

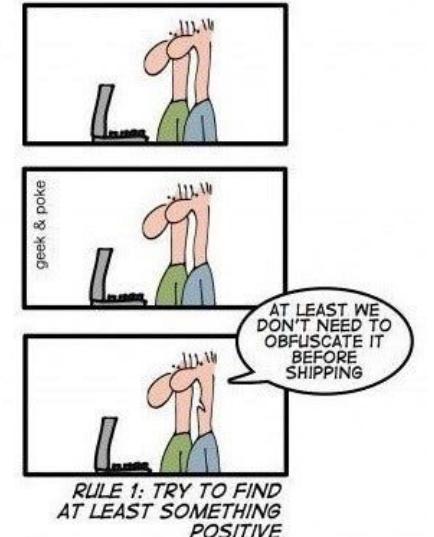
QA: Code Review & Static Analysis

17-313 Fall 2024

Foundations of Software Engineering

<https://cmu-17313q.github.io>

Eduardo Feo Flushing



Learning Goals

- Learn to get early feedback to reduce risk
- Find ways to catch our technical errors
- Gain an understanding of the relative strengths and weaknesses of static analysis
- Examine several popular analysis tools and understand their use cases
- Understand how analysis tools are used in large open source software

Administrivia

- Past Exams posted
- Cheat Sheet
 - One double-sided A4 .
 - You must submit it.
 - Handwritten = Bonus points.
 - Printed cheat sheets permitted but not awarded points.
- Midterm Next Sunday, October 6th
- Review Session: Thursday during Recitation

P2B Grading Retrospective

- Improve Git usage
 - PRs not linked to issues
 - No dependencies / tags
- Align project board with repo
- Inconsistent PR quality (some good, some bad)
- Make your contributions visible

Smoking Section

- Last **two** full rows



Risk Analysis

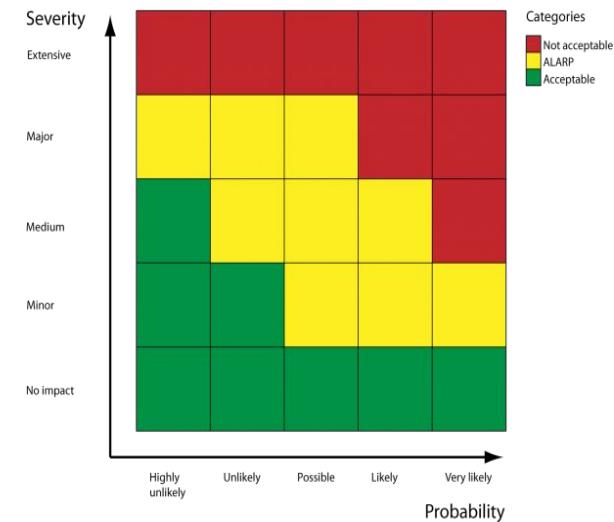
- **Probability** a human makes a mistake: **Very Likely**
- **Severity**: ranges, but could be extensive

Solution:

Use **CI** to catch your mistakes, make you look better, and mitigate your risks!

QA: Static Analysis
(today's lecture)

Use **code reviews** to teach and learn
(today's lecture)



**For problems we can't
easily automate, we can
perform code review**

Boeing Model 299 test on October 30, 1935.

- Plane crashed because of locked elevator control surface (opposite effect of MCAS)



Checklists help manage complex processes



The Checklist: <https://www.newyorker.com/magazine/2007/12/10/the-checklist>

How to create a checklist?

- Start with problems we have seen before
 - “Safety regulations are written in blood”
- Justify why this is not automatable
- Not all checklist items need to be very specific
 - An item could be “does this team know we are proposing this change”

Activity: Create a checklist for code reviews

- In pairs, think about common mistakes your “friend” made the last time they were coding.
 - Write your names on a piece of paper.
 - Write down two checklist items that would have caught those errors.
- Divide into teams: left and right sides of the classroom.
- Which team had the most unique/good entries in their list?

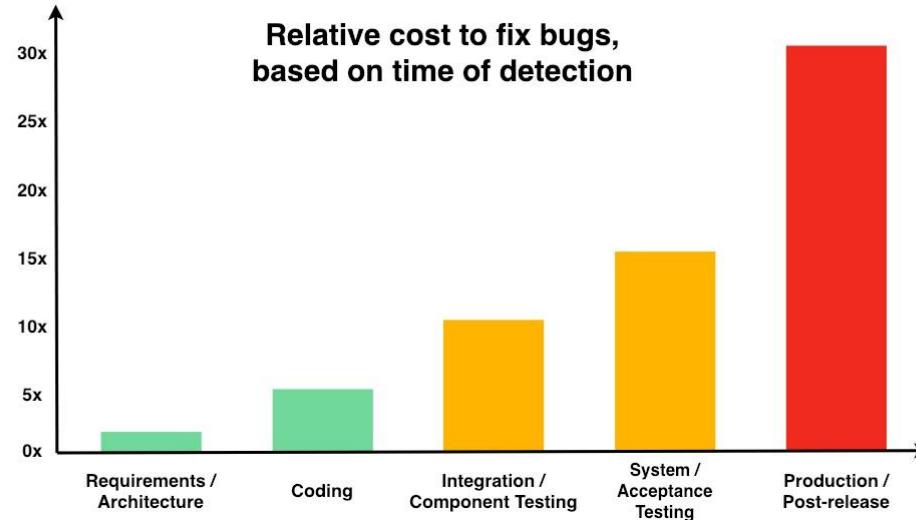
Activity: Create a checklist for code reviews

- In groups, think about common mistakes your “friend” made the last time they were coding.
 - Write your names on a piece of paper.
 - Write down two checklist items that would have caught those errors.

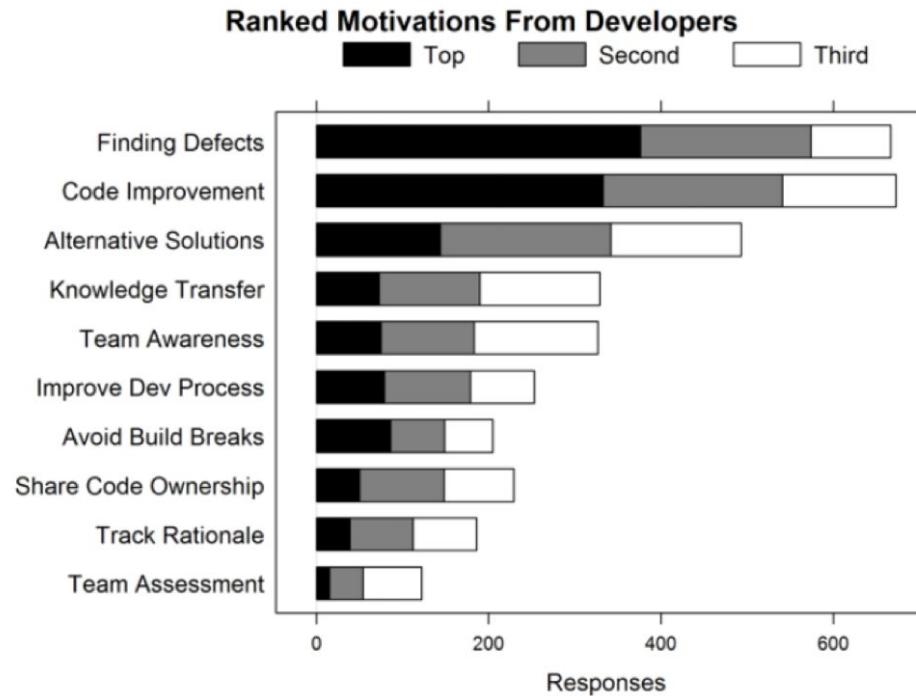
Expectations and outcomes for code review

Motivation

- Linus's Law: "Given enough eyeballs, all bugs are shallow."
 - - The Cathedral and the Bazaar, Eric Raymond

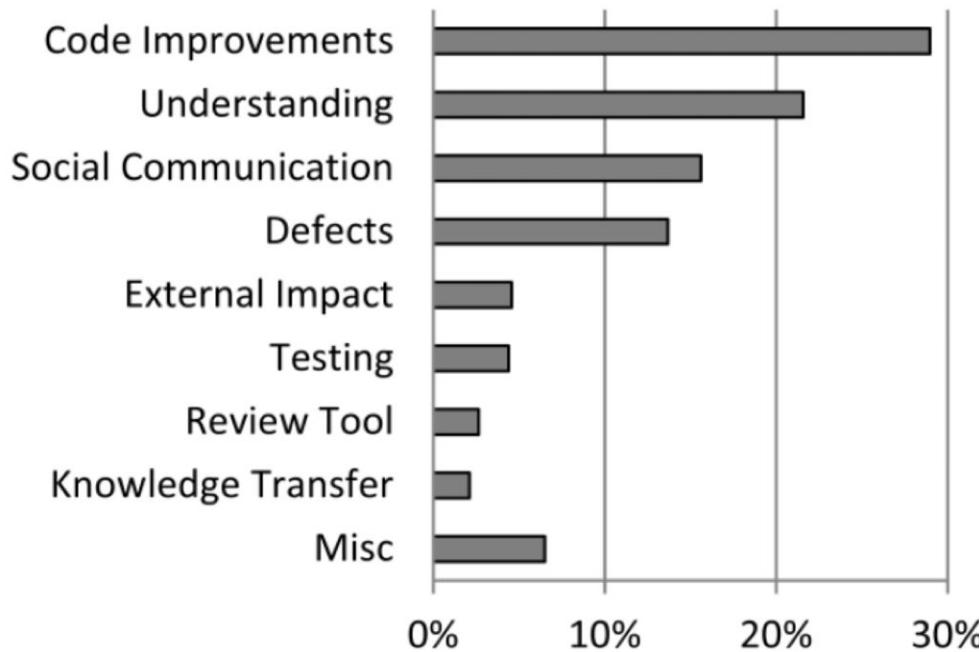


Code Review at Microsoft



Bacchelli, Alberto and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.

Outcomes (Analyzing Reviews)



Mismatch of Expectations and Outcomes

- Low quality of code reviews
 - Reviewers look for easy errors, as formatting issues
 - Miss serious errors
- Understanding is the main challenge
 - Understanding the reason for a change
 - Understanding the code and its context
 - Feedback channels to ask questions often needed
- No quality assurance on the outcome

Code Review at Google

- Introduced to “force developers to write code that other developers could understand”
- Three benefits:
 - checking the consistency of style and design
 - ensuring adequate tests
 - improving security by making sure no single developer could commit arbitrary code without oversight

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. International Conference on Software Engineering

Code Review

- Start with the “big ideas”
- Automate the little things
- Focus on understanding
- Remember a person wrote the code
- Don’t overwhelm the person with feedback

Don't forget that coders are people with feelings

- A coder's self-worth is in their artifacts
- CI can avoid embarrassment
- Identify defects, not alternatives; do not criticize coder
 - “*you* didn't initialize variable a” -> “I don't see where variable a is initialized”
- Avoid defending code; avoid discussions of solutions/alternatives
- Reviewers should not “show off” that they are better/smarter
- Avoid style discussions if there are no guidelines
- The coder gets to decide how to resolve fault

Outline

- **goto fail;** and similar unfamous bugs
- Static analysis tools
 - Linters for maintainability
 - Pattern-based static analyzers
- Challenges of static analysis

Outline

- **goto fail;** and similar unfamous bugs
- Static analysis tools
 - Linters for maintainability
 - Pattern-based static analyzers
- Challenges of static analysis

```
1. static OSStatus
2. SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
3.                                     SSLBuffer signedParams,
4.                                     uint8_t *signature,
5.                                     UInt16 signatureLen) {
6.     OSStatus err;
7.     ....
8.     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9.         goto fail;
10.    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11.        goto fail;
12.        goto fail;
13.    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14.        goto fail;
15.    ...
16. fail:
17.     SSLFreeBuffer(&signedHashes);
18.     SSLFreeBuffer(&hashCtx);
19.     return err;
20. }
```

goto fail;

Analysis

Apple's SSL iPhone vulnerability: how did it happen, and what next?

Charles Arthur

SSL vulnerability in iPhone, iPad and on Mac OS X appeared in September 2012 - but cause remains mysterious as former staffer calls lack of testing 'shameful'

goto fail; // Apple SSL bug test site

This site will help you determine whether your co

YOUR BROWSER IS VULNERABLE

We have examined your OS and browser version information and determined that your system is vulnerable to a serious security flaw. This means that anyone who can intercept your network traffic (such as your ISP or other networks) can freely snoop on you, for example when you log into them right away. Other applications on your system such as mail.

Apple's SSL vulnerability is still active on Safari on Mac OS X as shown at the gotofail.com site.
Photograph: Public domain



/ business

tomorrow
belongs to those who embrace it
today

trending

tech

innovation

business

security

advice

buying guides

Home / Business / Companies / Apple

When will Apple get serious about security?

The tech community (and beyond) is an uproar over the recently revealed iOS and OS X SSL/TLS code flaw. Apple developers have questions about Apple's commitment to quality and the flaw itself.



Written by David Morgenstern, Contributor on Feb. 23, 2014

```
1. /* from Linux 2.3.99 drivers/block/raid5.c */
2. static struct buffer_head *
3. get_free_buffer(struct stripe_head * sh,
4.                  int b_size) {
5.     struct buffer_head *bh;
6.     unsigned long flags;
7.     save_flags(flags);
8.     cli(); // disables interrupts
9.     if ((bh = sh->buffer_pool) == NULL)
10.         return NULL;
11.     sh->buffer_pool = bh -> b_next;
12.     bh->b_size = b_size;
13.     restore_flags(flags); // re-enables interrupts
14.     return bh;
15. }
```

ERROR: function returns with
interrupts disabled!

December 2014

S	M	T	W	T	F	S
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	9	10

Twitter's week year bug

ISO 8601 rule: *The first week of the year is the week containing the first Thursday.*

"So if January 1 falls on a Friday, it belongs to the last week of the previous year. If December 31 falls on a Wednesday, it belongs to week 01 of the following year."

Use yyyy instead of YYYY

```
DateTimeFormatter.ofPattern("dd MMMM yyyy").format(zonedDateTime)
```

Twitter kicks Android app users out for five hours due to 2015 date bug

The social network celebrated 2015 in style, by breaking its Android app and mobile website - and all, it seems, because of one misplaced letter



© Crashy bird: Twitter was down for five hours overnight. Photograph: Richard Drew/AP

If you're worried about how your New Year's Eve will go, don't. It's not even 2015 yet, and Twitter's already had a worse one than you.

The service was down for many users over five and a half hours on Monday morning UK time, between midnight and 5am (7pm to midnight ET, and 4pm to 9pm PT), after a bug in a line of code caused the service to think that it

Could you have found them?

- How often would those bugs trigger?
- Driver bug:
 - What happens if you return from a driver with interrupts disabled?
 - Consider: that's one function
 - ...in a 2000 LOC file
 - ...in a module with 60,000 LOC
 - ...IN THE LINUX KERNEL

Some defects are very difficult to find via testing, inspection.

Defects of interest...

- Are on uncommon or difficult-to-force execution paths. (vs testing)
- Executing (or interpreting/otherwise analyzing) all paths concretely to find such defects is infeasible.
- What we really want to do is check the **entire possible state space** of the program for particular properties.
- What we **CAN** do is check an **abstract state space** of the program for particular properties.

Activity: Analyze the Python program statically

```
def n2s(n: int, b: int):
    if n <= 0: return '0'
    r = ''
    while n > 0:
        u = n % b
        if u >= 10:
            u = chr(ord('A') + u-10)
        n = n // b
        r = str(u) + r
    return r
```

1. What are the set of data types taken by variable **u** at any point in the program?
2. Can the variable **u** be a negative number?
3. Will this function always return a value?
4. Can there ever be a division by zero?
5. Will the returned value ever contain a minus sign '-'?

What is Static Analysis?

- **Systematic** examination of an **abstraction** of program **state space**.
 - Does not execute code! (like code review)
- **Abstraction:** produce a representation of a program that is simpler to analyze.
 - Results in fewer states to explore; makes difficult problems tractable.
- Check if a **particular property** holds over the entire state space:
- Liveness: “something good eventually happens.”
 - Safety: “this bad thing can’t ever happen.”
 - Compliance with mechanical design rules.

What static analysis can and cannot do

- **Type-checking** is well established
 - Set of data types taken by variables at any point
 - Can be used to prevent type errors (e.g. Java) or warn about potential type errors (e.g. Python)
- Checking for **problematic patterns** in syntax is easy and fast
 - Is there a comparison of two Java strings using `==`?
 - Is there an array access `a[i]` without an enclosing bounds check for `i`?

What static analysis can and cannot do

- Reasoning about **termination** is **impossible** in general
 - Halting problem
- Reasoning about **exact values is hard**, but conservative analysis via abstraction is possible
 - Is the bounds check before `a[i]` guaranteeing that `I` is within bounds?
 - Can the divisor ever take on a zero value?
 - Could the result of a function call be `42`?
 - Will this multi-threaded program give me a deterministic result?
 - Be prepared for “**MAYBE**”
- Verifying some advanced properties is possible but expensive
 - CI-based static analysis usually over-approximates conservatively

The Bad News: Rice's Theorem

Every static analysis is necessarily incomplete, unsound, undecidable, or a combination thereof

“Any nontrivial property about the language recognized by a Turing machine is undecidable.”

Henry Gordon Rice, 1953

Static Analysis is well suited to detecting certain defects

- **Security:** Buffer overruns, improperly validated input...
- **Memory safety:** Null dereference, uninitialized data...
- **Resource leaks:** Memory, OS resources...
- **API Protocols:** Device drivers; real time libraries; GUI frameworks
- **Exceptions:** Arithmetic/library/user-defined
- **Encapsulation:**
 - Accessing internal data, calling private functions...
- **Data races:**
 - Two threads access the same data without synchronization

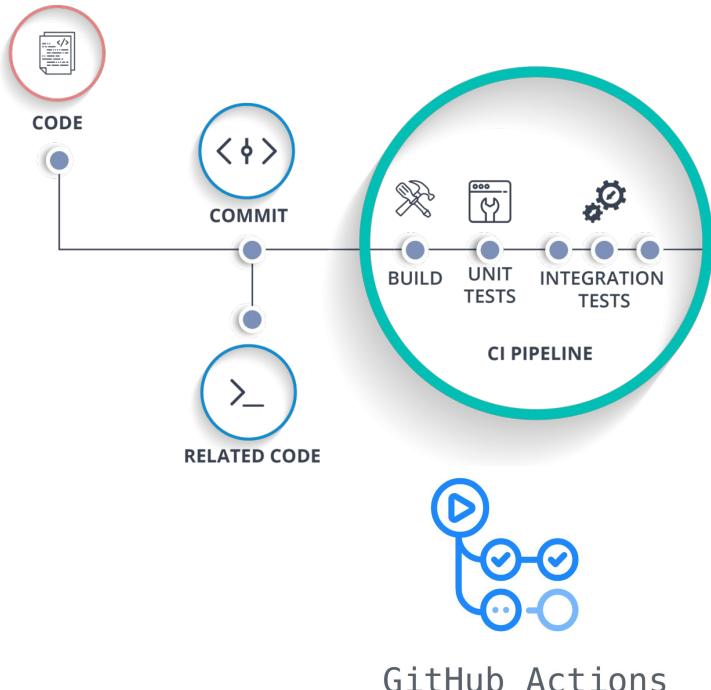
Outline

- `goto fail;` and similar unfamous bugs
- **Static analysis tools**
 - Linters for maintainability
 - Pattern-based static analyzers
- Challenges of static analysis

Tools for Static Analysis



Static analysis is a key part of CI



SonarCloud interface showing static analysis results:

- public/scss/admin/settings.scss:
 - Intentionality: Remove this commented out code. (Maintainability, Code Smell, Major)
- public/scss/modules/bottom-sheet.scss:
 - Intentionality: Unexpected empty source. (Maintainability, Code Smell, Major)
- public/scss/modules/picture-switcher.scss:
 - Intentionality: Unexpected duplicate "padding". (Reliability, Bug, Major)
 - Intentionality: Unexpected missing generic font family. (Reliability, Bug, Major)

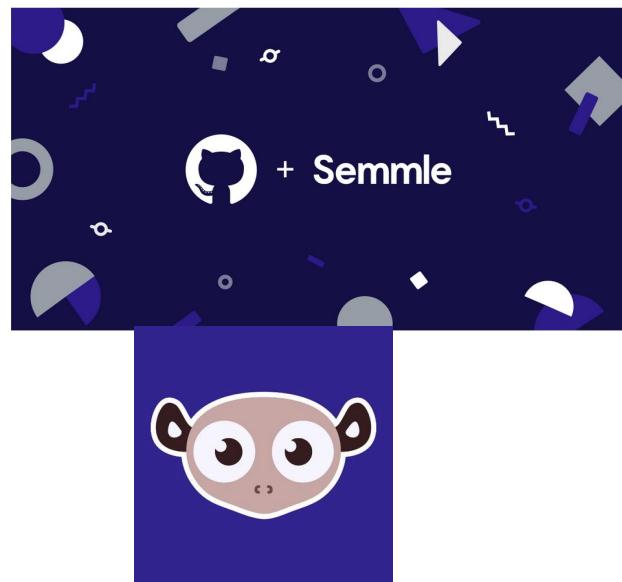
sonarcloud

Static analysis used to be an academic amusement; now it's heavily commercialized

GitHub acquires code analysis tool Semmle

Frederic Lardinois @fredericl / 1:30 pm EDT • September 18, 2019

Comment



Marketplace Search results

Types Apps Actions

Categories API management Chat Code quality Code review Continuous integration Dependency management Deployment IDEs Learning Localization Mobile Monitoring Project management Publishing

Search for apps and actions

Apps

Built on your workflow with apps that integrate with GitHub.

306 results filtered by Apps

App	Description
Zube	Agile project management that lets the entire team work with developers on GitHub
Crowdin	Agile localization for your projects
BackHub	Reliable GitHub repository backup, set up in minutes
Codacy	Automated code reviews to help developers ship better software, faster
Semaphore	Test and deploy at the push of a button
DeepScan	Advanced static analysis for automatically finding runtime errors in JavaScript code
WhiteSource Bolt	Detect open source vulnerabilities in real time with suggested fixes for quick remediation
Slack + GitHub	Connect your code without leaving Slack
GiltLocalize	Continuous Localization for GitHub projects
Code Climate	Automated code review for technical debt and test coverage
Flapstastic	Manage failing unit tests. Click a checkbox to instantly disable any test on all branches. Works with your current test suite
Depfu	Automated dependency updates done right



<https://www.sdccentral.com/articles/news/snyk-secures-150m-snags-1b-valuation/2020/01/>

<https://techcrunch.com/2019/09/18/github-acquires-code-analysis-tool-semmle/>

<https://github.com/marketplace>

News

Snyk Secures \$150M, Snags \$1B Valuation



Sydney Sawaya | Associate Editor
January 21, 2020 1:12 PM



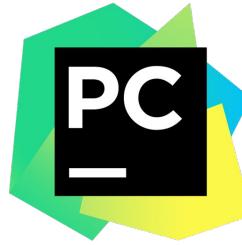
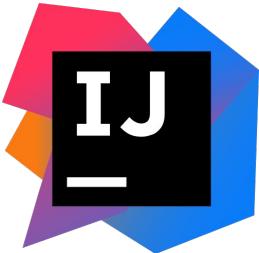
Snyk, a developer-focused security startup that identifies vulnerabilities in open source applications, announced a \$150 million Series C funding round today. This brings the company's total investment to \$250 million alongside reports that put the company's valuation at more than \$1 billion.



snyk

Carnegie
Mellon
University

Static analysis is also integrated into IDEs



eclipse

A screenshot of the Visual Studio Code interface. The left pane shows a C++ file named 'cppcoreguidelines.cpp'. A yellow bar at the top of the code editor indicates inspection results for Clang-Tidy. The code contains several violations highlighted with red underlines and error messages. One specific violation is highlighted with a yellow box and the message 'Do not use pointer arithmetic'.

```
// To enable only C++ Core Guidelines checks
// go to Settings/Preferences | Editor | Inspections | C/C++ | Clang-Tidy
// and provide: -*,cppcoreguidelines-* in options

void fill_pointer(int* arr, const int num) {
    for(int i = 0; i < num; ++i) {
        arr[i] = 0;
    }
}

void fill_array(int ind) {
    int arr[3] = {1,2,3};
    arr[ind] = 0;
}

void cast_away_const(const int& magic_num)
{
    const_cast<int&>(magic_num) = 42;
}
```

A screenshot of the SonarQube IDE integration in an IDE. The left pane shows a Java file with inspection results. The right pane displays detailed analysis results for a specific vulnerability: 'Cross-site Scripting (XSS)'.

Cross-site Scripting (XSS)
Vulnerability: CWE-79
Unsanitized input from the HTTP request body flows into send, where it is used to render an HTML page returned to the user. This may Scanning attack (XSS).

Data Flow - 12 steps

```
1 index.js:8 | var item = req.body.content;
2 index.js:8 | // ...item.replace(...).match("...") && item.matchingRegex) {
3 index.js:9 | Check if the item matches the regular expression;
4 index.js:10 | if(item.matchingRegex) {
5 index.js:15 | var t = todo;
6 index.js:15 | var reminder = t.todoString().indexOf(reminder);
7 index.js:16 | var time = t.todoString().slice(reminder + reminder.length);
8 index.js:16 | var time = time.substring(0, time.indexOf(reminder));
9 index.js:17 | return t;
```

Linters

Cheap, fast, and lightweight static source analysis

Use linters to improve maintainability

Why? We spend more time reading code than writing it.

- Developers spend most of their time maintaining code
 - Various estimates of the exact %, some as high as 80%
- Code ownership is usually shared
- The original owner of some code may move on
- Code conventions make it easier for other developers to quickly understand your code

Use Style Guidelines to facilitate communication

- Indentation
- Comments
- Line length
- Naming
- Directory structure
- ...



Style Guidelines

This document collects the emerging principles, conventions, abstractions, and best practices for writing Rust code.

Since Rust is evolving at a rapid pace, these guidelines are preliminary. The hope is that writing them down explicitly will help drive discussion, consensus and adoption.

Whenever feasible, guidelines provide specific examples from Rust's standard libraries.

Guideline statuses

Every guideline has a status:

- [FIXME]: Marks places where there is more work to be done. In some cases, that just means going through the RFC process.
- [FIXME #NNNN]: Like [FIXME], but links to the issue tracker.
- [RFC #NNNN]: Marks accepted guidelines, linking to the rust-lang RFC establishing them.

Guideline stabilization

One purpose of these guidelines is to reach decisions on a number of cross-cutting API and stylistic choices. Discussion and development of the guidelines will happen primarily on <http://discuss.rust-lang.org/>, using the Guidelines category. Discussion can also occur on the [guidelines issue tracker](#).

Guidelines that are under development or discussion will be marked with the status [FIXME], with a link to the issue tracker when appropriate.

Once a concrete guideline is ready to be proposed, it should be filed as an [FIXME]-needs-RFC. If the RFC is accepted, the official guidelines will be updated to match, and will include the tag [RFC #NNNN] linking to the RFC document.

What's In This Document

This document is broken into four parts:

- **Style**: provides a set of rules governing naming conventions, whitespace, and other stylistic issues.
- **Guidelines by Rust feature**: places the focus on each of Rust's features, starting from expressions and working the way out toward crates, dispensing guidelines relevant to each.
- **Topical guidelines and patterns**: The rest of the document proceeds by cross-cutting topic, starting with Ownership and resources.
- **APIs for a changing Rust**: discusses the forward-compatibility hazards, especially those that interact with the pre-1.0 library stabilization process.



Guidelines are inherently opinionated, but **consistency** is the important point.
Agree to a set of conventions and stick to them.

Use linters to enforce style guidelines
Don't rely on manual inspection during code review!

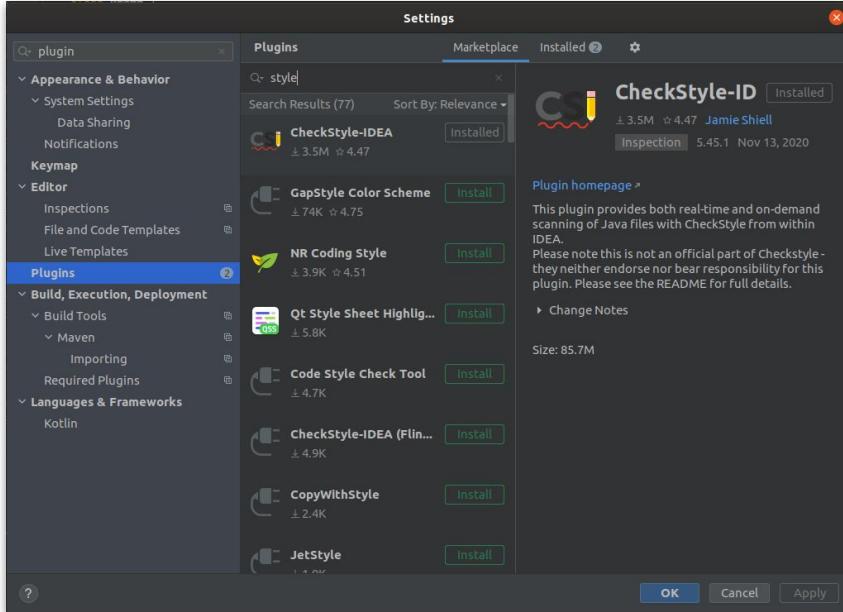


RuboCop



Automatically reformat your existing code

Developer time is valuable!



Style is an easy way to improve readability

- Everyone has their own opinion (e.g., tabs vs. spaces)
- Agree to a convention and stick to it
 - Use continuous integration to enforce it
- Use automated tools to fix issues in existing code

Pattern-based Static Analysis Tools

- Bad Practice
- Correctness
- Performance
- Internationalization
- Malicious Code
- Multithreaded Correctness
- Security
- Dodgy Code



FindBugs Bug Descriptions
This document lists the standard bug patterns reported by FindBugs version 3.0.1.

Summary

Description	Category
BC: Class method should not assume anything about the type of its argument	Bad practice
BL: Class has no public constructor	Bad practice
CN: Class implements Cloneable but does not define or use clone method	Bad practice
CS: clone method does not call super.clone()	Bad practice
DC: Class definition does not contain documentation	Bad practice
DN: Class definition does not contain class definition	Bad practice
CNT: Round value of known constant found	Bad practice
Co: Abstract class defines concrete equals method	Bad practice
Co: Abstract class defines concrete hashCode method	Bad practice
Miscarval[1][2][3]	Bad practice
PNL: Bug description[1][2][3]	Bad practice
PNL: Bug description[1][2][3]	Bad practice
PC: Comparable.equals() returns Integer.MIN_VALUE	Bad practice
Co: comparableFor[1][2][3] returns false or null	Bad practice
DE: Method might drop exception	Bad practice
DE: Method might ignore exception	Bad practice
DM: Dead code	Bad practice
DM: Dead code found in a loop due to reuse of Entry objects	Bad practice
DM: Random object created and used only once	Bad practice
DM: Don't use removeall to clear a collection	Bad practice
DM: Don't use removeall to clear a collection	Bad practice
DM: Method invokes dangerous method runOnUiThread	Bad practice
ES: Comparison of String parameter value == <value>	Bad practice
ES: Comparison of String parameter value == <value>	Bad practice
EG: Abstract class defines concrete equals method	Bad practice
EG: Equals checks for incompatible operand types	Bad practice
EG: Equals method fails for subtypes	Bad practice
EG: Comparable.equals() method defined	Bad practice
EG: Comparable.equals() method defined	Bad practice
EL: Explicit invocation of finalizer	Bad practice
EL: Finalizer nulls field	Bad practice
EL: Finalizer nulls field	Bad practice
EL: Finalizer does not call superclasses' finalizer	Bad practice
EL: Finalizer does nothing but call superclass' finalizer	Bad practice
FS: Format string should use %s rather than %n	Bad practice
GD: Dead code	Bad practice
HE: Class defines equals() but not hashCode()	Bad practice
HE: Class defines hashCode() but not equals()	Bad practice
HE: Class defines hashCode() and uses Object.equals()	Bad practice
HE: Class defines hashCode() and uses Object.equals()	Bad practice
HE: Class inherits equals() and uses Object.hashCode()	Bad practice
IC: Inconsistent class definition	Bad practice
IMSE: Dangerous catching of IllegalMonitorStateException	Bad practice
IS: Needless instantiation of class that only supplies static methods	Bad practice
IR: Instantiation of recursive class that causes stack overflow exception	Bad practice
JPFR: Slice of non serializable object into HttpSession	Bad practice
CP: Fields of immutable classes should be final	Bad practice
ME: Public enum method unconditionally sets its field	Bad practice



Example: Bad Practice

```
String x = new String("Foo");
String y = new String("Foo");

if (x == y) {
    System.out.println("x and y are the same!");
} else {
    System.out.println("x and y are different!");
}
```

ES_COMPARING_STRINGS_WITH_EQ
Comparing strings with ==

Example: Bad Practice

```
String x = new String("Foo");
String y = new String("Foo");

if (x == y) {
    if (x.equals(y)) {
        System.out.println("x and y are the same!");
    } else {
        System.out.println("x and y are different!");
    }
}
```

ES_COMPARING_STRINGS_WITH_EQ
Comparing strings with ==

Example: Performance

```
public static String repeat(String string, int times)
{
    String output = string;
    for (int i = 1; i < times; ++i) {
        output = output + string;
    }
    return output;
}
```

SBSC_USE_STRINGBUFFER_CONCATENATION

Method concatenates strings using + in a loop

Example: Performance

```
public static String repeat(String string, int times)
{
    StringBuffer output = new StringBuffer(string);
    for (int i = 1; i < times; ++i) {
        output.append(string);
    }
    return output.toString();
}
```

SBSC_USE_STRINGBUFFER_CONCATENATION

Method concatenates strings using + in a loop

Use type annotations to detect common errors

- Uses a conservative analysis to prove the absence of certain defects:
 - Unsanitized input, Null pointer errors, uninitialized fields, certain liveness issues, information leaks, SQL injections, bad regular expressions, incorrect physical units, bad format strings, ...
- Assuming that code is annotated and those annotations are correct
- Use annotations to enhance type system



Remember the Mars Climate Orbiter incident from 1999?

SIMSCALE Blog Product Solutions Learning Public Projects Case Studies Careers Pricing Log In Sign Up

When NASA Lost a Spacecraft Due to a Metric Math Mistake

WRITTEN BY Ajay Harish UPDATED ON March 10th, 2020 APPROX READING TIME 11 Minutes

Blog > CAE Hub > When NASA Lost a Spacecraft Due to a Metric Math Mistake

In September of 1999, after almost 10 months of travel to Mars, the Mars Climate Orbiter burned and broke into pieces. On a day when NASA engineers were expecting to celebrate, the ground reality turned out to be completely different, all because someone failed to use the right units, i.e., the metric units! The Scientific American Space Lab made a brief but interesting video on this very topic.

NASA'S LOST SPACECRAFT
The Metric System and NASA's Mars Climate Orbiter

The Mars Climate Orbiter, built at a cost of \$125 million, was a 338-kilogram robotic space probe launched by NASA on December 11, 1998 to study the Martian climate, Martian atmosphere, and surface changes. In addition, its function was to act as the communications relay in the Mars Surveyor '98 program for the Mars Polar Lander. The navigation team at the Jet Propulsion Laboratory (JPL) used the metric system of millimeters and meters in its calculations, while

Does this program compile? No

```
void demo() {  
    @m int x;  
    x = 5 * m;  
  
    @m int meters = 5 * m;  
    @s int seconds = 2 * s;  
  
    @mPERs int speed = meters / seconds;  
    @m int foo = meters + seconds;  
    @s int bar = seconds - meters;  
}
```

@m indicates that x represents meters

To assign a unit, multiply appropriate unit constant

```
In [1]: from astropy import units as u
```

though note that this will conflict with any variable called `u`.

Units can then be accessed with:

```
In [2]: u.m
```

```
Out[2]: m
```

```
In [3]: u.pc
```

```
Out[3]: pc
```

```
In [4]: u.s
```

```
Out[4]: s
```

```
In [5]: u.kg
```

```
Out[5]: kg
```

We can create composite units:

```
In [6]: u.m / u.kg / u.s**2
```

```
Out[6]:  $\frac{\text{m}}{\text{kg s}^2}$ 
```

```
In [7]: repr(u.m / u.kg / u.s**2)
```

```
Out[7]: 'Unit("m / (kg s2)")'
```

Equivalencies

Equivalencies can be used to convert quantities that are not strictly the same physical type:

```
: (450. * u.nm).to(u.GHz)
```

```
-----  
UnitConversionError                                     Traceback (most recent call last)  
/sw/lib/python3.4/site-packages/astropy/units/core.py in _get_converter(self, other, equivalencies)  
    865     try:  
--> 866         scale = self._to(other)  
    867     except UnitsError:
```

“Malicious” User Inputs

```
void processRequest() {  
    String input = getUserInput();  
    String query = "SELECT ... " + input;  
    executeQuery(query);  
}
```

Taint Analysis

Prevents untrusted (tainted) data from reaching sensitive locations (sinks)



Taint Checking using Annotations

```
void processRequest() {  
    @Tainted String input = getUserInput();  
    executeQuery(input);  
}  
  
public void executeQuery(@Untainted String input) {  
    // ...  
}  
  
@Untainted public String validate(String userInput) {  
    // ...  
}
```

Indicates that data is tainted

Argument ***must*** be untainted

Guarantees that return value is untainted

Does this program compile? No

```
void processRequest() {  
    @Tainted String input = getUserInput();  
    if (input.contains("drop tables")) {  
        input = validate(input);  
    }  
    executeQuery(input);  
}
```

input is NOT
guaranteed to be
@Untainted

Does this program compile? Yes

```
void processRequest() {  
    @Tainted String input = getUserInput();  
    input = validate(input);  
    executeQuery(input);  
}
```

Outline

- **goto fail;** and similar unfamous bugs
- Static analysis tools
 - Linters for maintainability
 - Pattern-based static analyzers
- **Challenges of static analysis**

What makes a good static analysis tool?

- Static analysis should be **fast**
 - Don't hold up development velocity
 - This becomes more important as code scales
- Static analysis should report **few false positives**
 - Otherwise developers will start to ignore warnings and alerts, and quality will decline
- Static analysis should be **continuous**
 - Should be part of your continuous integration pipeline
 - Diff-based analysis is even better -- don't analyse the entire codebase; just the changes
- Static analysis should be **informative**
 - Messages that help the developer to quickly locate and address the issue
 - Ideally, it should suggest or automatically apply fixes

Lessons for Static Analysis Tools at Google

- Make It a Compiler Workflow
- Value of compiler checks.
- Reporting issues sooner is better
- Warn During Code Review
- Engineers working on static analysis must demonstrate impact through hard data.

contributed articles

DOI:10.1145/3188720

For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAK

**Lessons
from Building
Static Analysis
Tools at Google**



Lessons for Static Analysis Tools at Google

- Finding bugs is easy
- Most developers will not go out of their way to use static analysis tools.
- Developer happiness is key.
- Do not just find bugs, fix them.
- Crowdsource analysis development.

contributed articles

DOI:10.1145/3188720

For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAK

**Lessons
from Building
Static Analysis
Tools at Google**



Reasons engineers do not always use static analysis tools or ignore their warnings

- Not integrated.
 - The tool is not integrated into the developer's workflow or takes too long to run
- Not actionable
 - Whenever possible, the error should include a suggested fix that can be applied mechanically
- Not trustworthy
 - Users do not trust the results
- Not manifest in practice.
 - The reported bug is theoretically possible, but the problem does not actually manifest in practice
- Too expensive to fix.
 - Fixing the detected bug is too expensive or risky
- Warnings not understood

What you need to know



Early feedback through code reviews and static analysis is crucial for reducing risk and preventing technical errors.



Effective code reviews combine structured checklists with an empathetic, constructive approach to foster collaboration and improve code quality.



Static analysis tools enhance code quality and maintainability while integrating seamlessly with CI for continuous checks.



Static analysis has strengths in detecting issues like security vulnerabilities and performance problems, but it also has limitations and challenges.