
LISTEN AND LOOK
AT YOUR PHP CODE!
Gabriele Santini
Forum AFUP 2010

Gabriele Santini

sqli GROUP

- Architect/Consultant at SQLI
- Contributor to PHP_CodeSniffer
  - So expect a special focus on this…
- Sonar PHP Plugin
  - I have to show you!
- Ex-mathematician :
  - love business modelling
  - love architectures
  - love quality assurance

Static Analysis

- All you can say about your program *without actually execute the code*
  - The rest is also interesting, let's talk about it another time !
- Examples ?
  - Syntax check, coding style, anti-patterns, metrics, OO design analysis, ...
- *What PHP does before executing the code ?*

Levels of analysis

- Lexical analysis
  - Read sources linearly searching for known patterns
  - Convert them to a sequence of tokens

Levels of analysis

- Lexical analysis
  - Read sources linearly searching for known patterns
  - Convert them to a sequence of tokens
- Syntactic Analysis
  - Parse the tokens to find their logical structure

Levels of analysis

- Lexical analysis
  - Read sources linearly searching for known patterns
  - Convert them to a sequence of tokens
- Syntactic Analysis
  - Parse the tokens to find their logical structure
- Opcode (Bytecode) Generation
  - Generates an intermediary code that the Zend Engine will be able to execute

Lexical Analysis

- Tokenizer

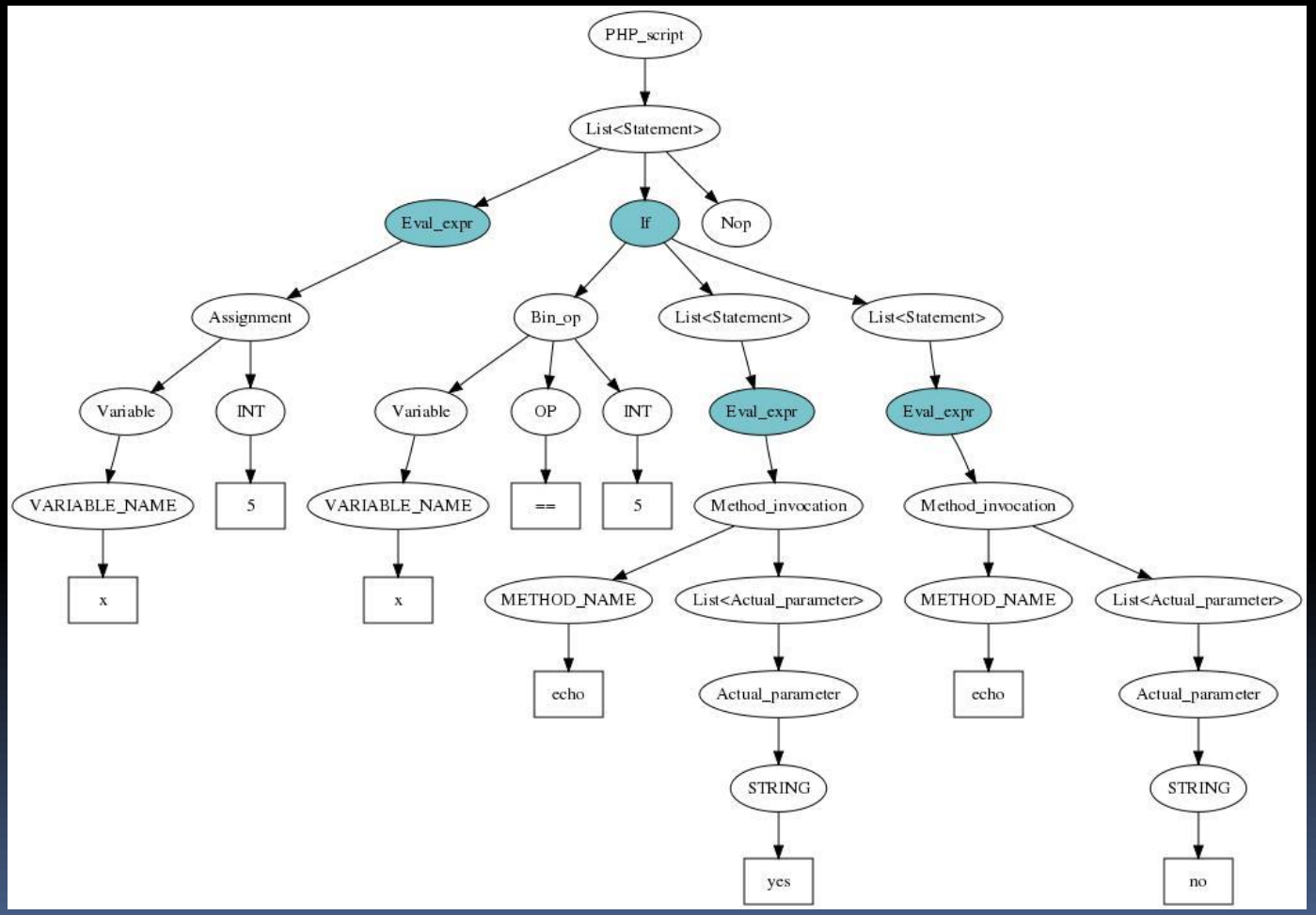
```php
<?php
if (1 < 2) {
    echo "Hello";
}
?>
```

```
1    <?php       T_OPEN_TAG
2    if          T_IF
2                T_WHITESPACE
     (
2    1           T_LNUMBER
2                T_WHITESPACE
     <
2                T_WHITESPACE
2    2           T_LNUMBER
     )
2                T_WHITESPACE
     {
2                T_WHITESPACE
3    echo        T_ECHO
3                T_WHITESPACE
3    "Hello"     T_CONSTANT_ENCAPSED_STRING
     ;
3                T_WHITESPACE
     }
4                T_WHITESPACE
5    ?>          T_CLOSE_TAG
```

Syntactic Analysis

- Produces an AST
  - Abstract Syntax Tree
    - Decompose code in a tree-like form
    - Can be executed once a context is given
    - Used in compilers

Syntactic Analysis



Opcodes Generation

- Mistery tool...

```php
<?php
if (1 < 2) {
    echo "Hello";
}
?>
```

```
line     # *  op                 return   operands
-------------------------------------------------
   2     0  >   EXT_STMT
         1      IS_SMALLER        ~0       1, 2
         2    > JMPZ                       ~0, ->6
   3     3  >   EXT_STMT
         4      ECHO                       'Hello'
   4     5    > JMP                        ->6
   6     6  >   EXT_STMT
         7    > RETURN                     1
```

GIVE US THE TOOLS !

PHP_CodeSniffer

- By Greg Sherwood
  - PEAR library
  - Venerable project
- Code Style
  - But also a lot more
- Works at lexical analysis level
  - Heavily use the tokenizer extension

PHP_CodeSniffer

- Hands on

```
blacksun@blacksun-laptop:/demo$ phpcs test_phpcs.php

FILE: /media/sda1/tests/test_phpcs.php
----------------------------------------------------------------------
FOUND 7 ERROR(S) AND 0 WARNING(S) AFFECTING 7 LINE(S)
----------------------------------------------------------------------
  2 | ERROR | Missing file doc comment
  3 | ERROR | Multi-line function call not indented correctly; expected 4
    |       | spaces but found 0
  7 | ERROR | Closing parenthesis of a multi-line function call must be on a
    |       | line by itself
  9 | ERROR | Missing function doc comment
 20 | ERROR | Line indented incorrectly; expected 4 spaces, found 7
 24 | ERROR | Line indented incorrectly; expected 4 spaces, found 7
 28 | ERROR | Closing brace indented incorrectly; expected 0 spaces, found 4
----------------------------------------------------------------------
```

PHP_CodeSniffer

- Sniffs
  - Classes that detect Violations
    - One or more type per class
  - Grouped in folders by subject:
    - Commenting, Formatting, WhiteSpace
    - Files, ControlStructures, Strings
    - Functions, Classes, NamingConventions
    - CodeAnalysis, Metrics
  - You can create your own!

PHP_CodeSniffer

- Standards
  - Sets of Sniffs that define your coding style
  - Installed :
    - PEAR,
    - Generic,
    - Zend*,
    - Squiz, MySource
    - PHPCS

PHP_CodeSniffer 1.3

- Rulesets XML!

```
<ruleset name="MyPEAR">
 <description>A variation of the PEAR coding standard.</description>
 <!-- Include some additional sniffs from the Generic standard -->
 <rule ref="Generic.Functions.FunctionCallArgumentSpacing"/>
   <message>Please review spacing in function 'sprintf'</message>
 </rule>
 <rule ref="Generic.NamingConventions.UpperCaseConstantName"/>
 <!-- Lines can be 90 chars long, but never show errors -->
 <rule ref="Generic.Files.LineLength">
  <properties>
   <property name="lineLimit" value="90"/>
   <property name="absoluteLineLimit" value="0"/>
  </properties>
 </rule>
 <!- Not so important for us -->
 <rule ref="Generic.PHP.DisallowShortOpenTag">
  <severity>2</severity>
 </rule>
</ruleset>
```

Inside PHP_CodeSniffer

- Sniff class main methods
  - **register()**
    - Make the sniff a listener for the declared tokens
  - **process($phpcsFile, $stackPtr)**
    - Called by the file during parsing when a declared token is found
- File
  - Represents a parsed file
  - Holds the tokens structure and offers convenience methods

Inside PHP_CodeSniffer

- Life of a Sniff
  - DisallowMultipleStatementsSniff

```php
<?php
echo $y;
$x = 10; echo $y;

for ($i = 1; $i < $length; $i++) {
    echo 'x';
}
echo $x;
$y = 2;;
$this->wizardid = 10; $this->paint(); echo 'x';
?>
```

Inside PHP_CodeSniffer

- Life of a Sniff
  - DisallowMultipleStatementsSniff

```
public function register()
{
    return array(T_SEMICOLON);
}
```

```
public function process($phpcsFile, $stackPtr)
{
    […]
```

Inside PHP_CodeSniffer

- Life of a Sniff
  - DisallowMultipleStatementsSniff

```php
<?php
echo $y;
$x = 10; echo $y;

for ($i = 1; $i < $length; $i++) {
    echo 'x';
}
echo $x;
$y = 2;;
$this->wizardid = 10; $this->paint(); echo 'x';
?>
```

Inside PHP_CodeSniffer

- Life of a Sniff
  - DisallowMultipleStatementsSniff

```
<?php
echo $y;
$x = 10; echo $y;

for ($i = 1; $i    $length; $i++) {
    echo 'x';
}
echo $x;
$y = 2;;
$this->wizardid = 10; $this->paint(); echo 'x';
?>
```

$stackPtr

Inside PHP_CodeSniffer

- Life of a Sniff
  - DisallowMultipleStatementsSniff

```
public function register()
{
    return array(T_SEMICOLON);
}
```

```
public function process($phpcsFile, $stackPtr)
{
    $tokens = $phpcsFile->getTokens();
    $previous = $phpcsFile->findPrevious(…);

    if ($previous === false) {
            return;
    }
// Continue =>
```

Inside PHP_CodeSniffer

- Life of a Sniff (2)

```
 // Ignore multiple statements in a FOR condition.
    // First some gym for nested parenthesis
    […]
    if ($tokens[$owner]['code'] === T_FOR) {
                    return;
    }
    […]
    // If the previous semicolon is on the same line we add an error
    // to this file
    if ($tokens[$previous]['line'] === $tokens[$stackPtr]['line']) {
        $error = 'Each PHP statement must be on a line by itself';
        $phpcsFile->addError($error, $stackPtr);
        return;
    }
}//end process()
```

PHP_CodeSniffer

- At SQLI we have some framework standards
  - Zend Framework
    - Based on Thomas Weidner work
  - Symfony
    - In collaboration with Fabien Potencier
  - Waiting for a serious release after 1.3 release

PHP_CodeSniffer

- At SQLI we have some framework standards
  - Zend Framework
    - Based on Thomas Weidner work
  - Symfony
    - In collaboration with Fabien Potencier
  - Waiting for a serious release after 1.3 release
- That's nice, but…
- Where are the standards for the other tools ?
  - I'ld expect a Drupal, Wordpress, Cake official standard

PHP_CodeSniffer

- How far a standard can go in detection ?

PHP_CodeSniffer

- How far a standard can go in detection ?
- Interestingly far for generic PHP Code

PHP_CodeSniffer

- How far a standard can go in detection ?
- Interestingly far for generic PHP Code
- Very far if you know your tool's structure
  - Imagine for example forcing PHP alternative syntax in Symfony views…
  - Or checking for escaping in Zend Views !

PHP_Depend

- By Manuel Pichler
- Functional port of JDepend
  - OO design analysis
  - Metrics visualisation
  - Dependency analyzer
- Works at the syntactic analysis level

PHP_Depend

- How it works

- PHP_Depend first makes an AST off your code
  - A « personal » one, made by PHP objects
  - ASTComment, ASTClosure, ASTEvalExpression, …
  - This is made by the Builder/Parser component
  - Using PHP Reflection

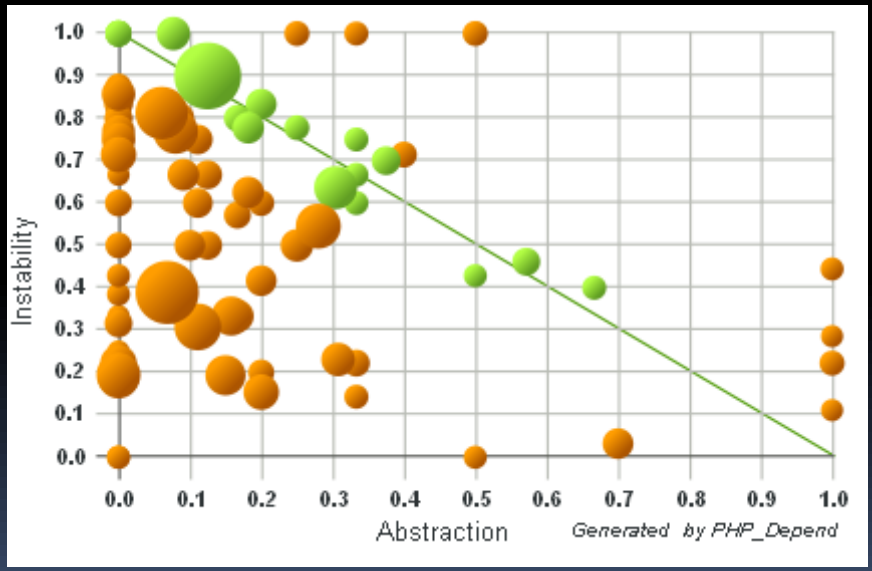
PHP_Depend

- How it works (2)

- Then PHP_Depend can answer questions by « visiting » the AST
  - Task of Metrics Analyzers, that extend AbstractVisitor
  - IOC, the visitor decides what to do according to AST Class : visitMethod, visitForStatement(), …
  - Analyzers can fire listeners during analyze() call
    - To get ongoing informations about the visit process

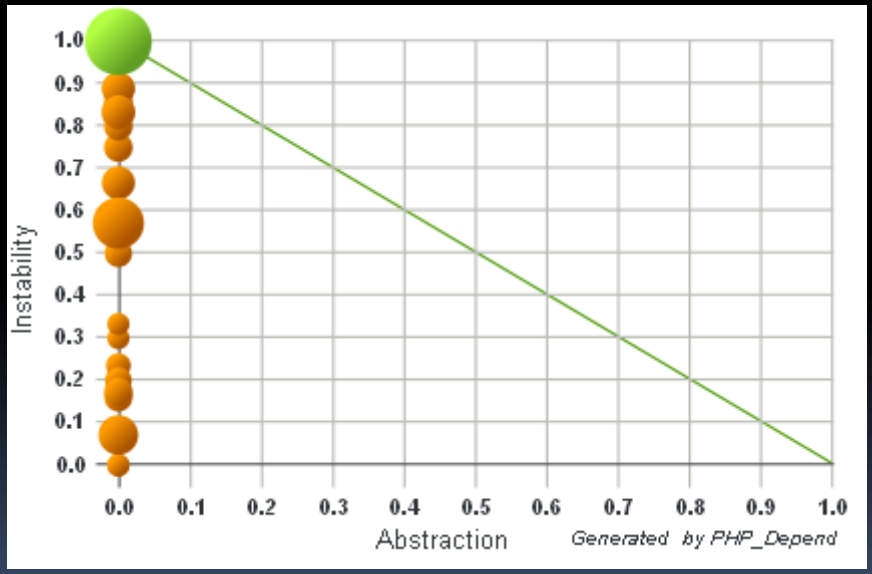
PHP_Depend

- What it gives:
  - The Abstraction/Instability graph



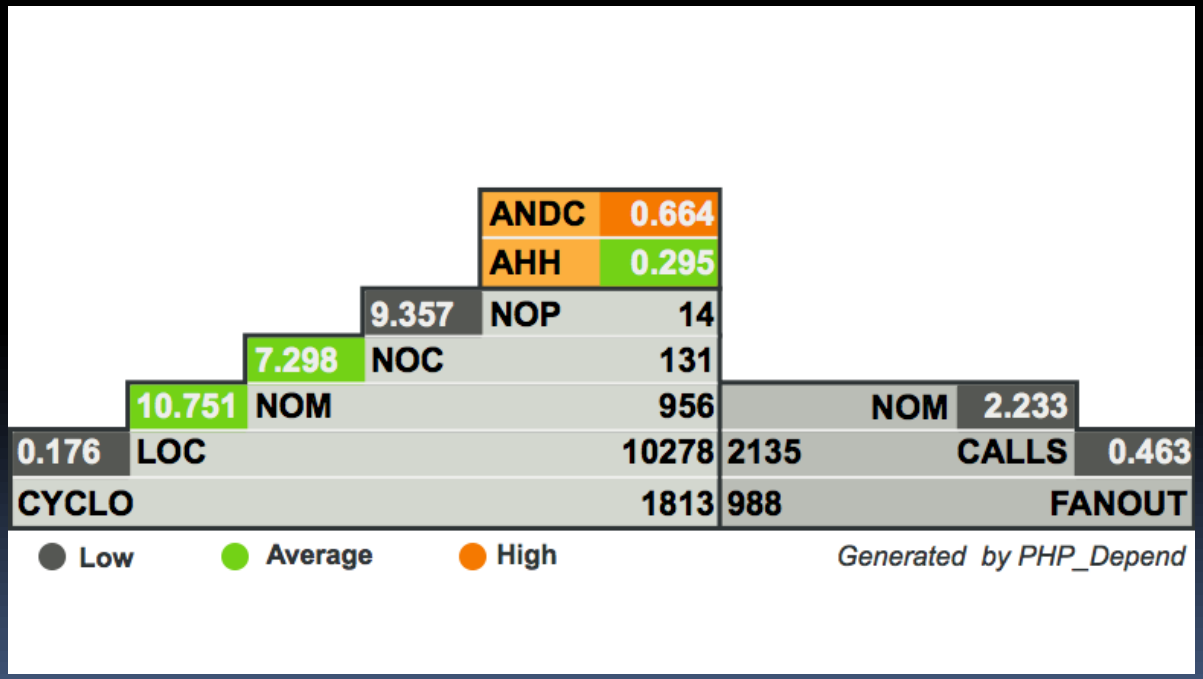
PHP_Depend

- What it gives:
  - The Abstraction/Instability graph



PHP_Depend

- What it gives:
  - The Pyramid !!



PHPMD

- By Manuel Pichler
- Detects rules violations
  - Analog to PHP_Codesniffer
- Works at syntactic analysis level
  - Actually on the same AST
  - Depends on PHP_Depend
  - Has rulesets !

PHPMD

- What it gives:

- Code Size Rules
  - complexities, lengths, too many, ...
- Design Rules
  - OO, exit, eval
- Naming Rules
  - Too short/long identifiers, old constructors, ...
- Unused Code Rules
  - Methods, members, parameters

phploc

- By Sebastian Bergmann
- Simple tool to give basic metrics
  - Fast, direct to the goal
- Works mostly on lexical level
  - But use bytekit for ELOC if it can

phpcpd

- By Sebastian Bergmann
- Simple tool to detect duplicated code
- Works at lexical analysis level
  - Use the tokenizer to minimize differences
    - Comments, whitespaces, ...
  - Takes a minimum number of lines and tokens
    - Encodes according to this
    - Uses an hash table to find duplicates

vld

- Vulcan Logic Disassembler
  - By Derick Rethans
- Works at bytecode level
  - Shows generated bytecodes
  - Calculates possible paths (CFG)
  - Find unreachable code
  - Could be used for code coverage path metrics

vld

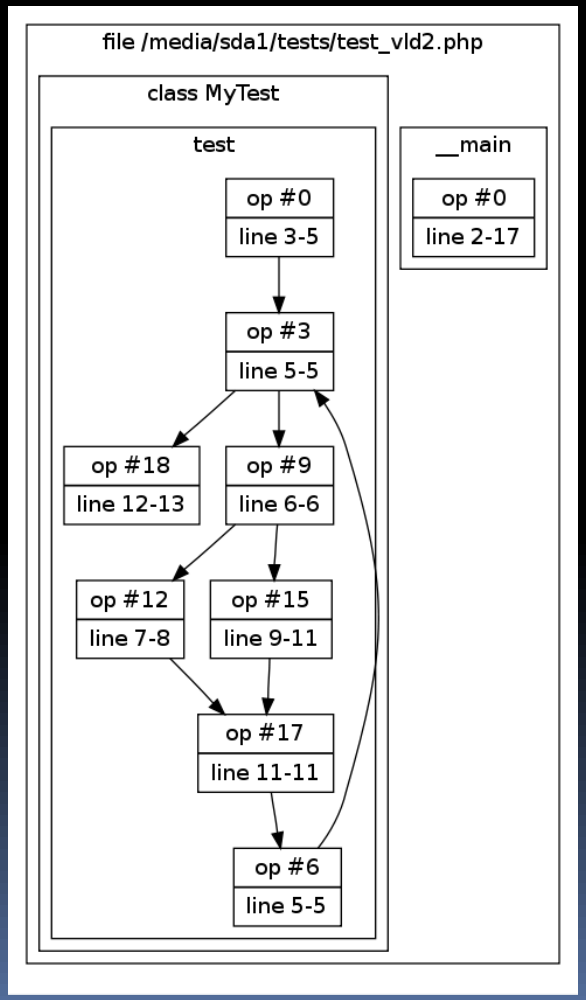
- Output

```
blacksun@blacksun-laptop:/media/sda1/tests$ php -dvld.active=1 -dvld.dump_paths=1 -dvld.sav
e_paths=1 -dvld.save_dir=/media/sda1/tests test_vld2.php
filename:       /media/sda1/tests/test_vld2.php
function name:  (null)
number of ops:  8
compiled vars:  none
line     # *  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   2     0  >   EXT_STMT
         1      NOP
  16     2      EXT_STMT
         3      ZEND_INIT_STATIC_METHOD_CALL                                'MyTest', 'test'
         4      EXT_FCALL_BEGIN
         5      DO_FCALL_BY_NAME                             0
         6      EXT_FCALL_END
  17     7    > RETURN                                                      1

branch: #  0; line:     2-   17; sop:     0; eop:     7
path #1: 0,
```

vld

- Output

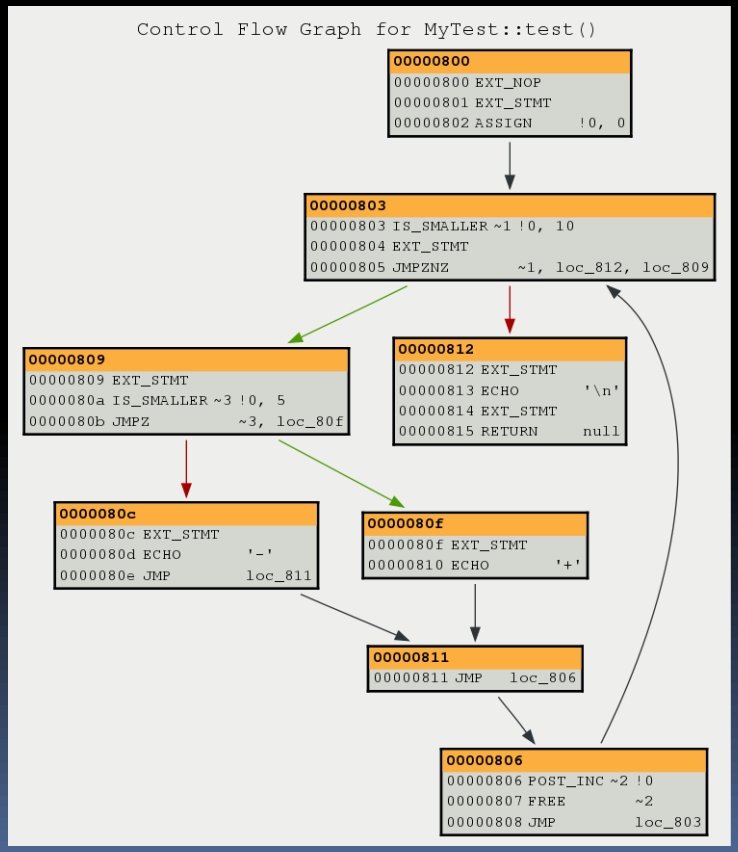

file /media/sda1/tests/test_vld2.php
class MyTest
test
op #0
line 3-5
op #3
line 5-5
op #18
line 12-13
op #9
line 6-6
op #12
line 7-8
op #15
line 9-11
op #17
line 11-11
op #6
line 5-5
__main
op #0
line 2-17

Bytekit

- By Stefan Esser (SektionEins)
- Works at … bytecode level
  - Similar to vld
  - Exposes opcodes to a PHP array
  - bytekit_disassemble_file($filename)
- Can be used directly for a custom script

Bytekit

- CFG visualisation



Bytekit-cli

- By Sebastian Bergmann
- PHP Interface to use bytekit to spot violation rules
  - Initial state
- Implemented rules :
  - Check for disallowed opcodes (example eval, exit)
  - Check for direct output of variables
  - In svn, check for unescaped ZendView

Padawan

- By Florian Anderiasch
- Focus on anti-pattern detection
  - alpha (?)
- Works on syntactic analysis level
  - Based on PHC (PHP compiler)
  - Use an XML dump of the AST PHC generates
  - Makes xpath searches on it

Padawan

- Interesting approach
  - Rules are fairly simple to write
  - Already many interesting tests :
    - Empty constructs (if, else, try,..), unsafe typecasts, loop repeated calls, unused keys in foreach, ...
- PHC not easy to install
- Risk on PHC manteinance

Phantm

- By Etienne Kneuss
- Highly experimental
  - Severe limitation on PHP dynamic features
  - False positives
- Works on syntax analysis level
  - Based on Java tools (Jflex, CUP, Scala)
- Reports violations
  - Non top-level declarations, call-time pass-by-ref, nontrivial include calls, assign in conditional, …
- Exploring Type Flow Analysis
  - Tries to infer types and check for type safety

Conclusion

- Use the right tool for the right job
  - Coding style is better analysed at the **lexical level**
  - OO design is better viewed after **syntactic analyses**
  - Unreachable code after **bytecoding**
- Contribute !
  - Plenty of things still to implement
  - Easy to have new ideas
  - At least use them (you should!) and give feedback

Restitution

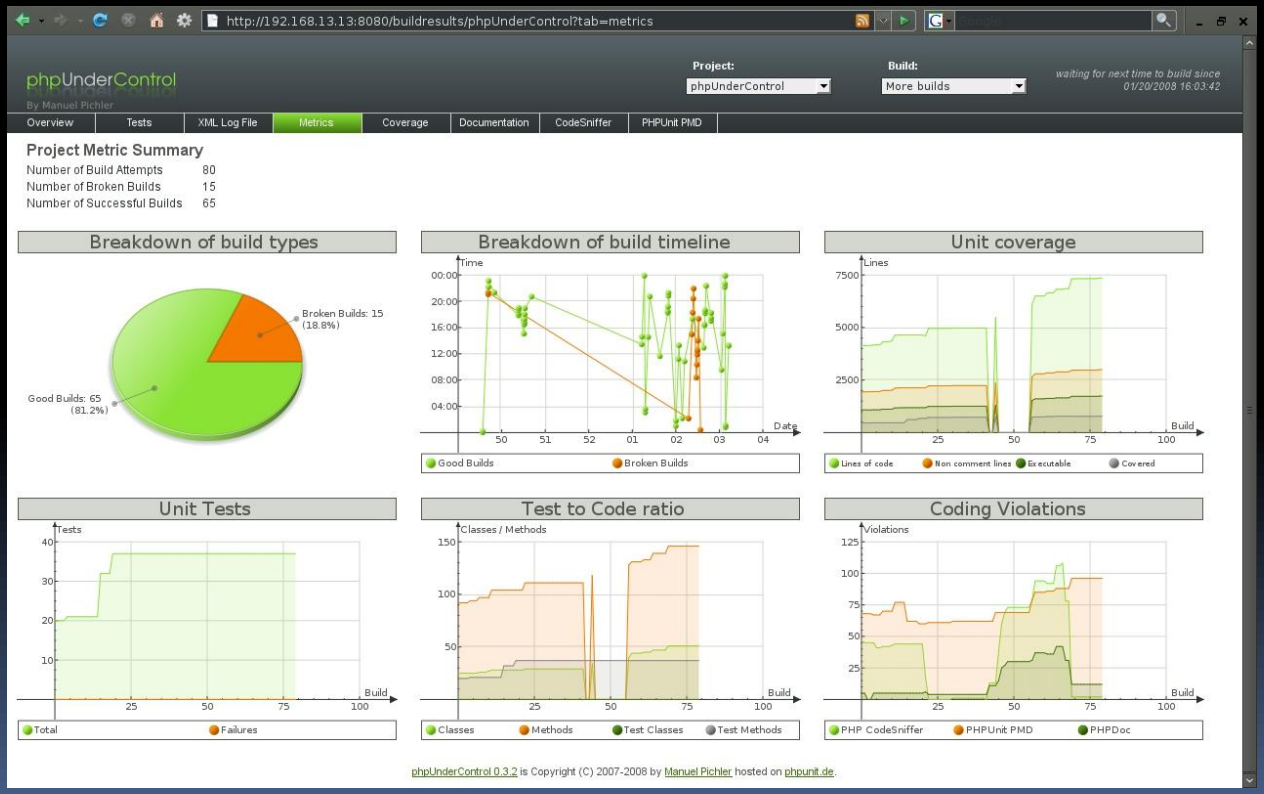
- Once all this is collected what to do with it ?
- At least, show it in a suitable form
- At best, integrate this in your CI system

phpUnderControl

- By Manuel Pichler
- CI for PHP
- Based on CruiseControl
  - Integrates natively various tools :
  - PHPUnit (+XDebug for code coverage),
  - PHP_CodeSniffer
  - PHPDocumentor
  - PMD via PHPUnit (now PHPMD)

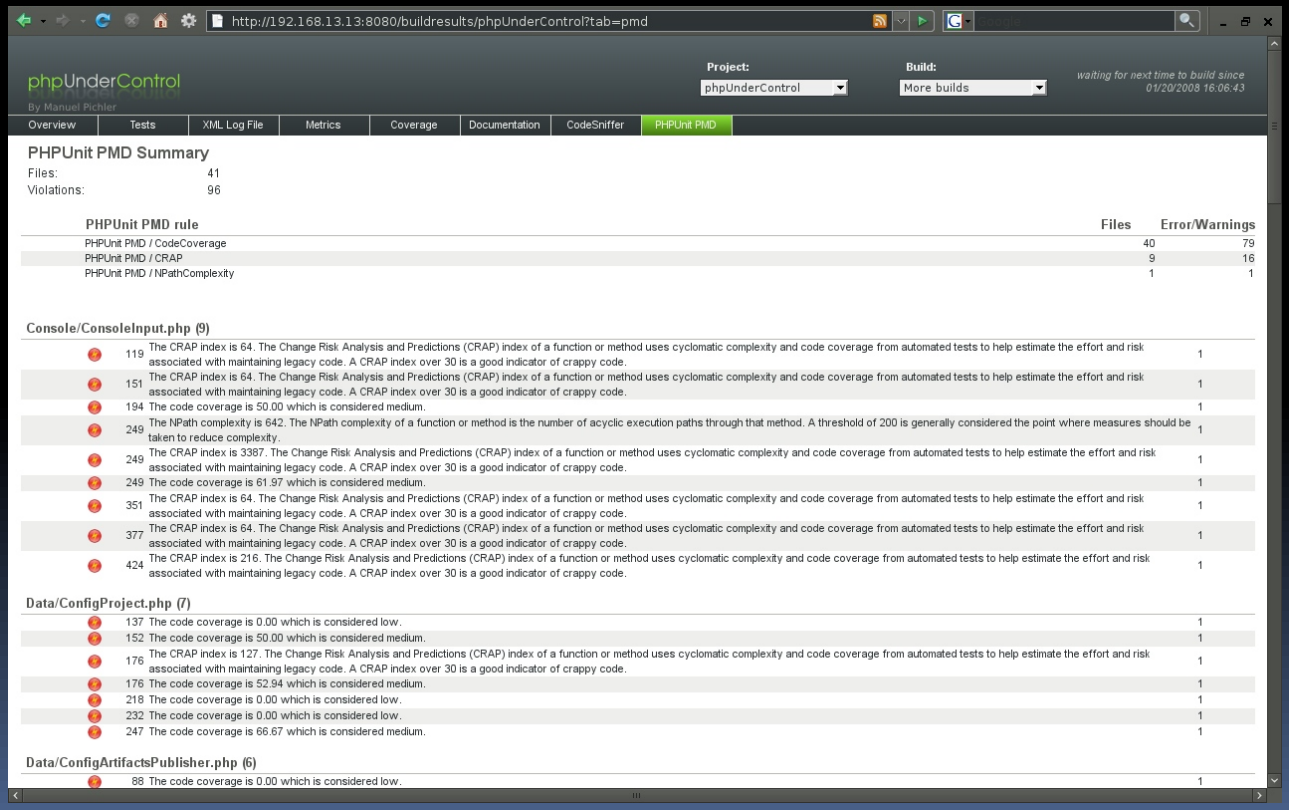
phpUnderControl

- What it gives : **metrics graphs**



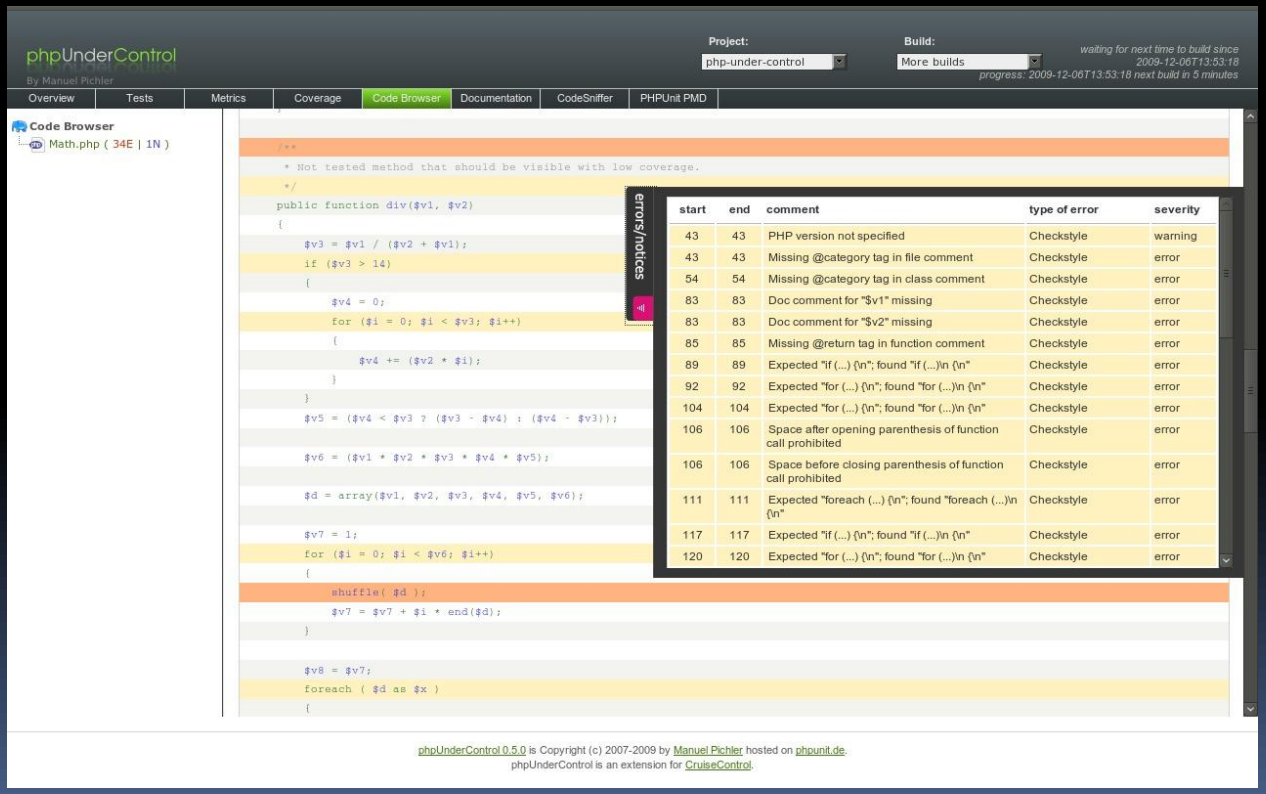
phpUnderControl

- What it gives : **report lists**



phpUnderControl

- What it gives : **PHPCodeBrowser**

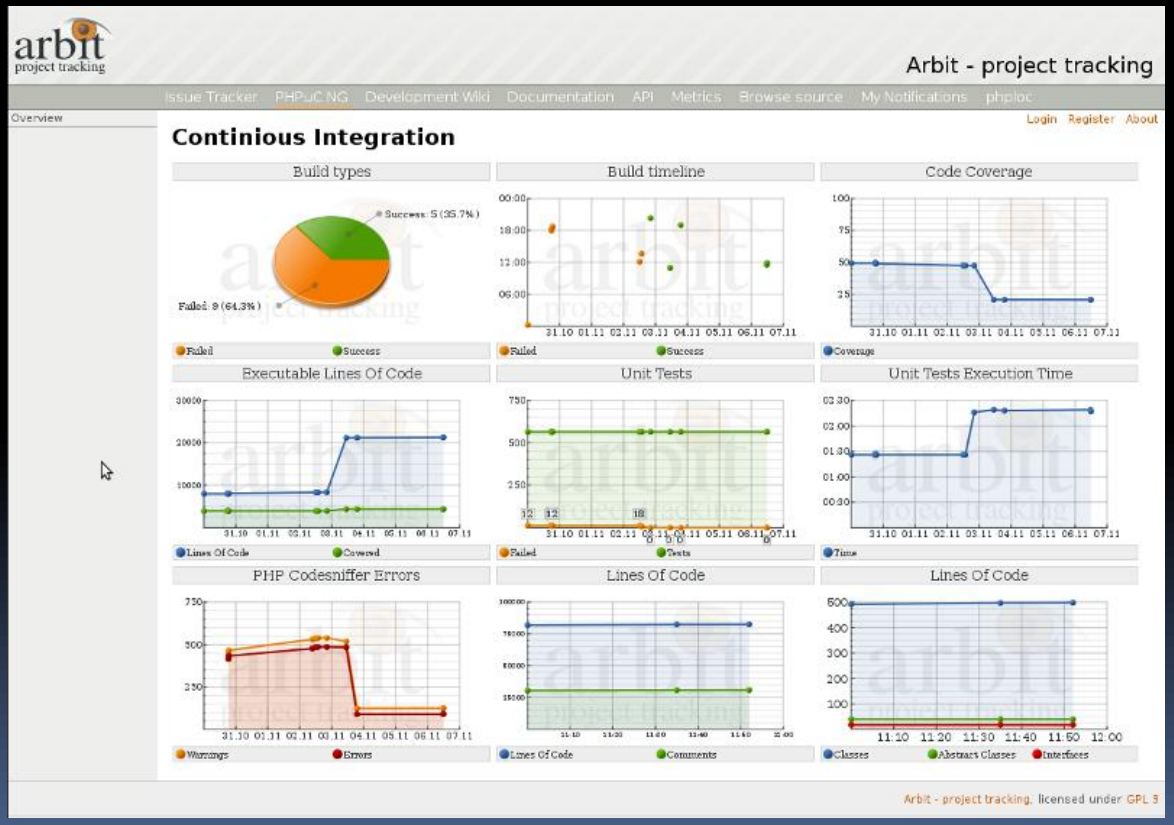


Arbit

- By Qafoo (with Manuel Pichler)
- Basically a project multi-services tool
  - Ticketing system
  - Repository browser
  - Continuous integration
- As Manuel is in it, some graphical presentations are unique for this tool
- Still alpha

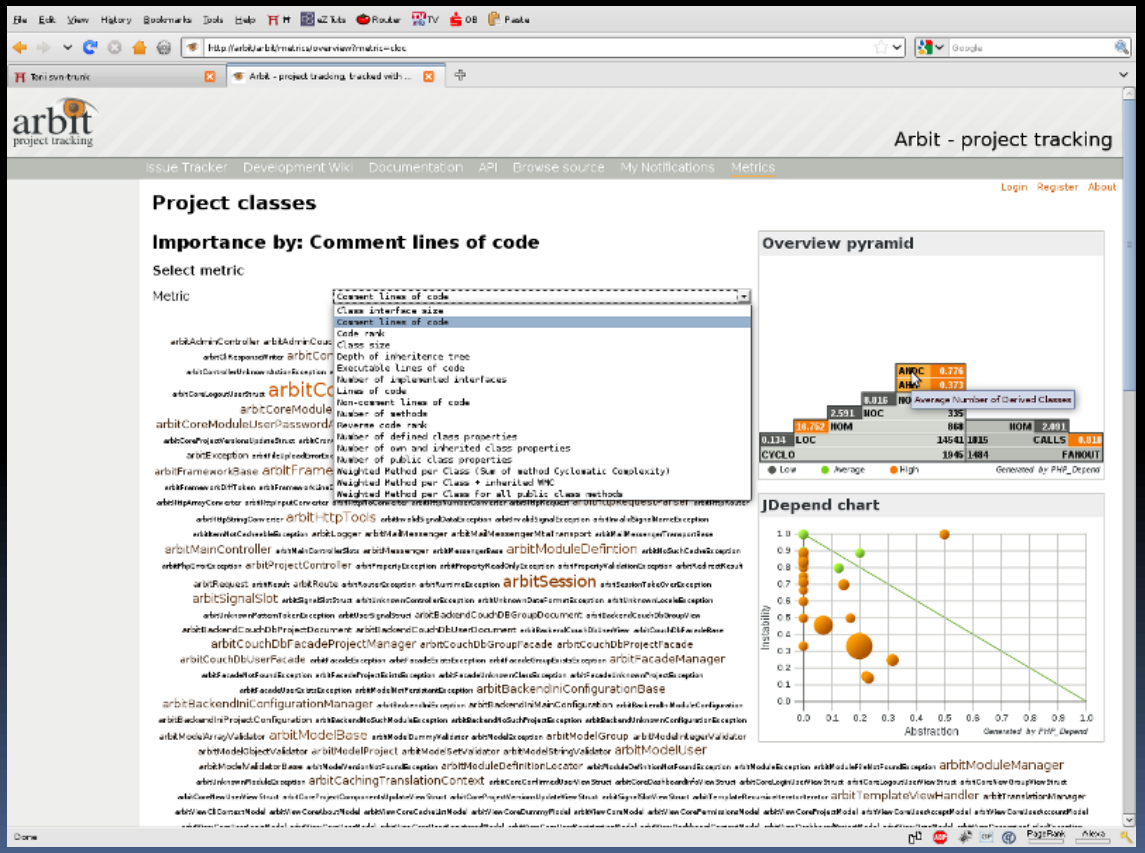
Arbit

- What it gives : more **metrics graphs**



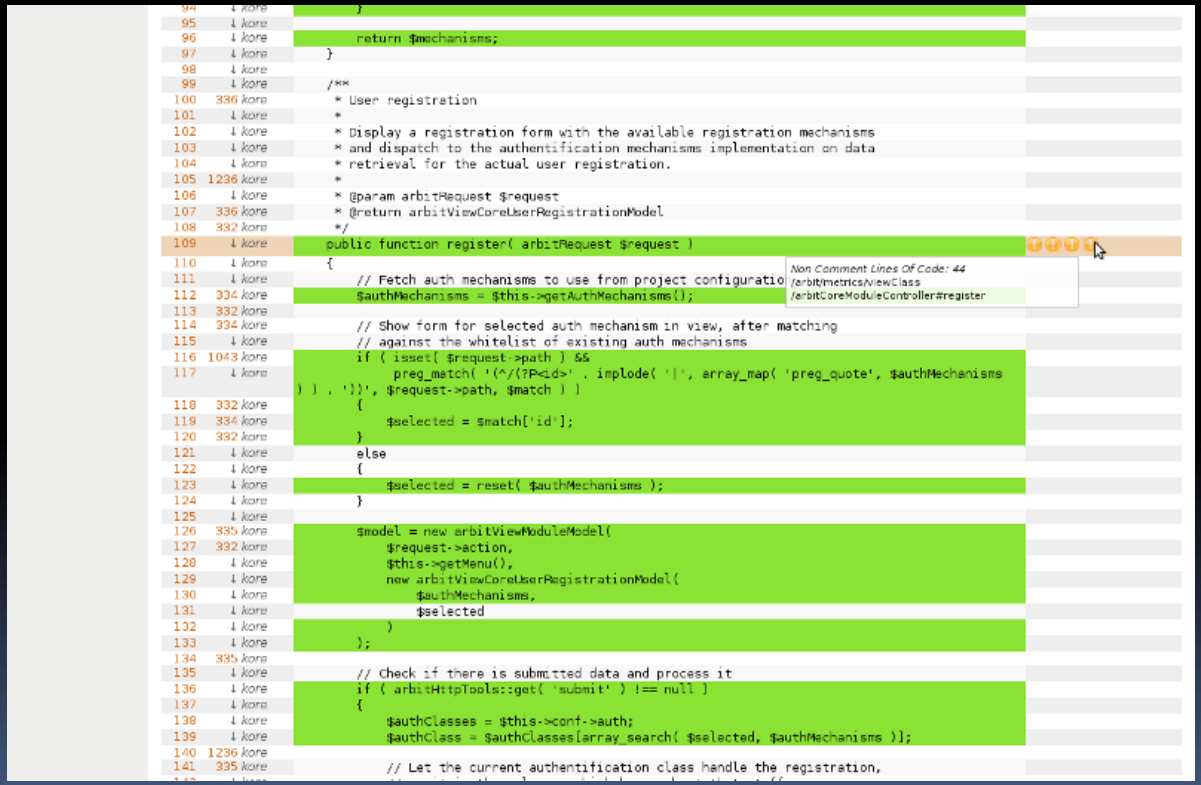
Arbit

- What it gives : **PHP_Depend overview**



Arbit

- What it gives : **Annotated sources**



Plugins Sonar for PHP

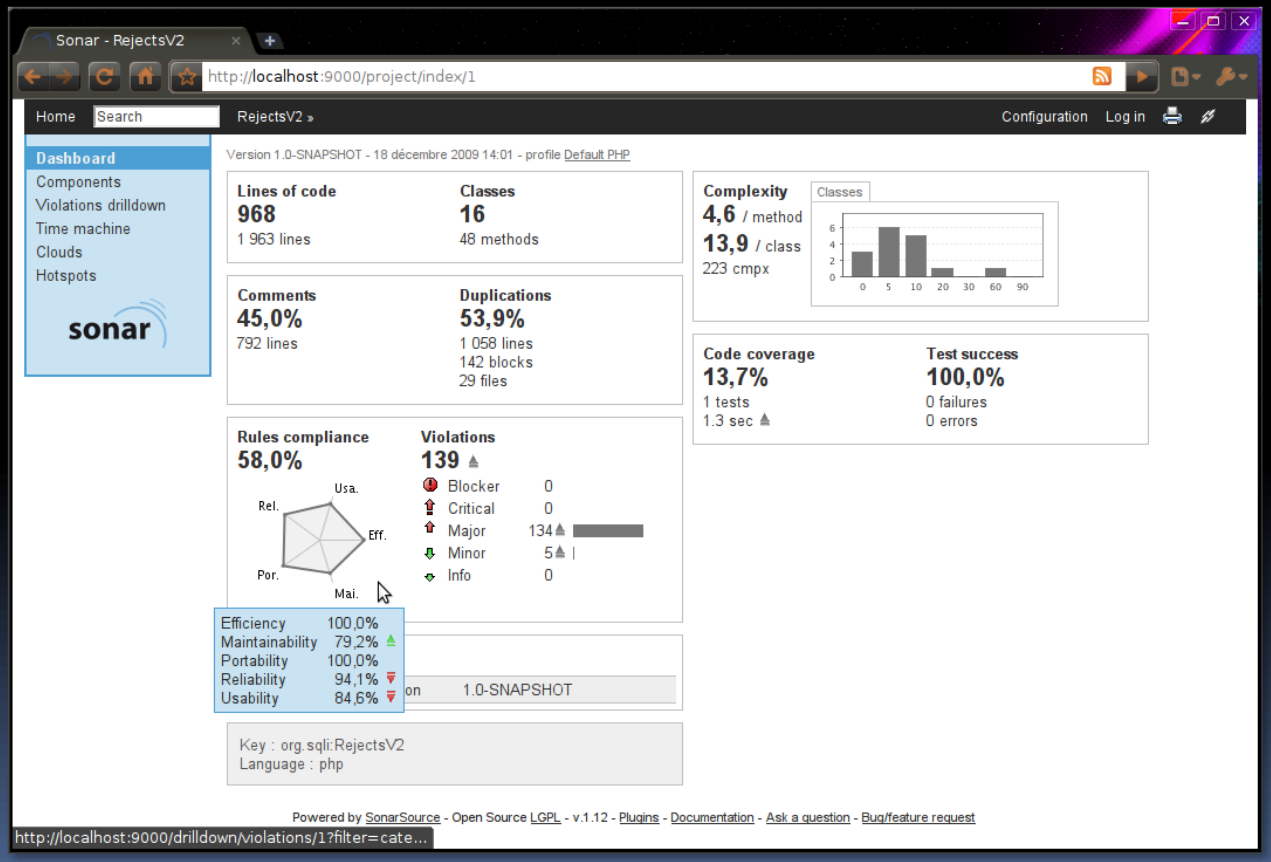
- By me ☺
  - Really by the Java guys at SQLI
  - Frédéric Leroy, Akram Ben Aissi, Jérôme Tama
- Sonar is the state of the art for Open Source QA Reporting in Java
  - Thought for multilanguage
- Can easely integrate all PHP reportings ported from Java tools
  - Junit => PHPUnit
  - JDepend => PHPDepend
  - Java PMD => PHPMD

Plugins Sonar for PHP

- Ok, not always so *easely*
  - CheckStyle is not PHP_CodeSniffer
  - Formats are not identical
  - Multi-language doesn't mean no work to add one
- First release on May 2010
  - 0.2 Alpha state, but workable
  - Easy to install : give it a try !
  - Last version demo : **sonar-php.sqli.com**
- Ok, enough, here are the screenshots

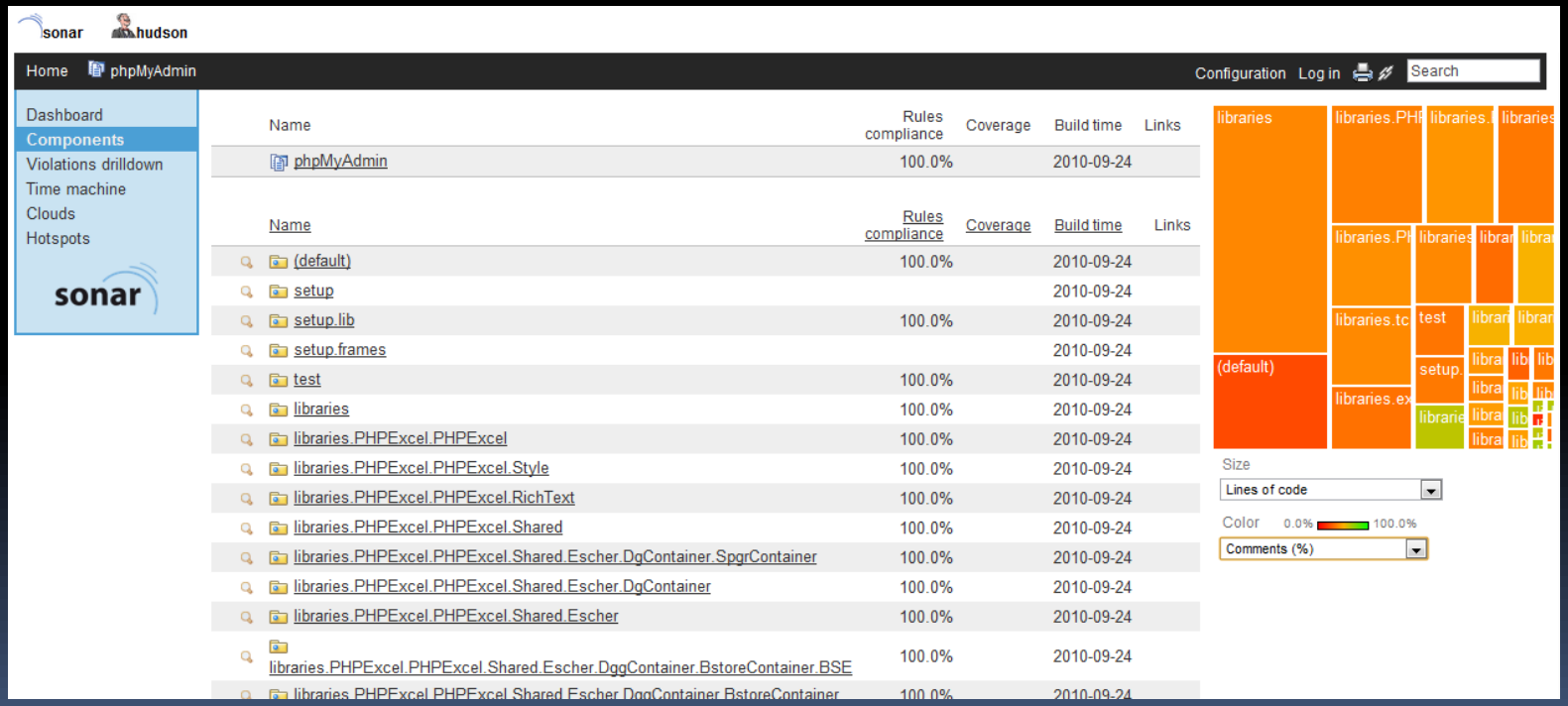
Plugins Sonar for PHP

- Dashboard



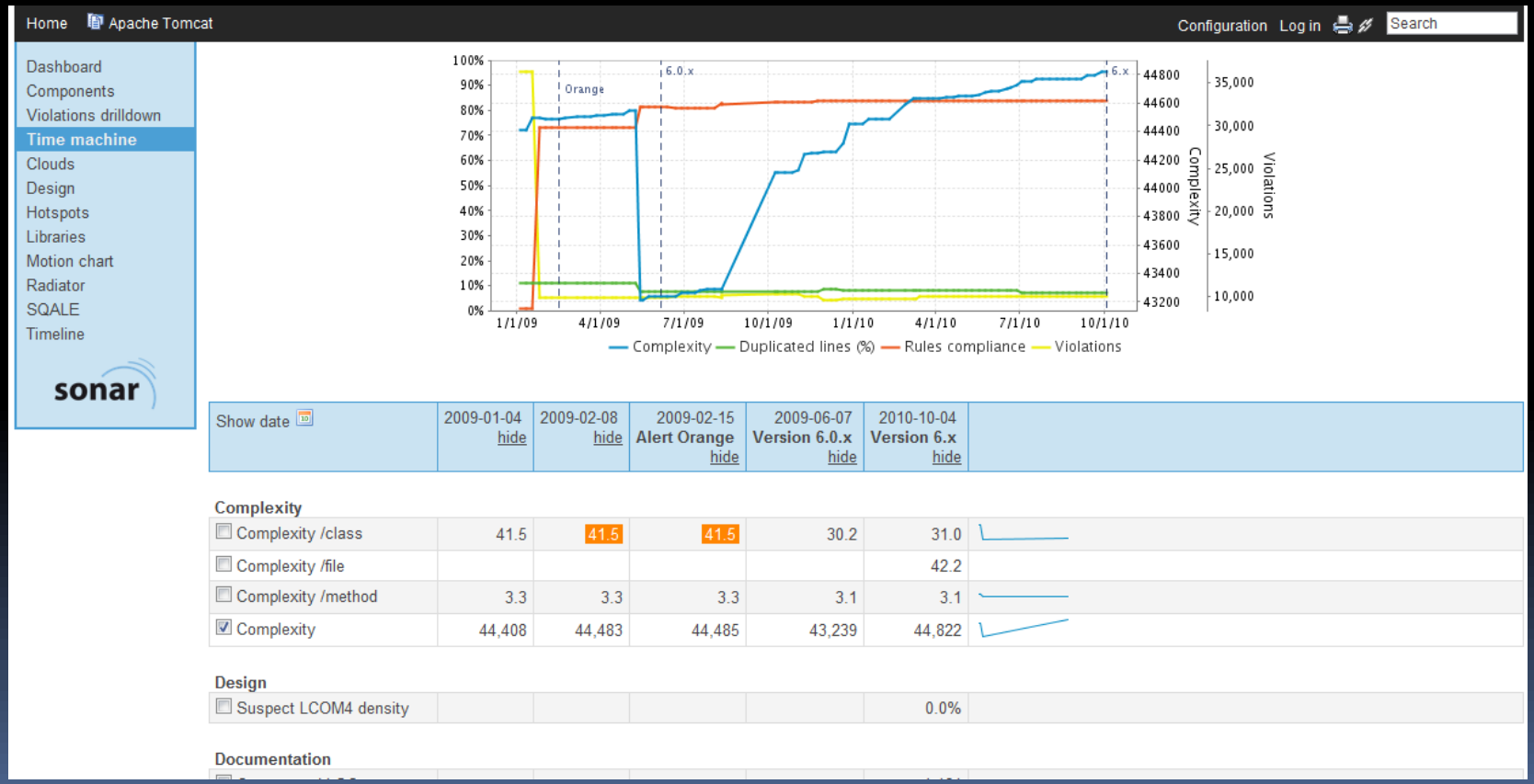
Plugins Sonar for PHP

- Components : **treemaps**



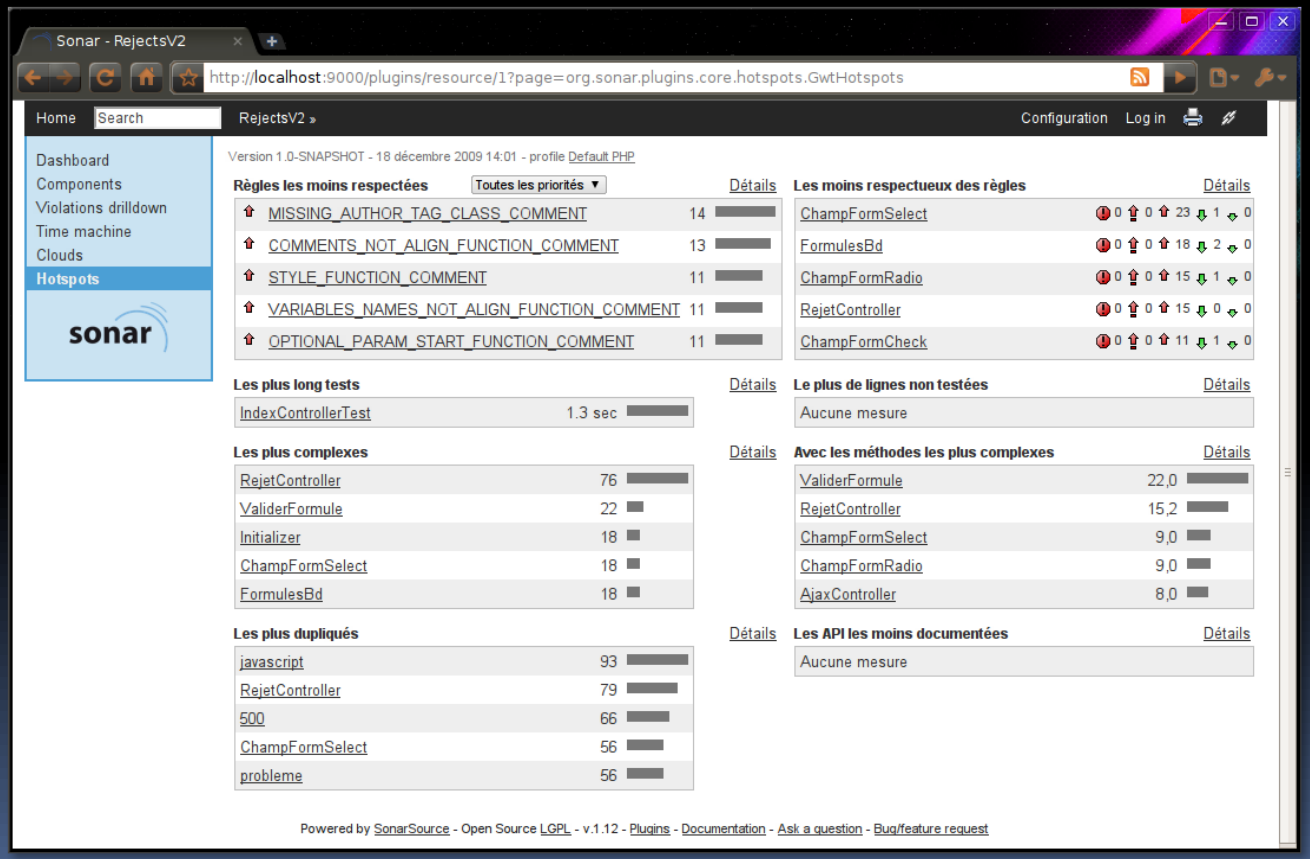
Plugins Sonar for PHP

- Time machine



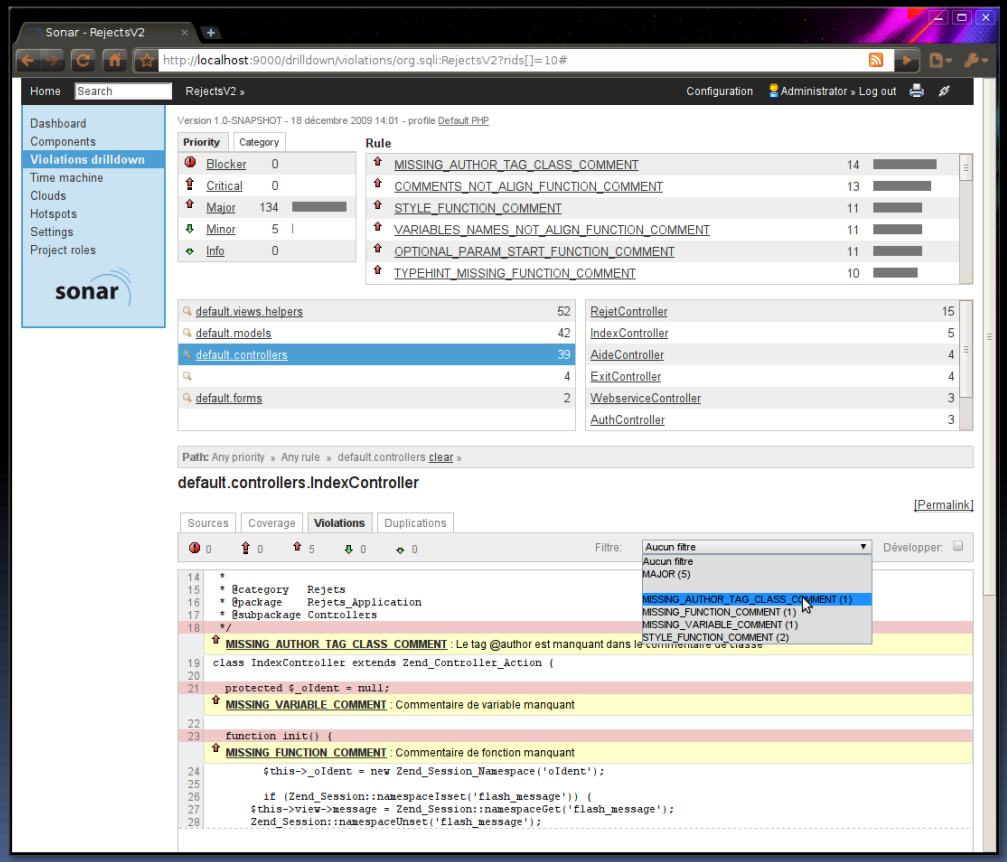
Plugins Sonar for PHP

- Hotspots



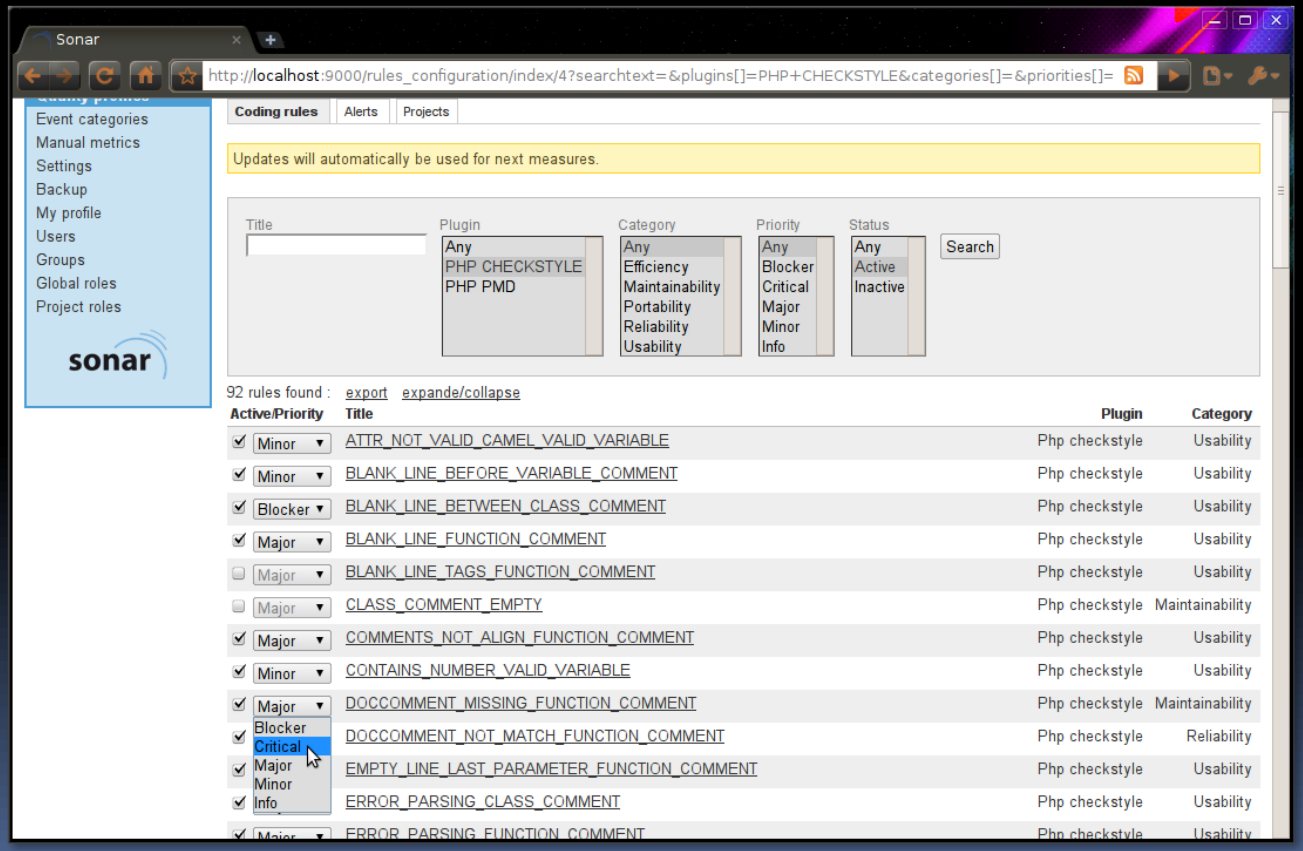
Plugins Sonar for PHP

- Violations



Plugins Sonar for PHP

- **Editing Code Profile**



Conclusion

- Sonar really goes further
  - Best integrates with Hudson
- Still is java…
  - But SonarSource really cooperates

- How to interact with phpUnderControl, Arbit ?
  - (actually our solution – PIC PHP SQLI- is based on phpUC + Sonar)
  - This needs to evolve