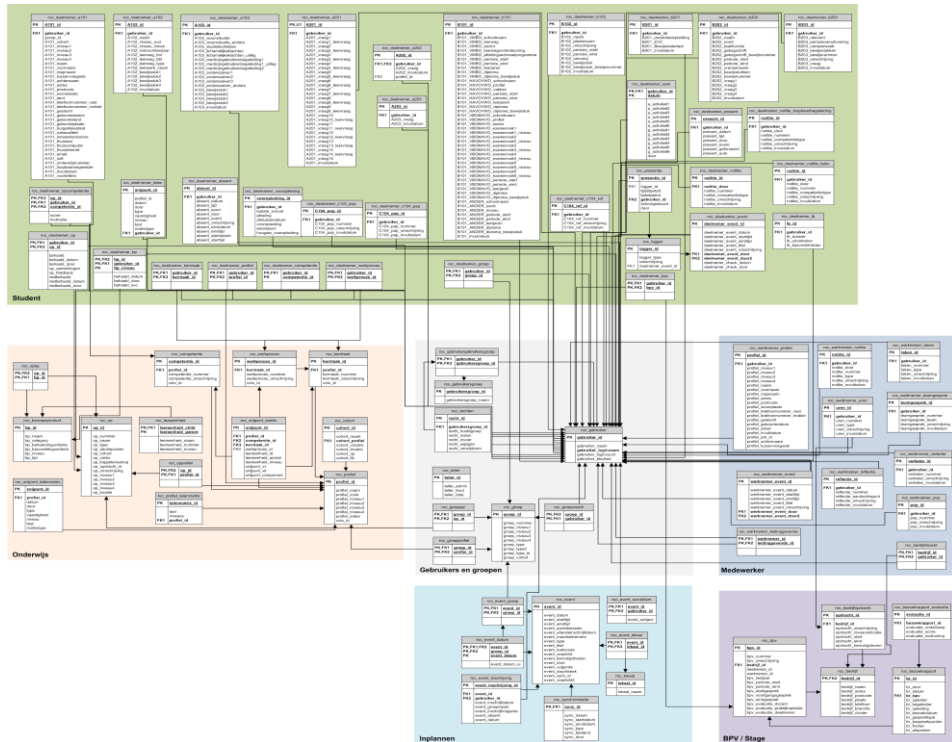
>> *Individual Schedule of Classes*

# The Subject is Complex...



# Project Architecture

- The project was conceived as a web-interface to various other software:





# The Good Parts

- The functionality of the prototype was exactly what the school wanted.
- There was a strong advocate within the school. He was also the designer of the functionality.
- The project manager was on good terms with the highest management in the school, while he could talk about technical stuff to the developers and IT-staff.
- The young programmers were passionate about getting things done.



# Why was the Software Bad?

- No structure, such as model- view- controller.
- Lots of copy-paste code.
- No objects, no classes.
- No normalization of the database.
- There were **800** separate calls to the database.
- No attention to security issues.
- No testing.
- No code versioning management.
- No development procedures.



# I did a review and then I was asked to help execute the recommendations:

- Make a functional description and a data model. ☹️
- Normalize the database. ☹️
- Implement test procedures 😊, automated if possible. ☹️
- Implement code versioning. 😊
- Implement Bug tracking. 😊
- Restructure the code into modules with object-oriented interfaces. ☹️



# Main Requirements

1. Roll-out the software to all 23 schools
2. Make the software and the database secure
3. Migrate from *MySQL* to Oracle
4. Many functional enhancements

**No interruption for existing users!**



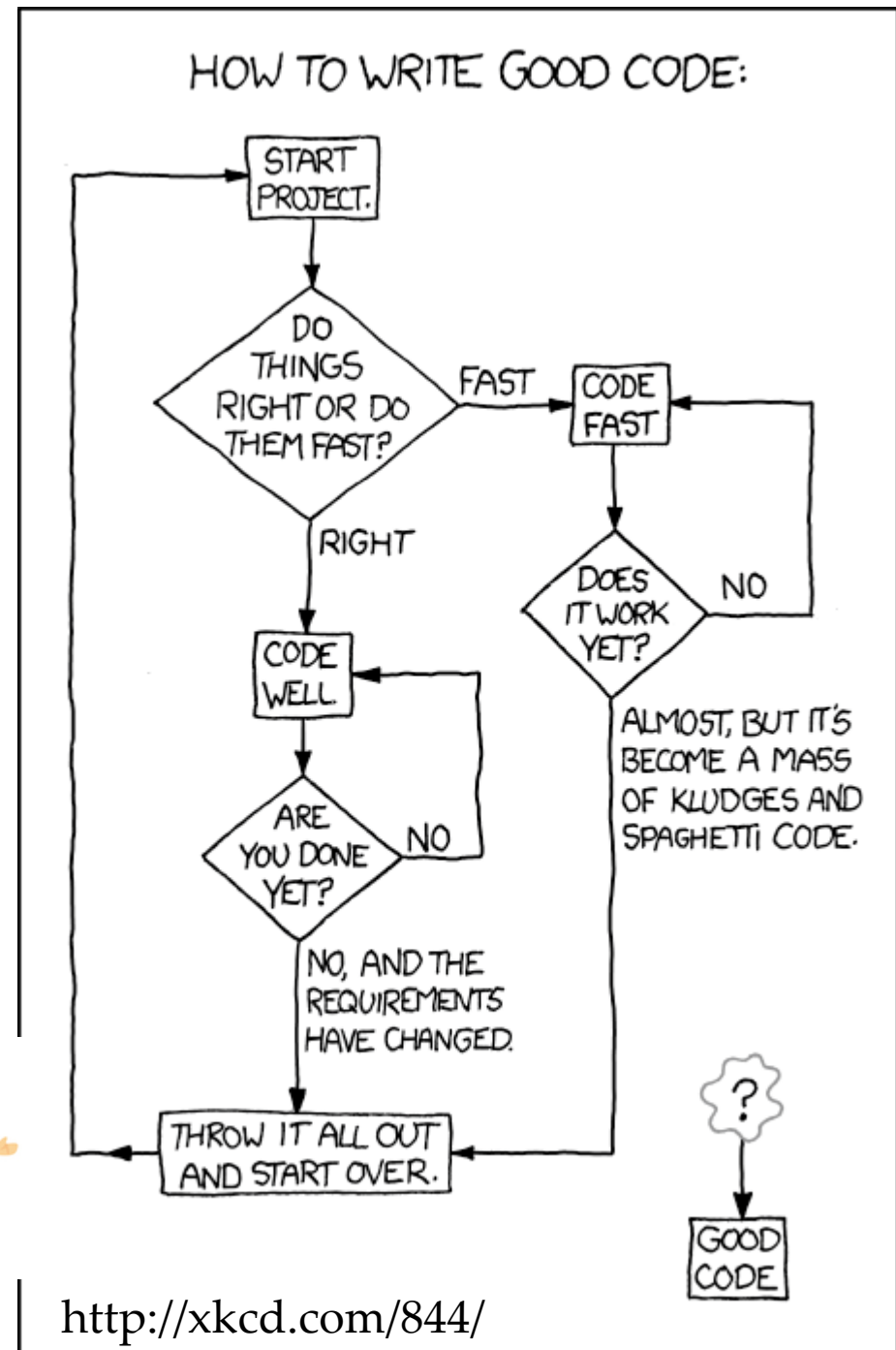
# Questions to Consider

- Given this project, do you think that you would have taken the job?
- Maybe you will change your mind during the talk...
- Look at the refactoring approach we took. Specifically for the conversion Oracle...
- What would you do differently?



# Rewrite the code?

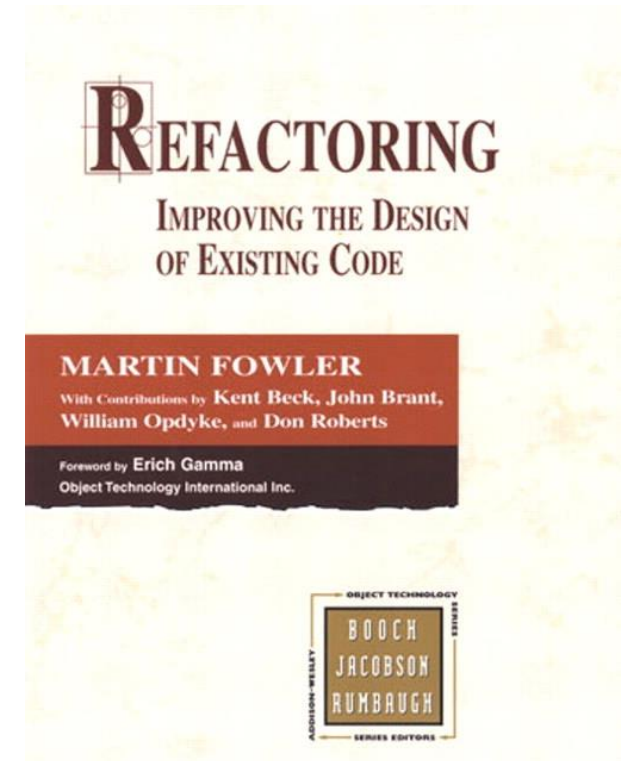
- It was clear that restructuring would cause a rewrite of the code.
- With inexperienced programmers there was a **high risk** that the quality would not be significantly better.
- Development of new functionality would be delayed.
- High cost, because maintenance of the old codebase and parallel development of new code would be required.



# Refactoring

- **Refactoring** is a controlled technique for improving the design of an existing code base.
- **Its essence is applying a series of small behavior-preserving transformations, each of which "too small to be worth doing".**
- However the cumulative effect of each of these transformations is quite significant.

Martin Fowler made refactoring well known with his book: "Refactoring, Improving the design of existing code"



# But... How do you Refactor PHP code?

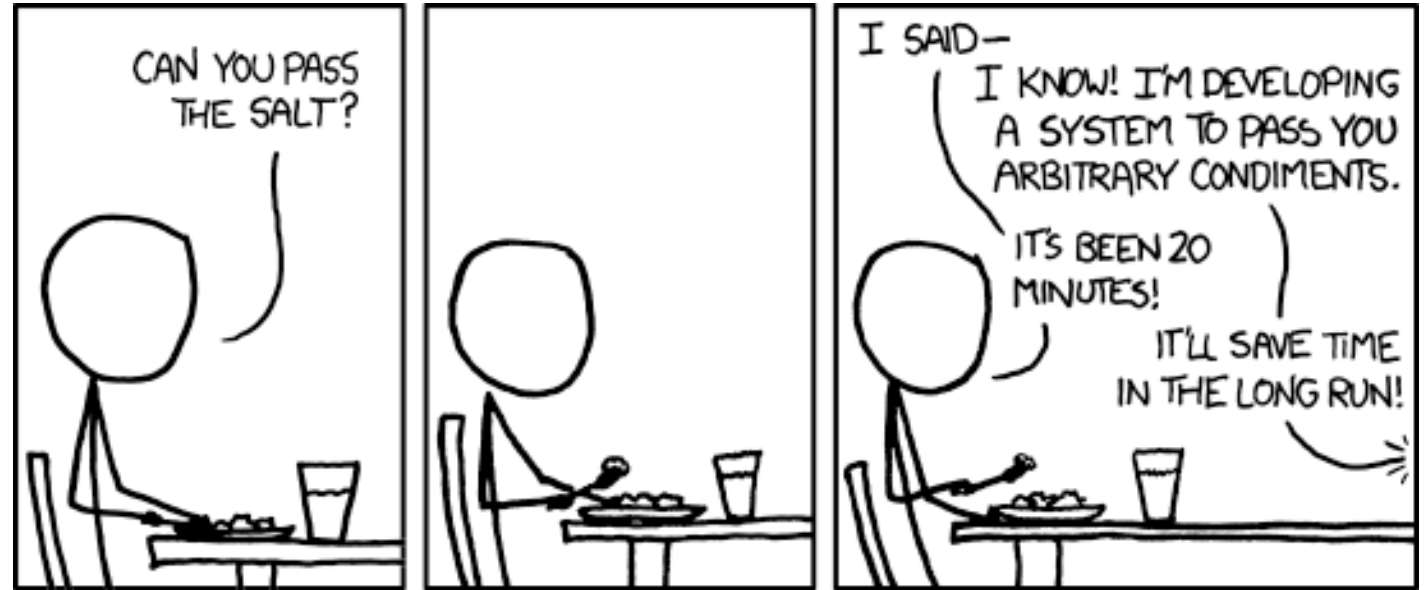
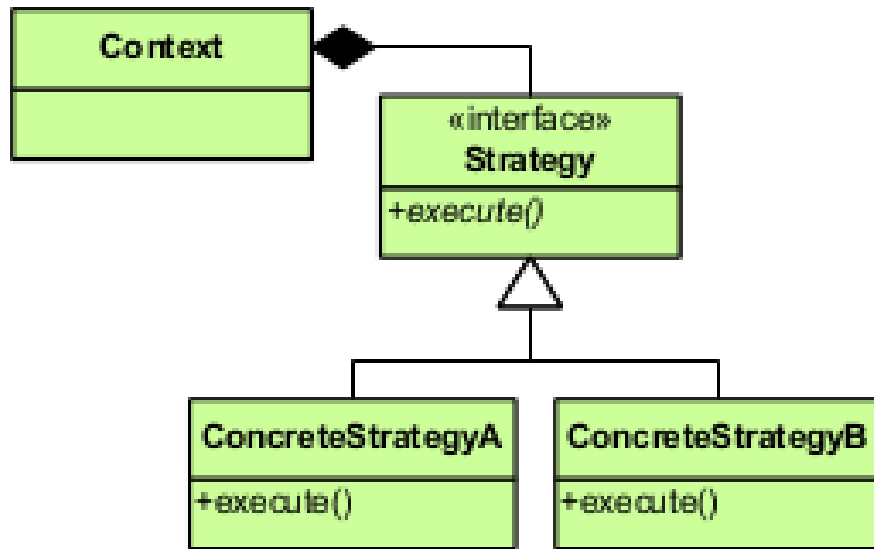
- Refactoring in Fowler's book is about modifying **class hierarchies**. Well researched and used for Java code.
- PHP code can be object-oriented, to a large extent. **Our PHP code did not have a single class.**
  - Therefore: **Use the spirit of the refactoring idea, not the rules.**
  - No refactoring of PHP code was done initially. In the spirit of refactoring we did evolutionary improvement later.

And... How do you Provide Added Value to the School while Refactoring?





# Refactoring by Abstraction



<http://xkcd.com/974/>

## Strategy pattern



# Refactoring Mantra

- Every step is small.
- **Every step has added value.**
- **No refactoring of code that does not need to change because of new or changed functionality.**
- The refactoring is done in a way that it can **accommodate the new functionality by adding just enough abstraction.**
- The application is functioning as normal after every change.
- Less risk, because there is always a working version of the application that can be rolled-out.
- Less deadline stress, because also with just a partial refactoring, improvements and bug fixes are available.



# Bug Tracking, Code Versioning, Testing.

## Bug Tracking ☺

- Mantis for development problems,
- Topdesk for user problems and new feature requests.

## Code Version Control ☺

- Code version control – Subversion (SVN)

## Implement test procedures ☺

- Automated tests where all testing should occur via user interaction is theoretically possible (tools exist, we tried BadBoy).
  - Too time consuming, large effort.
- A testing group was formed.
- A set of test scripts was made.
- Many old bugs were found.



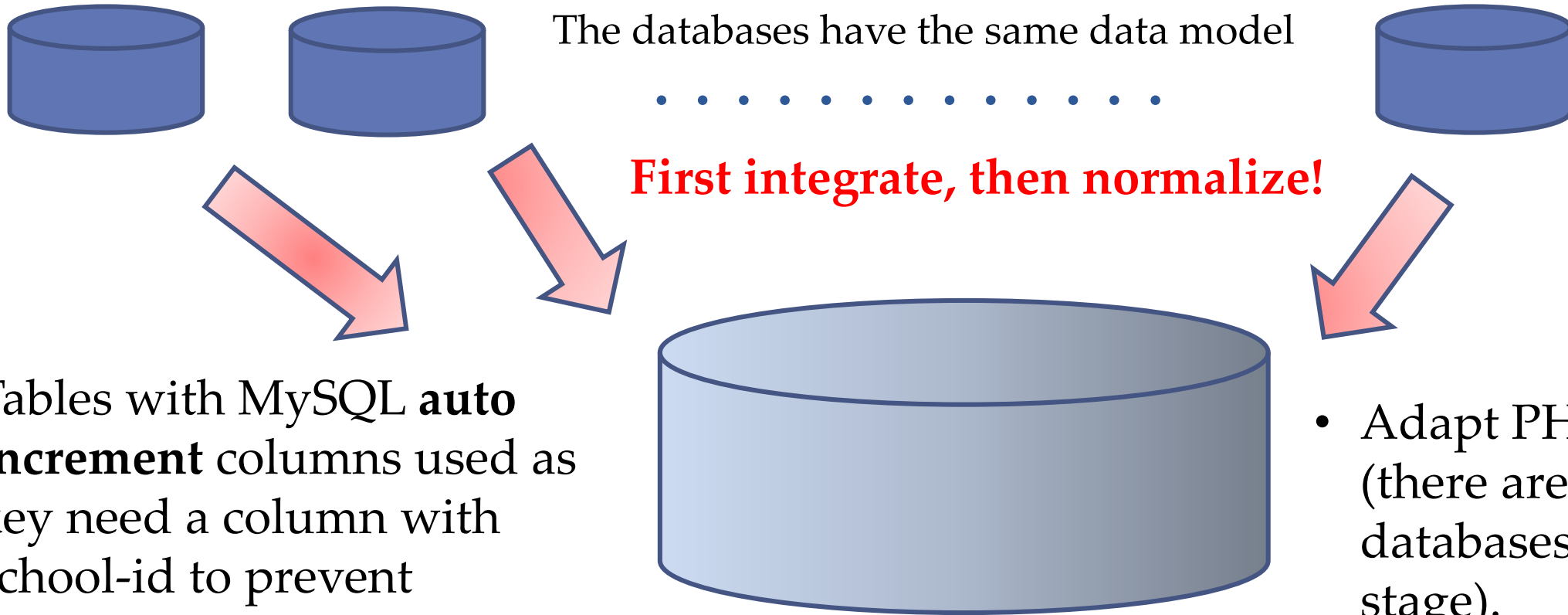
# Data Model , Normalization, Functional Description

- We found duplicate data, Data stored in vectors, columns that should have been rows, tables without keys, etc.
- But: If we would normalize the data, what would have to change in the code?
- It was hard to see what the impact would be and how to control it.
- Therefore: **Database normalization was too risky** and would stop new development for too long.
- A data model was easy to make from the current database using Microsoft Visio. 😊
- The model itself was not so promising. 😞
- **Only normalize new data for new functionality.**



# 23 -> 1 Database conversion

*Important first step to allow development on one database and one code base!*



- Tables with MySQL **auto increment** columns used as key need a column with school-id to prevent duplicate keys.
- We ended up defining school-id columns for all tables

- Adapt PHP code (there are still 3 databases at this stage).

- Testing on a merged database and on the separate databases!

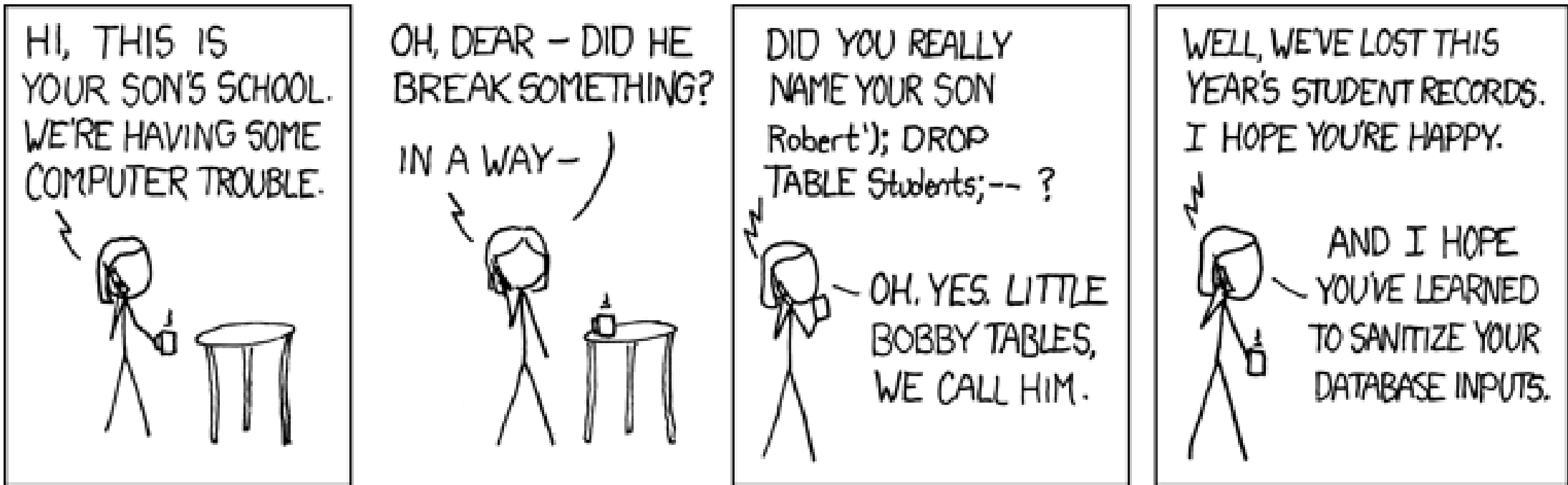


# Performance Blues

- After integration of all databases into 1, the performance of MySQL became an issue.
- What did we do?
  - Convert from MyISAM to InnoDB, the transactional storage engine of MySQL. InnoDB has many more options for tuning.
    - Make table buffer size large enough.
    - Activate the slow-query log and analyse the results.
  - Migrate to a larger, faster new server.
  - Rebuild the tables, to let MySQL sort the rows in an optimal fashion.
  - Look at COUNT(\*) queries without WHERE. Change to EXIST where possible.



# Security



[HTTP://XKCD.COM/327/](http://xkcd.com/327/)



# Next Step: Database Access Abstraction

Why? To prepare for implementing better **security** and to make **transition to Oracle** possible.

We considered using a PHP extension. There are three that cover MySQL: **mysql\_**, **mysqli** and **PDO**.

- The `mysql_` extension is what was used in P&S. It offers no OO interface, no prepared statements (data binding), no transactions.
- The `mysqli` extension offers the best functionality, but only for MySQL
- PDO is an abstraction layer that offers functionality for many databases, including Oracle.



# Typical PHP example from P&S

```
$sql_deelnemer_result = sqlq("SELECT * FROM $roc_db[A103] WHERE gebruiker_id = '$id'");
```



DB access,  
Already slightly  
sanitized, through use  
of the sqlq() function.

No separation of  
Model and View



HTML output

```
function editDeelnemer($valmessage, $valfield) {
    global $roc_image, $id, $roc_db;

    $sql_deelnemer_result = sqlq("SELECT * FROM $roc_db[A103] WHERE gebruiker_id = '$id'");
    $row_deelnemer = mysql_fetch_assoc($sql_deelnemer_result);

    // A103 deelnemer velden
    if (($valmessage == "") && ($valfield == ""))
        if (is_array($row_deelnemer)) foreach ($row_deelnemer as $key => $value) $$key = $value;
    else
        foreach ($_POST as $key => $value) $$key = $value;

    if ($valmessage != "") showvalidationerror($valmessage, "100%");

    $access5 = checkGroupAccess("5");
    $access3467 = checkGroupAccess("3,4,6,7");

    if ($access5) {
        // (Naam en nummer van je bewijsstuk hiernaast invullen. Bewijsstukken kunnen o.a. zijn, een kopie van je doktersverklaring of va
        ?>

        <form name="form_deelnemer_edit" action="<?php echo "$PHP_SELF?a=dbupdate&id=$id"; ?>" method="post">
            <table border="0" cellspacing="0" cellpadding="0" class="admintable">
                <tr>
                    <td height="24" colspan="2" align="left" valign="middle" class="adminheader">Gezondheid en thuissituatie</td>
                </tr>
                <tr>
                    <td width="140" class="deelnemeredit1" valign="top" align="left">Woonsituatie<br/>
                    <td width="100%" class="deelnemeredit2" valign="top" align="left">
                        <input type="radio" name="A103_woonsituatie" value="1" <?php if ($A103_woonsituatie == 1) echo "CHECK" />
                        <input type="radio" name="A103_woonsituatie" value="2" <?php if ($A103_woonsituatie == 2) echo "CHECK" />
                    </td>
                </tr>
            </table>
        </form>
    }
}
```

Direct insertion of  
variable values, open  
for attack!

# Stringing a Query Together

```
$sql_deelnemer_result = sqlq("SELECT * FROM  
$roc_db[A103] WHERE gebruiker_id = '$id'");
```

DB access,  
Already slightly  
sanitized, through use  
of the sqlq() function.

No separation of  
Model and View

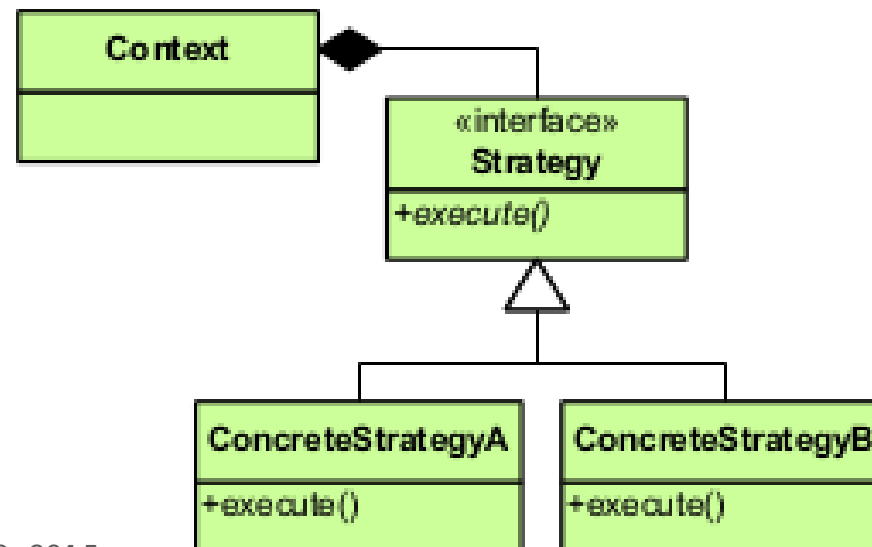
HTML output

```
// A103 deelnemer velden  
if (($valmessage == "") && ($valfield == ""))  
    if (is_array($row_deelnemer)) foreach ($row_deelnemer as $key => $value) $$key = $value;  
else  
    foreach ($_POST as $key => $value) $$key = $value;  
  
if ($valmessage != "") showvalidationerror($valmessage, "100%")  
  
$access5=checkGroupAccess("5");  
$access3467=checkGroupAccess("3,4,6,7");  
  
if ($access5) {  
    // (Naam en nummer van je bewijsstuk hiernaast invullen,  
    ?>  
  
<form name="form_deelnemer_edit" action="<?php echo "<?php echo $PHP_SELF . "/>";>  
  
    <table border="0" cellspacing="0" cellpadding="0" class="admintable">  
        <tr>  
            <td height="24" colspan="2" align="left" valign="middle" class="adminheader">Gezondheid en thuissituatie</td>  
        </tr>  
        <tr>  
            <td width="140" class="deelnemeredit1" valign="top" align="left">Woonsituatie<br>  
            <td width="100%" class="deelnemeredit2" valign="top" align="left">  
                <input type="radio" name="A103_woonsituatie" value="1" <?php if ($A103_woonsituatie == 1) echo "CHECK" >  
                <input type="radio" name="A103_woonsituatie" value="2" <?php if ($A103_woonsituatie == 2) echo "CHECK" >
```

Direct insertion of  
variable values, open  
for attack!

# Database Access Abstraction

- 1<sup>st</sup> plan: use PDO, the PHP Data Object extension.
  - Can be used for both MySQL and Oracle
  - Allows parameter binding, to shield against SQL injection.
  - ❖ No Oracle-approved database extension
  - ❖ Still MySQL or Oracle specific code needed
- 2<sup>nd</sup> plan: use custom database abstraction layer using a strategy pattern.



# Transform Database connections

```
$connection =  
  mysql_connect($hostname, $username,  
  $password) or die(mysql_error());  
$db = mysql_select_db($dbname,  
  $connection) or die(mysql_error());
```

```
$db = new clsDB();  
$connection = $db->connect($host, $user, $pw, $db);
```

- Convert the other `mysql_` functions such as: `mysql_num_rows`, `mysql_fetch_assoc`, `mysql_free_result` and others if present.

× 50

We noticed that a new connection was often not needed.

No `mysql_` function visible anymore

# Transform Database Access

SQL strings could be very long or part of if-then-else constructions.

× 800 !

We developed a PHP script to help with this conversion

```
$result = sqlq("...");
```

```
$sQuery = "...";  
$result = sqlq($sQuery);
```

```
// $result = sqlq($sQuery);  
$GLOBALS['oDB'] → query($sQuery, $aSubstitutes);
```

Instance of clsDB class, containing DB-connection

No parameter binding initially, because \$aSubstitutes may still be empty!

# Parameter Binding Example

```
$aSubstitutes['bpcode0'] = $bpcode[0];  
  
$sQuery="SELECT bpcategory_child  
        FROM roc_beroeepsproduct_category  
        WHERE bpcategory_parent='0'  
        AND bpcategory_nummer=:bpcode0";  
$GLOBALS['oDB'] → query($sQuery, $aSubstitutes);
```

Note that the code stays functioning, even if parameter binding is only partially done.



Parameter binding applied!

# Security Checklist

- SQL parameters are sanitized before use. This should make **SQL-injection** impossible.
- **LIKE queries**. A hacker could try to mis-use smart search fields and cause queries to take very long.
- Input fields where **malicious HTML** could be entered. Remedy: the *htmlspecialchars* PHP function.
- The **register\_globals** .ini setting. Global variables, whose values could automatically overwrite local variables.
- **\$\_REQUEST** takes GET, POST or COOKIE variables. Using COOKIE variables could be unsafe.
- Remove hidden fields like:  

```
<input type="hidden" name="foo" value="whatever" />
```



# Security Checklist - 2

- The **magic\_quotes** .ini setting. Relates to automatic escaping of ' and ". Must be off.
- Use **explicit paths** for file or script names, to avoid use of of another script than intended.
- **Command Injection** by executing shell-commands. No problem for P&S.
- **Secure file upload**. Check for example whether an image is really an image. No problem, because no uploads are used.
- **Application errors**. Log all PHP error messages.
- **Access Rights. Single-sign on. Session hi-jacking.. X-site scripting and code injection. URL rewriting.**





# New Functionality

- For new modules, which were made after the first restructuring, we made normalized data models, on which a great deal of discussion and thought was spent.
- We also made models, views and controllers in the code to interact with these new database parts.
- Old sections interacting with the new modules were cleaned-up.



# New Functionality

- Interfaces for a new release of GP-Untis, the time table scheduler.
- Extensive refactoring of the code related to this interface into a true MVC setup.
- Reporting for **student counting** on October 1<sup>st</sup>, and February 1<sup>st</sup>, this determines the school budget, around 120 million euros.
- Reporting for the **850 hour norm**, the number of hours in a school year that lessons should be offered to a student.
- Signaling of unauthorized absence of more than 3 consecutive school days.
- Logging of changes to the database.
- Registration of study progress and (foreign) language learning.
- New intake form and process for new students.





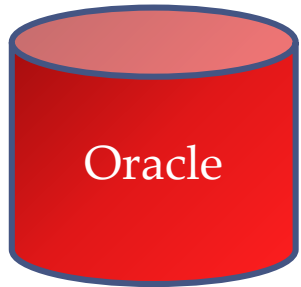
# MySQL -> Oracle



1

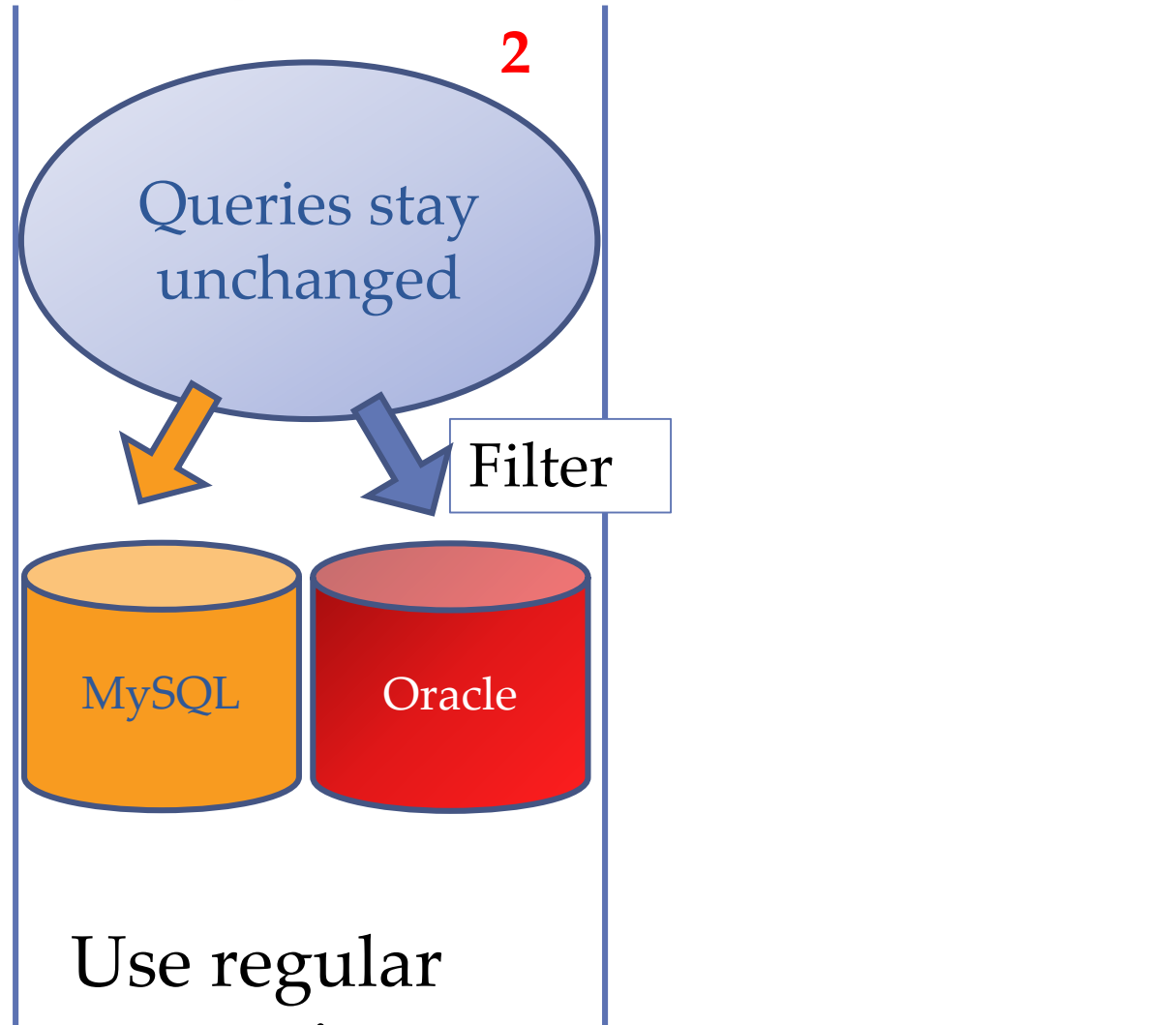
Adapt queries from MySQL to Oracle

Freeze code base





# MySQL -> Oracle

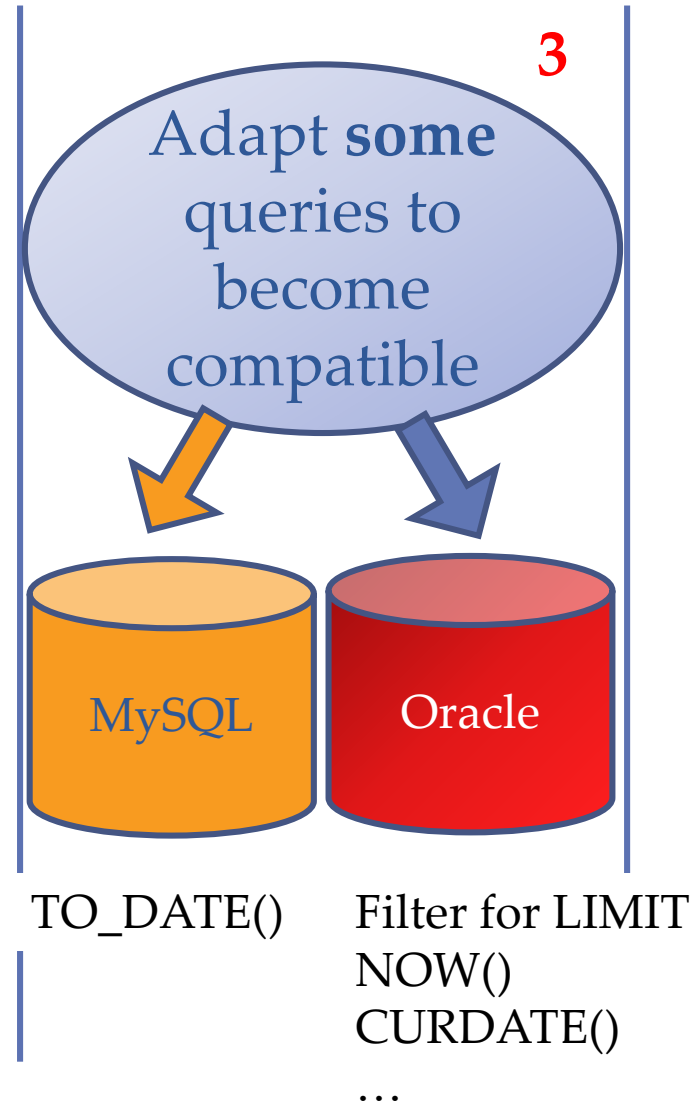


Use regular expressions





# MySQL -> Oracle

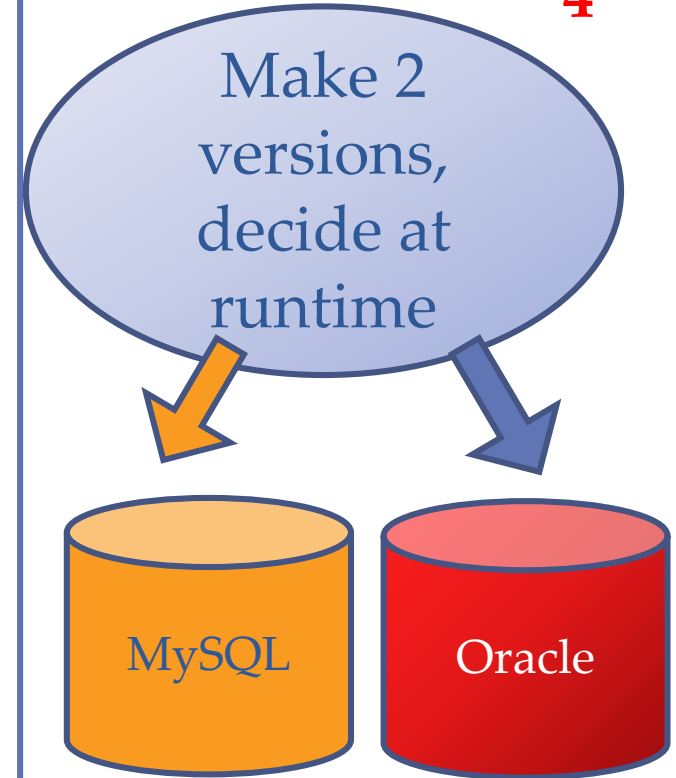




# MySQL -> Oracle

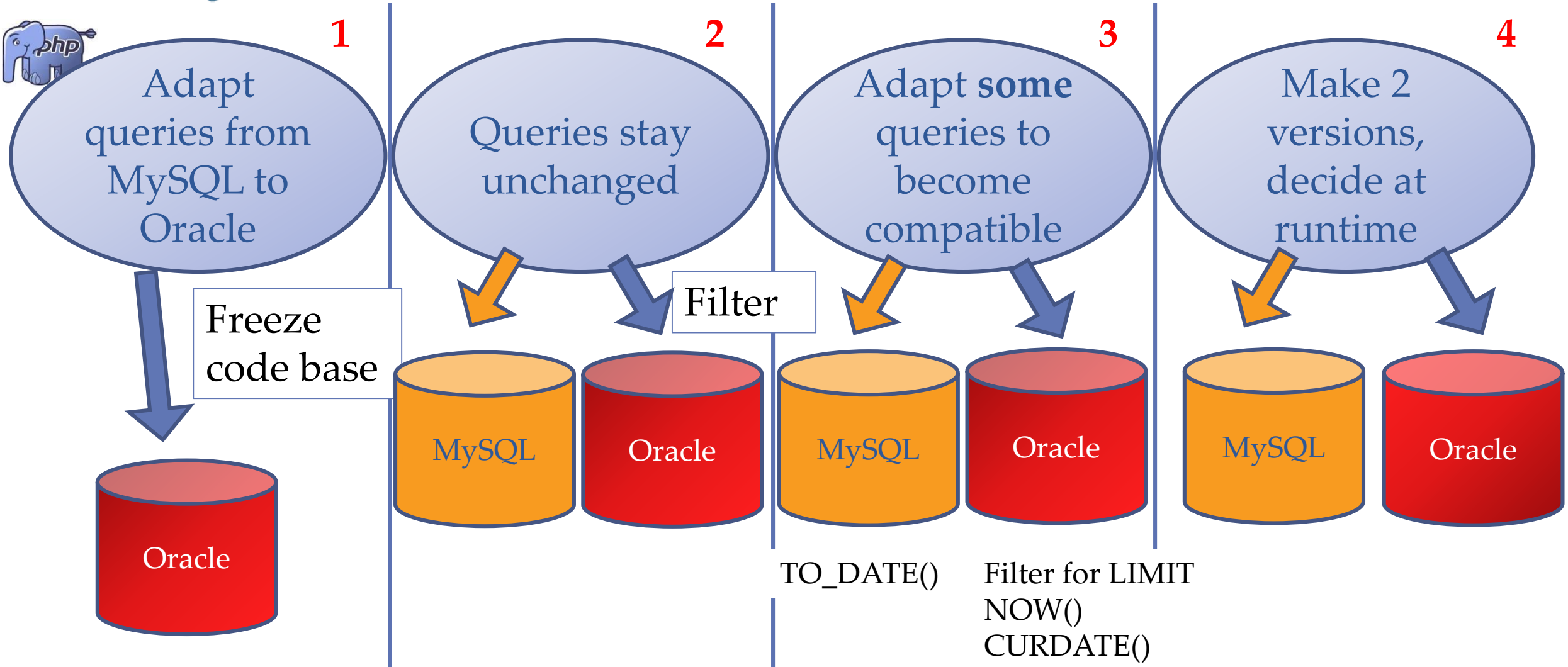


4





# MySQL -> Oracle



• And the winner is ... **3**





# MySQL -> Oracle



Conversion principles:

- **Keep one code base.**
- Made possible by the database layer according to the strategy pattern.
- Make compatibility functions and adaptations to account for naming problems of columns.
- Server management aspects: SVN, DNS, admin of Oracle ...
- Allows to integrate bugs and improvements right until the actual conversion.







# MySQL -> Oracle

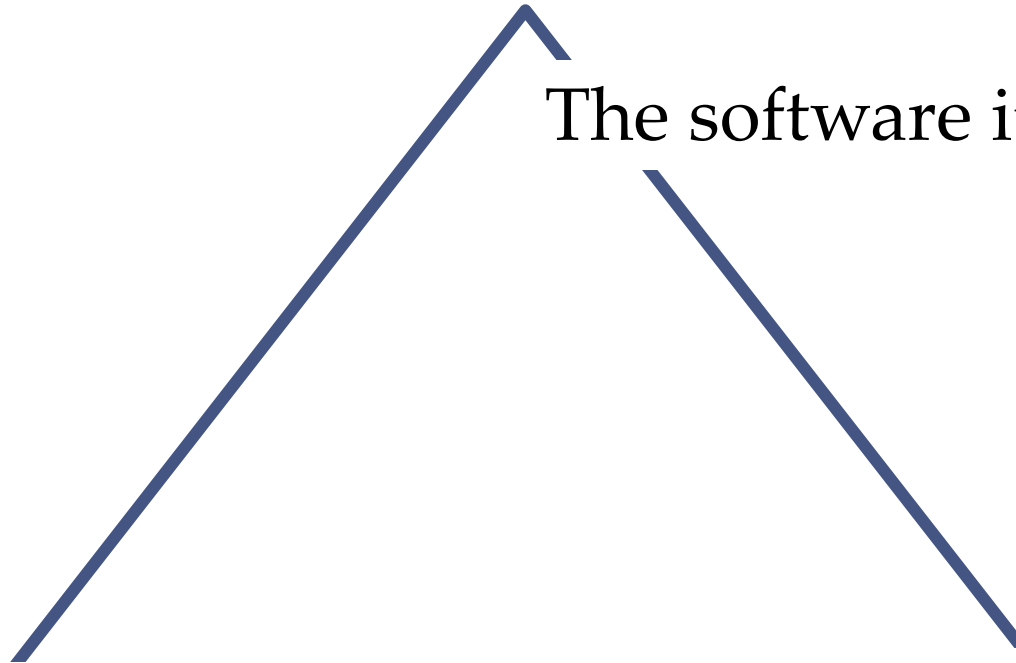


## We made compatibility functions for these items:

- MySQL *mysql\_num\_rows* (number of rows in a table)
- MySQL LIMIT (restrict number of rows returned)
- JOIN syntax not the same for MySQL and Oracle.
- Date formats are different. Date() function cannot be used.
- NOW() function does not exist, replace by SYSDATE().
- **Oracle column names can only be 30 chars long.**
- Naming of functions can be different
- Syntax of database procedures is different.
- MySQL Text columns, convert to CLOB
- MySQL auto-increment columns need extra tables.



# Quality of Software






The use of the software  
(entering correct data  
by teachers etc.)

Management of the  
software: servers,  
databases, networks



# A Review 2 Years Later

- Security is still OK.
  - Performance is good.
  - Users: >22.000 students, parents and teachers.
  - At peak times: thousands of concurrent users.
- 
- 
- All original programmers were replaced.
  - IT management had been replaced.
  - Renewed distrust of code quality (or the new developers just do not understand the complex subject?)
  - **New programmers know a framework: Zend. Why don't we convert?**
  - Because of fusion with yet-another-school, and because standard software becomes available, migration to another package was planned within a few years.
- 

# Conclusion of the Review 2 years later

- The proper functioning of P&S and the amazing transformation from a prototype for 3 schools to a core application for the fusion of 23 schools is, despite the flaws in the code base, for a large part due to a very restrained application of changes to this code base.
- The other side of the coin is, that with their current knowledge and with the current tooling, the programmers would *not* have developed code in this way.



# Threats

- Management not familiar with software development.
- Distrust between developers and IT-service staff.
- Lack of subject-knowledge.
- Programmers who write their own framework.
- Programmers knowing another framework and who want to convert the code to use it.
- Programmers who want to start over to make things “better”.



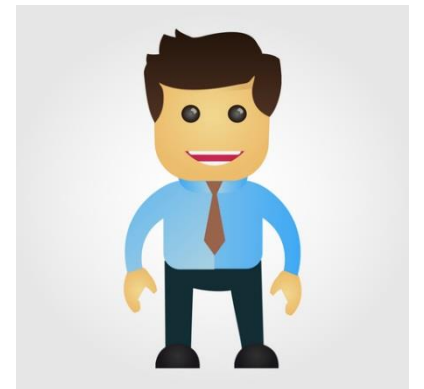
# Other Lessons

- **Continuity is more important than good code.**
- Small steps and refactoring can achieve anything but is very hard to explain to management.
- Cowboy programmers can be very productive but cause distrust that seems to stay with a project forever.
- Programmers with certifications and rigid principles about code quality can be a threat to continuity and productivity.
- Refactoring by abstraction: Look for ways to build new functionality within existing code without actually using it (ex. MySQL -> Oracle).
- Do not touch code that works.



# Again 2 Years Later...

- A consortium of schools developed new software, with Planning&Scores as one of the inputs.
- This software is now used in many vocational colleges in the Netherlands with a similar curriculum.
- It also replaced some parts of Planning & Scores in Eindhoven, but other parts are still used.



# Questions and Discussion

- Given this project, did you think at the start of this talk that you would have taken the job?
- Did you change your mind during the talk?
- Did you think we took the right approach with the conversion to Oracle?
- What would you have done differently?
- Do you have any recommendations for future, similar projects?





# The Problems and Advantages of Being Old and Female

- Because you are old and female, you need a long time to convince programmers that you know what you are talking about.
- Because you are old and female, they do not want to follow your orders.
- Because you are old, you are no threat to their career.
- Because you are old, maybe you cannot code very fast anymore, but actually not that much has changed since 1970, therefore you are faster at understanding new things on an overview level.
- Because you are old and female, the lunch talk about motorcycles and fast cars is not so interesting.
- Finally, it seems that even at the age of 91, you can be hired at a tech job! [http://youtu.be/WV\\_wQN7sUwM](http://youtu.be/WV_wQN7sUwM)



# Acknowledgements

- Willem van Dinther, the great motivator of the project and for being helpful with providing material.
- Jurgen Weisfelt, the project manager.
- The cowboys, who were very productive!



# Links

- PeopleSoft Campus Solutions
  - <http://www.oracle.com/us/products/applications/peoplesoft-enterprise/campus-solutions/overview/index.html>
- Fronter
  - <http://com.fronter.info/>
- GP-Untis
  - [http://www.grupet.at/home\\_en.php](http://www.grupet.at/home_en.php)
- Mantis
  - <https://www.mantisbt.org/>
- TOPdesk
  - <http://www.topdesk.com/us/>
- Subversion
  - <https://subversion.apache.org/>
- Picture of a "Strategy Pattern in UML" by Jason S. McDonald =
  - [http://commons.wikimedia.org/wiki/File:Strategy\\_Pattern\\_in\\_UML.png#mediaviewer/File:Strategy\\_Pattern\\_in\\_UML.png](http://commons.wikimedia.org/wiki/File:Strategy_Pattern_in_UML.png#mediaviewer/File:Strategy_Pattern_in_UML.png)

