AnalyzeD – Static Code Analysis for D

AnalyzeD

because code quality is not optional

Stefan Rohe
3th May 2013
DConf

Overview

Introduction / Motivation

Introduction

- Funkwerk Information Technologies
  - HQ in Munich, Germany
  - One of the market leaders in passenger information systems for public transportion

- http://www.funkwerk-itk.com

Motivation - Sun Microsystems rationale

- Code conventions are important to programmers for a number of reasons:
  - 40%-80% of the lifetime cost of a piece of software goes to maintenance.
  - Hardly any software is maintained for its whole life by the original author.
  - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.
- http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Motivation

- Started with D (D1, Tango) in mid of 2008.
- 05/2013 having 250kLOC D (src); 8 developers
- Writing Clean Code → Many Code Reviews
- Time waste and social difficulty to mark Code that violates (computer checkable) conventions
- Even more difficult if conventions and their reason haven't been written down anywhere
- Growing team; needed a way to effectively split knowledge about the conventions
- Got Java/C++ Programmers
- No courses or training available

Motivation

„You cannot always hire perfect engineers.“
Walter Bright, DConf 2013

Motivation - Teams

- Teams need to have a common understanding of code
- Need growth with:
  - Size of team
  - Distance between team members
  - Cultural differences
  - …

- C++ Coding Conventions → Effective C++, C++
- Java Coding Conventions → Oracle
- D Coding Conventions → Effective D (anyone?)

- C as well as C++ also define a safe subset
  (e.g.: MISRA C / C++)

Motivation – C/C++, Java, D

- Importance for Conventions grows as languages gets more complex

- D Multiparadigm Language
- not many clean finished projects to copy from
- not many books
- no courses / training available
- few rules of thumb
- few best practices, ...

- D is a huge toolbox, but when to use which tool?

Our D Way

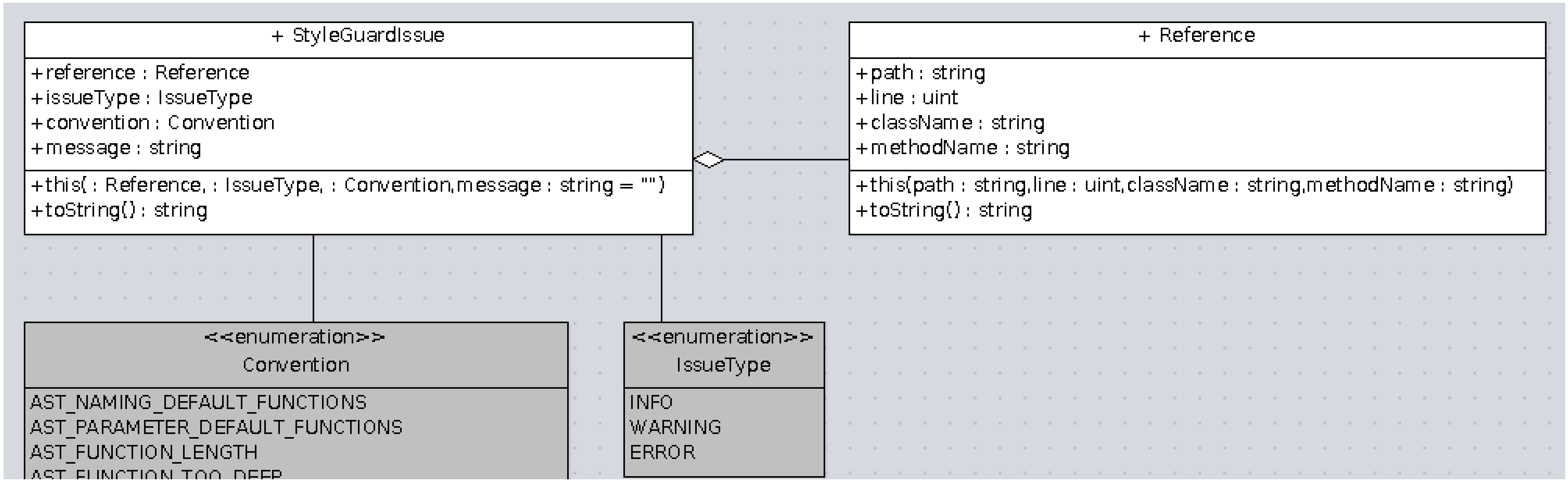
Our D Way

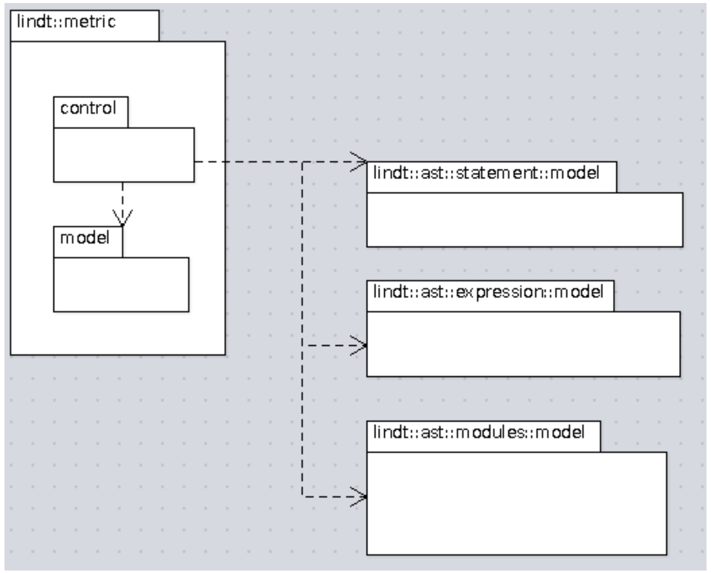
- Highly available systems that works 24/7
- Maintenance for 10+ years
- → Investment in code quality pays out fast
- Agile approach for development
- Main goal is readable, maintainable, concise code
- Asserts and all invariants are active in release version
- Contracts everywhere
- Tests:
  - Whitebox (unittest, dunit)
  - Blackbox (Python Acceptance tests)
  - Static Code Analysis

Our D Way – OO && UML

- Restricting to OO approach with one class/struct/interface per file
- A class is an item with a strict boundary. e.g.: Arrays will be dupped at these borders.
- We do not use Ddoc, because most important information about functions are not included.
  - In- / Out-Constrains
  - Exceptional Cases aka Throws
- Not implementation should be documented, but interactions should be.
- Therefore using UML Class Diagrams for each package
- Codebase gets synchronized with these diagrams through Antlr, python, stringtemplates magic
- Next to classes also model package dependency
  → no cyclic dependencies anymore

Our D Way – OO && UML




lindt::metric
control
model
lindt::ast::statement::model
lindt::ast::expression::model
lindt::ast::modules::model

Our D Way - Whitebox Tests

- Whitebox Test for Modules
- Private static functions which could be tested inplace are tested using D-unittest-Blocks (Unit =:= Function)
- Other tests require setups, teardowns, names (Testdox http://agiledox.sourceforge.net/). Need to be filtered, selected. For them using Dunit. (Unit =:= Class / Struct)
- http://github.com/linkrope/dunit

Our D Way - Whitebox Tests

- Unittests execute
  - in total,
  - separate and
  - in every combination of 2 (prove independence)

- Crash in Druntime for (short running) programs with multiple threads

Static Code Analysis

Static Code Analysis - AnalyzeD

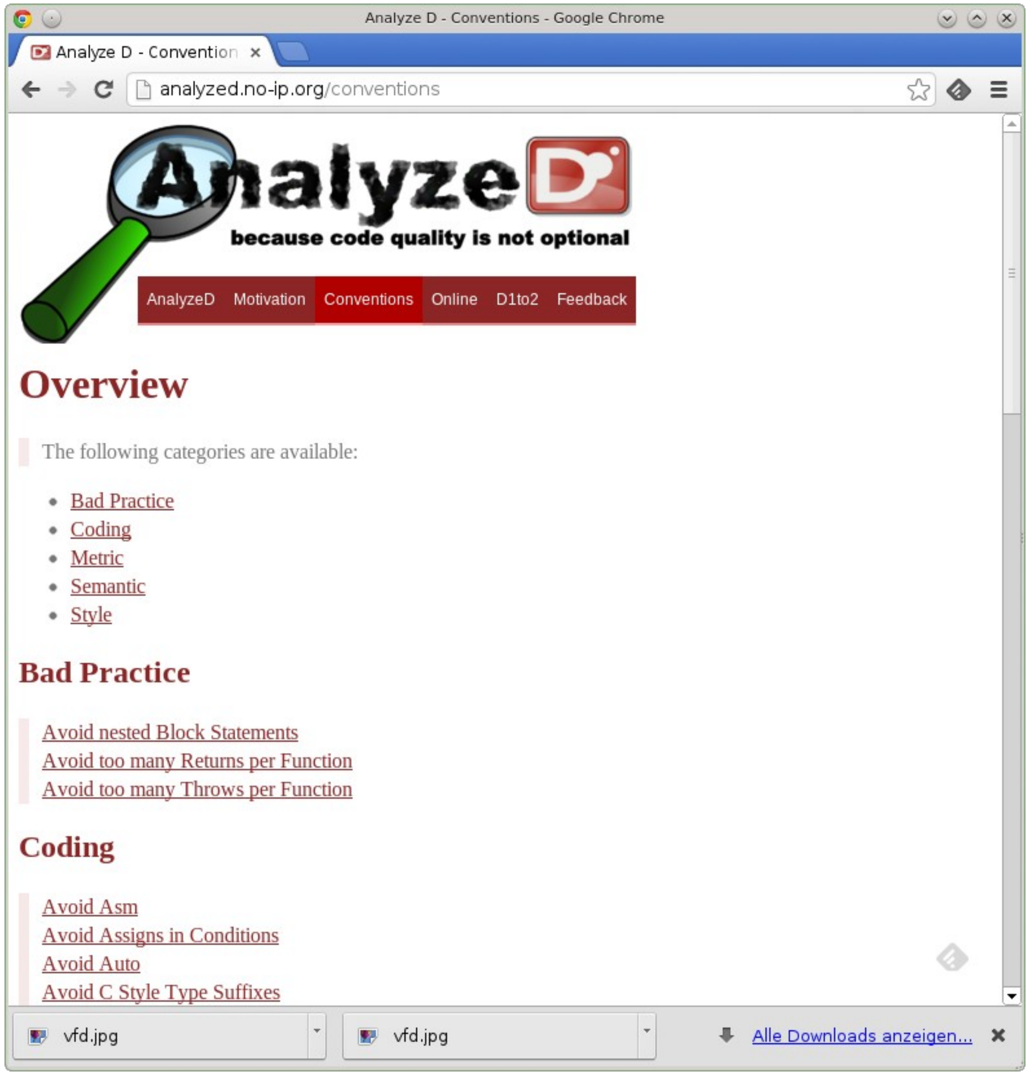
- Supporting the currently discovered conventions

- Restricted to own D code; string mixins (version, template) not supported → Not supporting whole D grammar.

- Goals:
  - Explain each convention and mention why it could be useful to follow it
  - As less as possible false positives
  - No annotations for the source code

Static Code Analysis - Categories

- Bad Practice
- Coding
- Metric
- Semantic
- Style


Analyze D - Conventions - Google Chrome
Analyze D - Convention
analyzed.no-ip.org/conventions
AnalyzeD
because code quality is not optional
AnalyzeD
Motivation
Conventions
Online
D1to2
Feedback
vfd.jpg
vfd.jpg
Alle Downloads anzeigen...

Overview

The following categories are available:

- Bad Practice
- Coding
- Metric
- Semantic
- Style

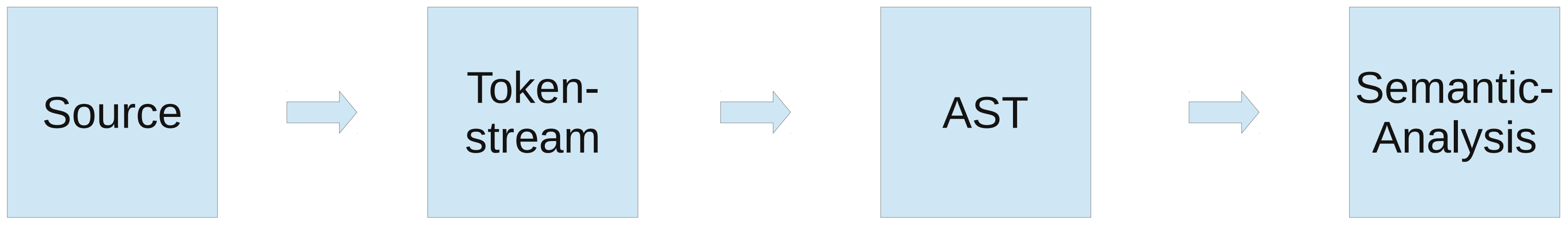
Bad Practice

Avoid nested Block Statements
Avoid too many Returns per Function
Avoid too many Throws per Function

Coding

Avoid Asm
Avoid Assigns in Conditions
Avoid Auto
Avoid C Style Type Suffixes

Static Code Analysis – Levels of Analysis



e.g.: overfull lines, ...

e.g.: outcommented code, ...

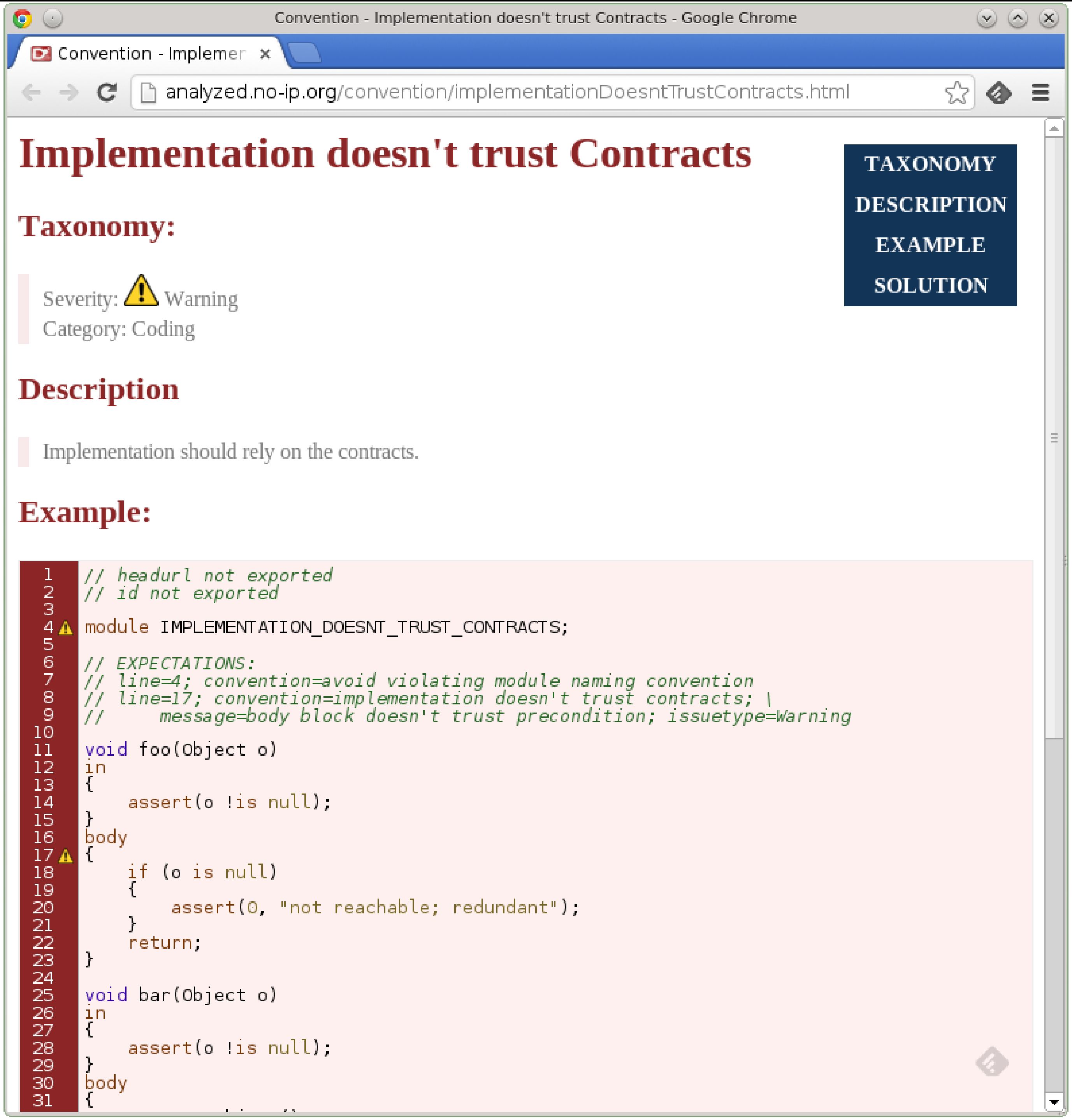
e.g.: avoid for, pointer, goto, public imports, ...

e.g.: unused code, unused declarations, ...

- Different convention checkers use different input
- Do not completly need to parse code, also could analyze invalid files with a subset of the rules

Static Code Analysis – Advantages in D

- Code is already well formalized.
  Contracts are built in.
- Developers should use and trust these contracts



Convention - Implementation doesn't trust Contracts - Google Chrome

analyzed.no-ip.org/convention/implementationDoesntTrustContracts.html

Implementation doesn't trust Contracts

TAXONOMY
DESCRIPTION
EXAMPLE
SOLUTION

Taxonomy:

Severity: ⚠ Warning
Category: Coding

Description

Implementation should rely on the contracts.

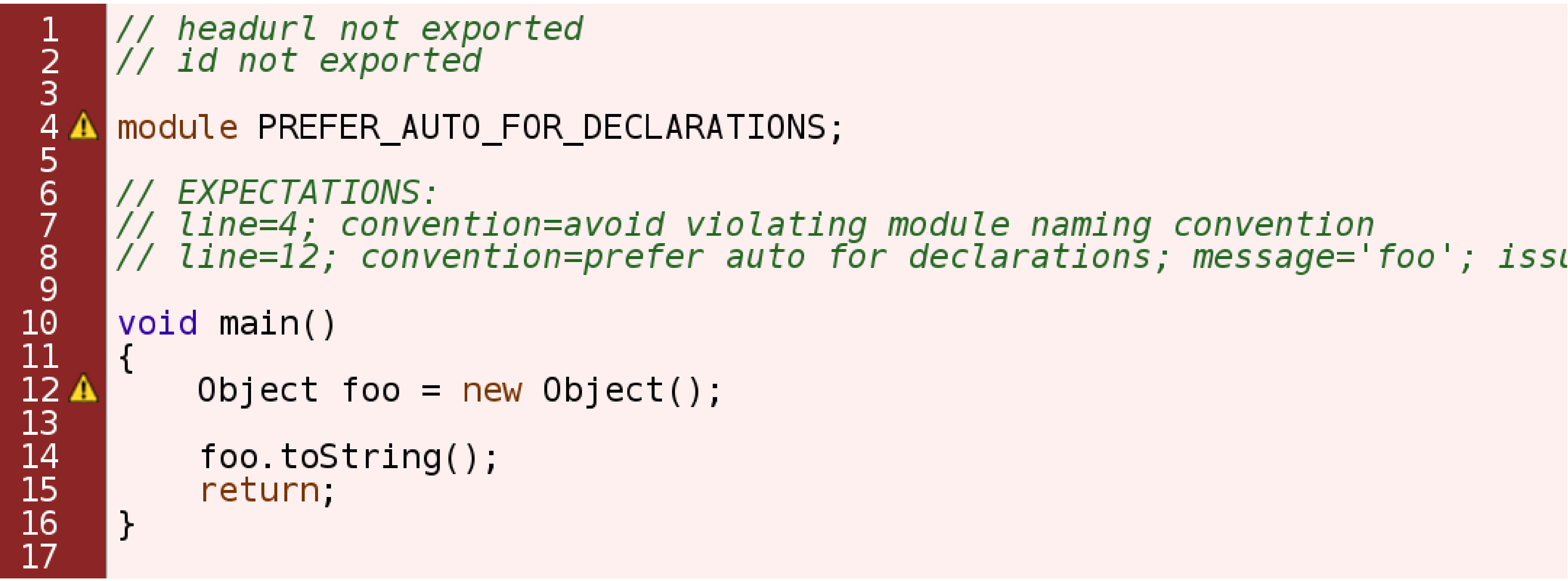
Example:

```
 1   // headurl not exported
 2   // id not exported
 3
 4 ⚠ module IMPLEMENTATION_DOESNT_TRUST_CONTRACTS;
 5
 6   // EXPECTATIONS:
 7   // line=4; convention=avoid violating module naming convention
 8   // line=17; convention=implementation doesn't trust contracts; \
 9   //     message=body block doesn't trust precondition; issuetype=Warning
10
11   void foo(Object o)
12   in
13   {
14       assert(o !is null);
15   }
16   body
17 ⚠ {
18       if (o is null)
19       {
20           assert(0, "not reachable; redundant");
21       }
22       return;
23   }
24
25   void bar(Object o)
26   in
27   {
28       assert(o !is null);
29   }
30   body
31   {
```

Static Code Analysis – Examples – Avoid For

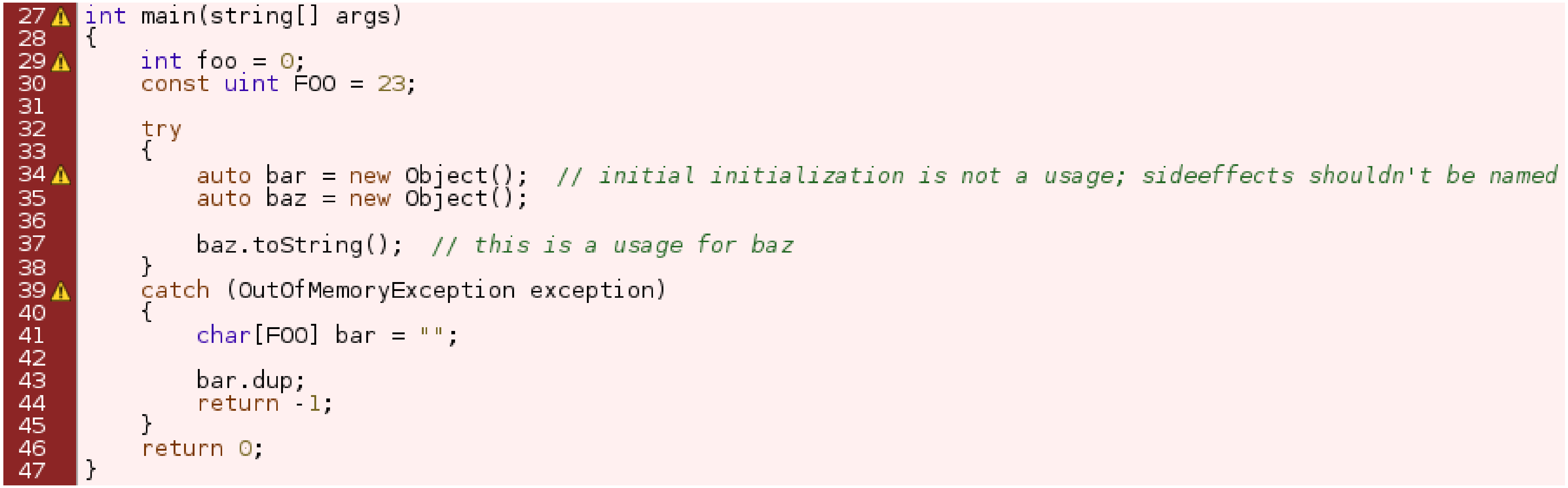
```
 1   // headurl not exported
 2   // id not exported
 3
 4 ⚠ module AVOID_FOR;
 5
 6   // EXPECTATIONS:
 7   // line=4; convention=avoid violating module naming convention
 8   // line=12; convention=avoid for; issuetype=Warning
 9
10   void main()
11   {
12 ⚠     for (int i = 0; i < 5; i++)
13       {
14           // for loops are not needed when there are ranges
15       }
16
17       // better
18       foreach (i; 0 .. 5)
19       {
20           // usage of ranges is shorter and less errorprone
21       }
22   }
23
```

Static Code Analysis – Examples – Prefer Auto



```
 1   // headurl not exported
 2   // id not exported
 3
 4 ⚠ module PREFER_AUTO_FOR_DECLARATIONS;
 5
 6   // EXPECTATIONS:
 7   // line=4; convention=avoid violating module naming convention
 8   // line=12; convention=prefer auto for declarations; message='foo'; issu
 9
10   void main()
11   {
12 ⚠     Object foo = new Object();
13
14       foo.toString();
15       return;
16   }
17
```

Static Code Analysis – Examples – Unused Variables

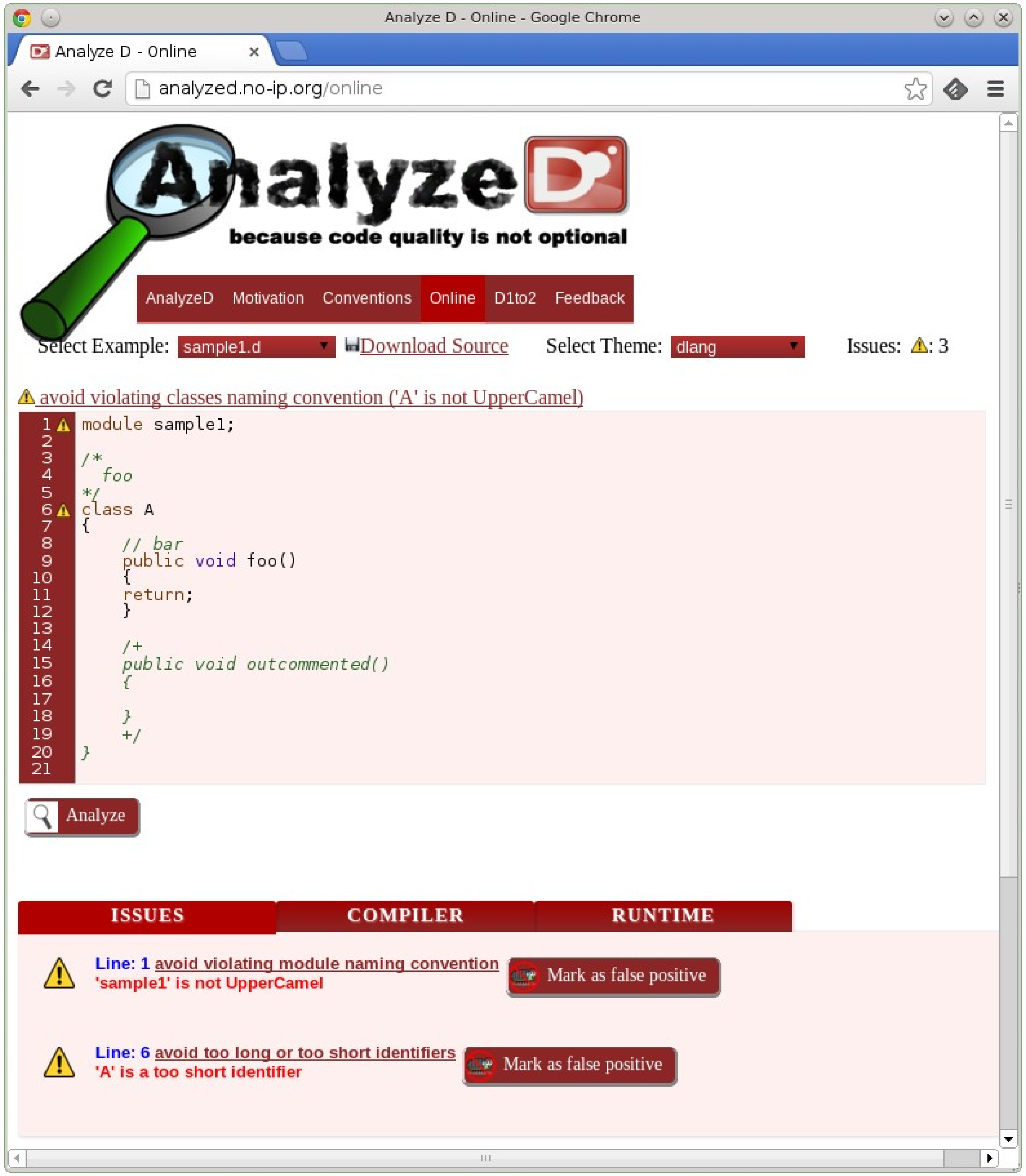


```
int main(string[] args)
{
    int foo = 0;
    const uint FOO = 23;

    try
    {
        auto bar = new Object();  // initial initialization is not a usage; sideeffects shouldn't be named
        auto baz = new Object();

        baz.toString();  // this is a usage for baz
    }
    catch (OutOfMemoryException exception)
    {
        char[FOO] bar = "";

        bar.dup;
        return -1;
    }
    return 0;
}
```

Static Code Analysis – Examples – Code Within Comments

```
 1   // headurl not exported
 2   // id not exported
 3
 4 ⚠ module AVOID_CODE_WITHIN_COMMENTS;
 5
 6   // EXPECTATIONS:
 7   // line=4; convention=avoid violating module naming convention
 8   // line=12; convention=avoid code within comments; message='// int foo
 9
10   int main()
11   {
12 ⚠     // int foo = 132;
13       return 0;
14   }
15
```

Static Code Analysis – http://analyzeD.no-ip.org/online


Analyze D - Online - Google Chrome
analyzed.no-ip.org/online
AnalyzeD
because code quality is not optional
AnalyzeD Motivation Conventions Online D1to2 Feedback
Select Example: sample1.d
Download Source
Select Theme: dlang
Analyze
ISSUES COMPILER RUNTIME
Mark as false positive

Issues: ⚠: 3

⚠ avoid violating classes naming convention ('A' is not UpperCamel)

```
module sample1;

/*
  foo
*/
class A
{
    // bar
    public void foo()
    {
    return;
    }

    /+
    public void outcommented()
    {

    }
    +/
}
```

Line: 1 avoid violating module naming convention
'sample1' is not UpperCamel

Line: 6 avoid too long or too short identifiers
'A' is a too short identifier

Metrics

Metrics

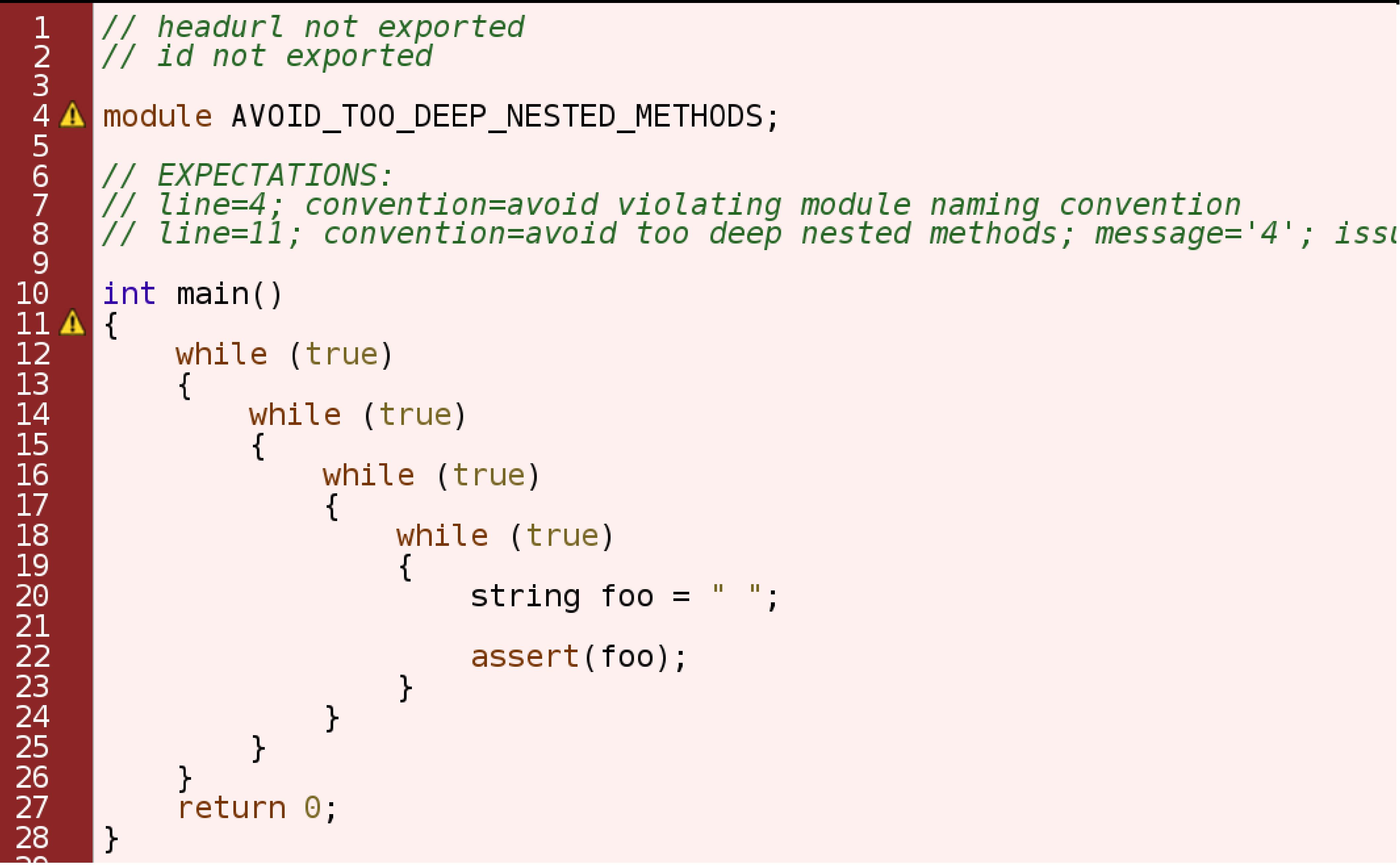
- growing code base
- need to find hotspots where review should be done and where bug clusters could hide

Methods:
- statements
- interface (parameters, throws)
- lines
- high cyclomatic complexity

Classes/Interfaces/Structs:
- Attributes / functions
- Constructors
- Lines

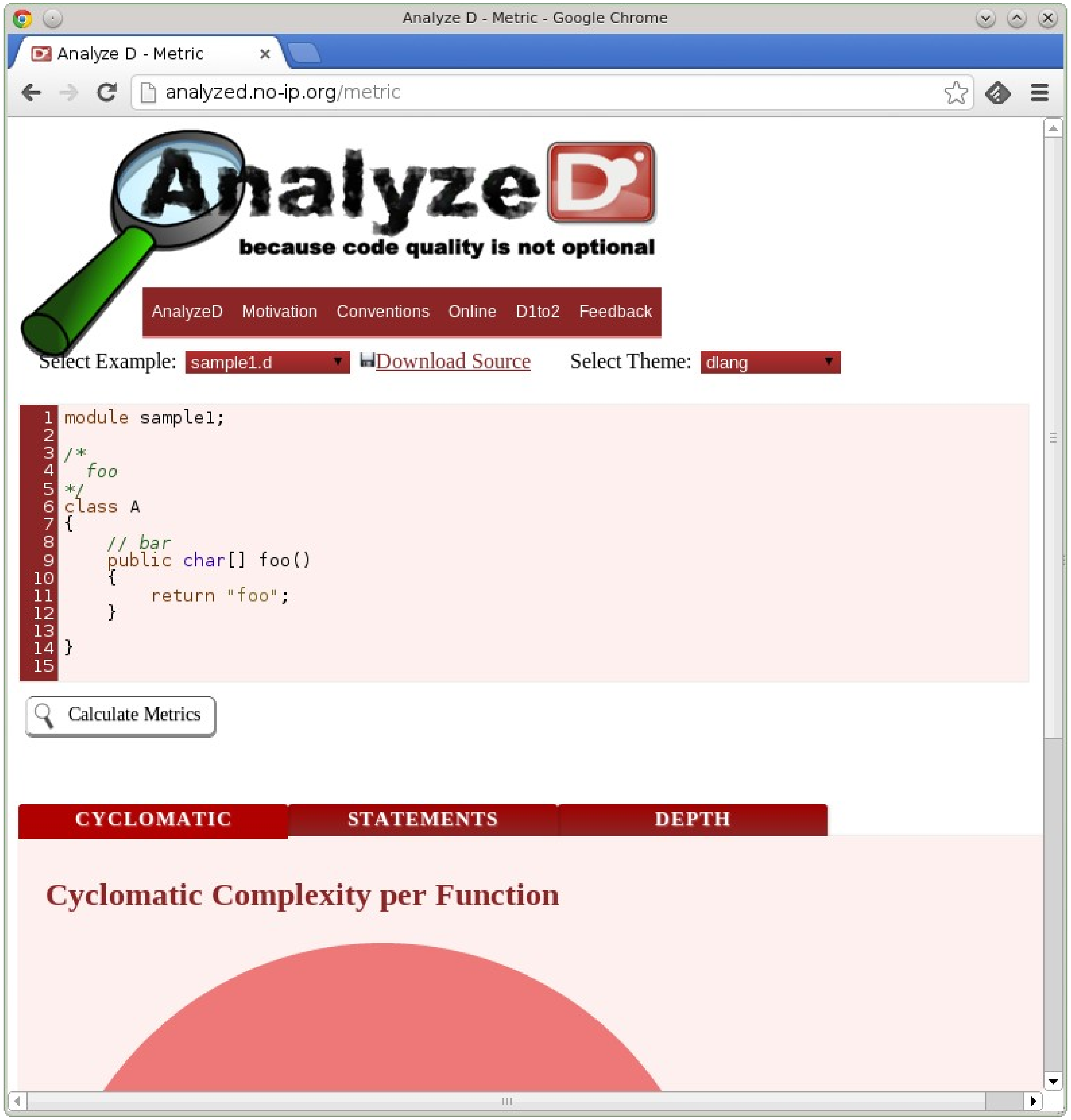
Similarity with other Tokensubsets / Duplicated Code

Metrics – High Hard Limits for Metric Violations



```
 1   // headurl not exported
 2   // id not exported
 3
 4 ⚠ module AVOID_TOO_DEEP_NESTED_METHODS;
 5
 6   // EXPECTATIONS:
 7   // line=4; convention=avoid violating module naming convention
 8   // line=11; convention=avoid too deep nested methods; message='4'; issu
 9
10   int main()
11 ⚠ {
12       while (true)
13       {
14           while (true)
15           {
16               while (true)
17               {
18                   while (true)
19                   {
20                       string foo = " ";
21
22                       assert(foo);
23                   }
24               }
25           }
26       }
27       return 0;
28   }
```

Metrics – http://analyzeD.no-ip.org/metric



Select Example: sample1.d

Select Theme: dlang

```
module sample1;

/*
  foo
*/
class A
{
    // bar
    public char[] foo()
    {
        return "foo";
    }

}

```

Cyclomatic Complexity per Function

D1to2

D1to2 - Usecase

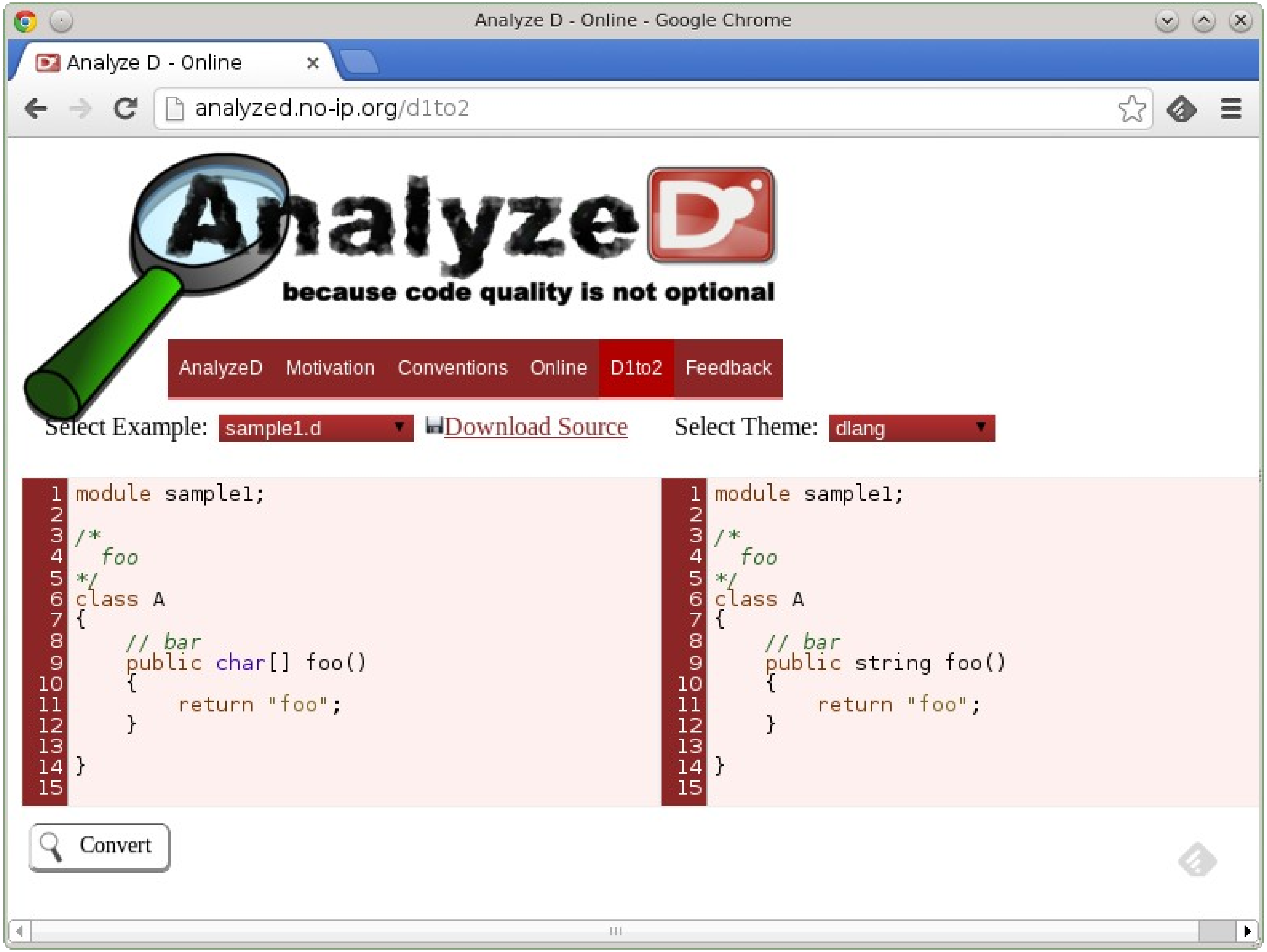
- D1 Tango Migration to D2 Phobos
- Highest migration effort in unittests, so mainly concentrated to automize that
- Goal was to automize 80%, give the interesting 20% to the developer

- Example generic rules:
  - Replace char[] by string
  - Replace Foo!(Bar)(baz) by Foo!Bar(baz)

- Example Tango rules:
  - Replace Tango.format(„{0}“, foo) with std.string.format(„%s“, foo)

- Step by step reducing Tango Dependencies. Currently Tango just for Xml and Logging.

D1to2 – http://analyzeD.no-ip.org/d1to2



```d
module sample1;

/*
  foo
*/
class A
{
    // bar
    public char[] foo()
    {
        return "foo";
    }

}
```

```d
module sample1;

/*
  foo
*/
class A
{
    // bar
    public string foo()
    {
        return "foo";
    }

}
```

Sonar

Sonar – Easy Way to Start With

There are already several D projects out there;
Quality and Style are different;
sometimes even within the projects.

There is also the D-Style (http://dlang.org/dstyle.html) which already defines several rules.

- naming conventions
- declaration style
- one statement per line
- spaces instead of tabs; multiple of 4
- braces on a single line
- avoid overfull lines

We are going to host a sonar instance for every D project which is willing to.

Sonar – How to introduce static code analysis?

Introduction of new conventions into existing code base difficult:

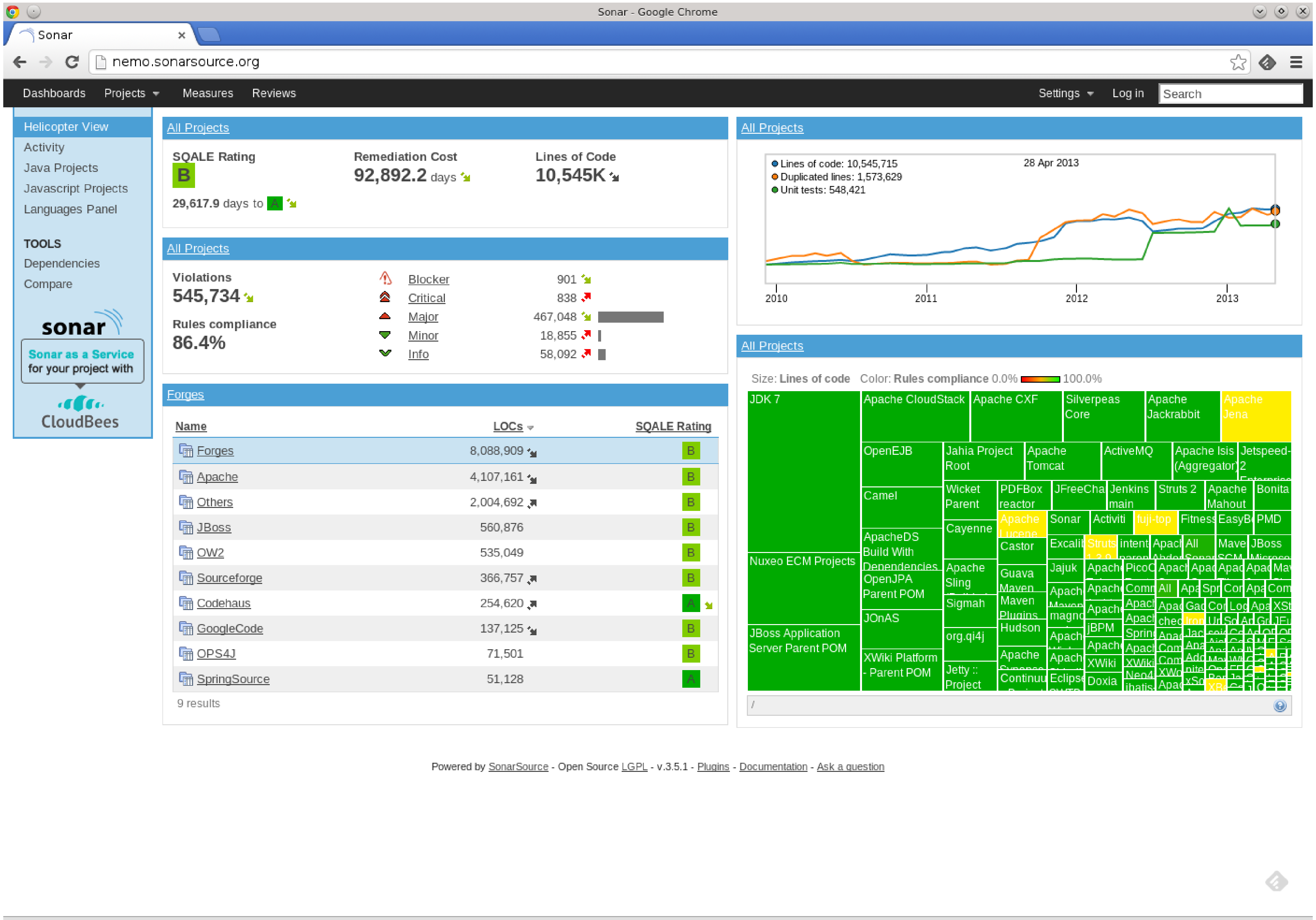
- just analyzing increments
- Analyzing whole codebase and increase number of active conventions

Sonar is an open source web based solution for quality checks on software for many languages; Continuous Quality Inspection

- Wrote a D Plugin for Sonar

  http://www.sonarsource.org/
- Sonar enables project specific settings for code analysis; which rules to follow; which particular findings are ok.

Sonar – How to introduce static code analysis?



Sonar - Google Chrome

nemo.sonarsource.org

Dashboards | Projects | Measures | Reviews | Settings | Log in | Search

Helicopter View
Activity
Java Projects
Javascript Projects
Languages Panel

TOOLS
Dependencies
Compare

sonar
Sonar as a Service for your project with CloudBees

All Projects

SQALE Rating **B**
29,617.9 days to A

Remediation Cost **92,892.2** days

Lines of Code **10,545K**

All Projects

Violations **545,734**

Rules compliance **86.4%**

Blocker	901
Critical	838
Major	467,048
Minor	18,855
Info	58,092

Forges

Name	LOCs	SQALE Rating
Forges	8,088,909	B
Apache	4,107,161	B
Others	2,004,692	B
JBoss	560,876	B
OW2	535,049	B
Sourceforge	366,757	B
Codehaus	254,620	A
GoogleCode	137,125	B
OPS4J	71,501	B
SpringSource	51,128	A

9 results

All Projects

28 Apr 2013
● Lines of code: 10,545,715
● Duplicated lines: 1,573,629
● Unit tests: 548,421

All Projects

Size: Lines of code Color: Rules compliance 0.0% 100.0%

Powered by SonarSource - Open Source LGPL - v.3.5.1 - Plugins - Documentation - Ask a question

Summary & Outlook

Summary

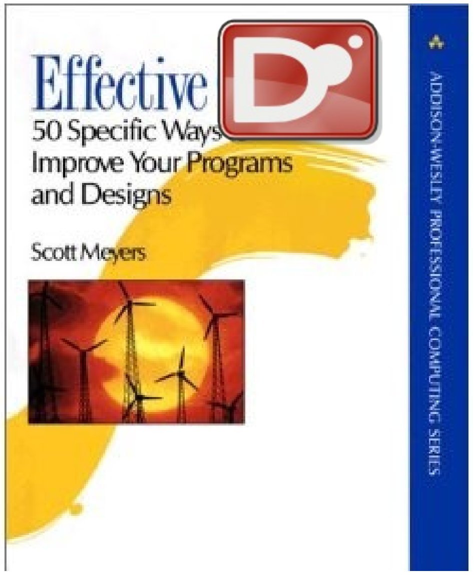
- Deployed D systems to whole over europe
- D can be used for commercial software
- Merging to D2, but still learning; D2 conventions evolve
- AnalyzeD found several „hidden“ bugs within our code base; reviews are now free of violations against AnalyzeD conventions

Outlook

- Use of available D Frontend / Antlr
  (Any Chance to get an updated D grammar published? Antlr Grammar?)

- Populate http://dlang.funkwerk-itk.com with links to our projects about D (AnalyzeD, Dunit, AntlrDruntime, Model generator, ...)

- Hopefully there will be soon an Effective D Book including best practices and rules of thumb.


Effective
50 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES