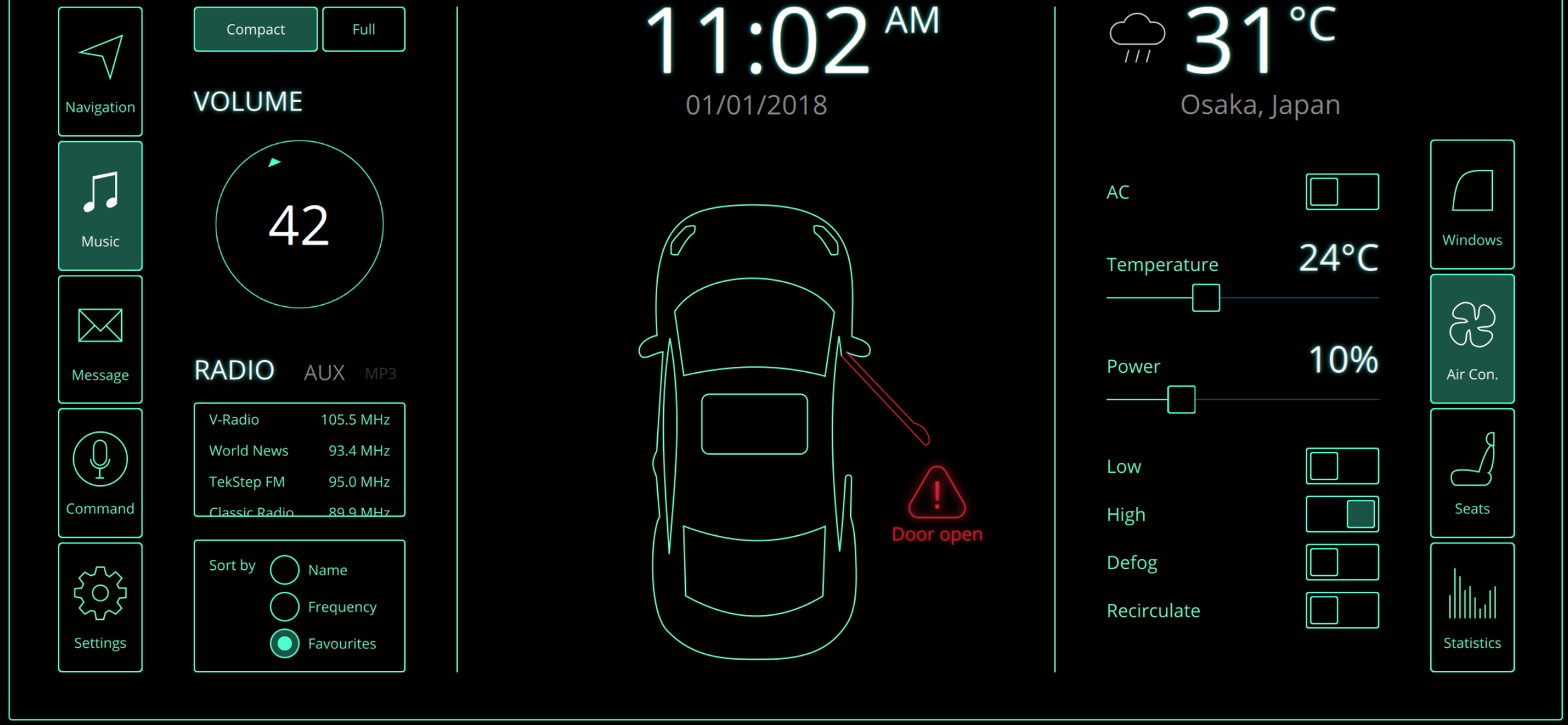


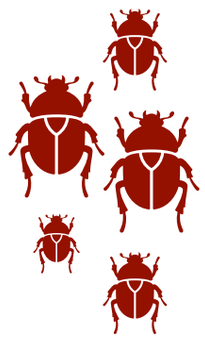
Boost Continuous Formal Verification in Industry

Felipe R. Monteiro
Mikhail R. Gadelha
Lucas C. Cordeiro

Consumer electronic products must be **as robust and bug-free as possible**, given that even medium product-return rates tend to be unacceptable

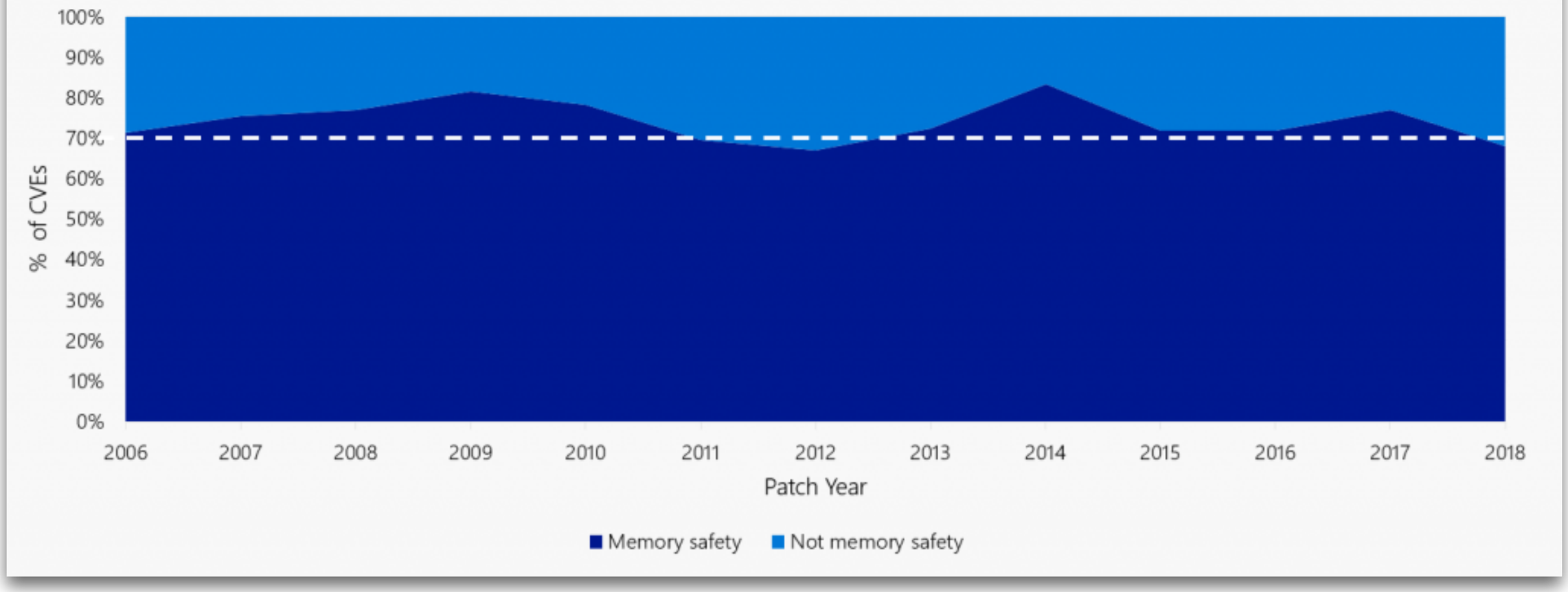
Consumer electronic products must be **as robust and bug-free as possible**, given that even medium product-return rates tend to be unacceptable

- *"Mozilla browser has around **20,000** open bugs"*
  Amir Michail, ICSE, 2005.



Consumer electronic products must be **as robust and bug-free as possible**, given that even medium product-return rates tend to be unacceptable

- "*Engineers reported the static analyser **Infer** was key to build a concurrent version of Facebook app to the Android platform.*"
  Peter O'Hearn, FLoC, 2018.



- "*The majority of vulnerabilities are caused by developers inadvertently inserting* ***memory corruption bugs into their C and C++ code****. As Microsoft increases its code base and uses more Open Source Software in its code,* ***this problem isn't getting better, it's getting worse.***"

  Matt Miller, Microsoft Security Response Centre, 2019.

"Formal automated reasoning is one of the investments that AWS is making in order to facilitate continued simultaneous growth in both functionality and security."

Byron Cook, FLoC, 2018.

“As analysis-tool developers, we must measure our success in terms of defects corrected, not the number presented to developers. This means our responsibility extends far beyond the analysis tool itself.”

Sadowski *et al.*, ACM Comm., 2018.

“*There has been a tremendous amount of valuable research in formal methods, but rarely have formal reasoning techniques been deployed as part of the development process of large industrial codebases.*”

Peter O’Hearn, FLoC, 2018.

facebook research

Our main vision is to…

Integrate formal verification techniques into the workflow of the main software development methodologies

We focus on…

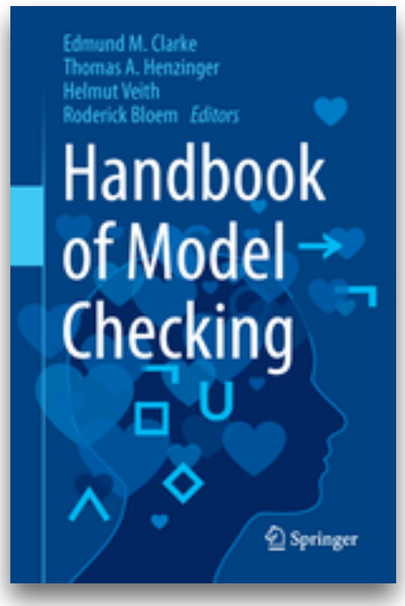
Software model checking techniques combined with DevOps culture, particularly, continuous integration

We focus on…

Software model checking techniques combined with DevOps culture, particularly, continuous integration

"***The main challenge is scalability***: *real-world software systems not only include complex control and data structure but depend on much "context" such as libraries and interfaces to other code, including lower-level systems code. As a result, proving a software system correct requires much more effort, knowledge, training, and ingenuity than writing the software in trial-and-error style.*"

E. M. Clarke et al., Handbook of Model Checking, 2018.



Continuous Formal Verification

- "***Continuous Formal Verification***." Formal verification of s2n, the open-source TLS implementation used in numerous Amazon services.

  Chudnov *et al.*, 2018.

- "***Continuous Reasoning***." Formal reasoning about a (changing) codebase is done in a fashion which mirrors the iterative, continuous model of software development that is increasingly practiced in industry.

  O'Hearn *et al.*, 2005.

- "***Continuous Verification***." Detect design errors as quickly as possible by exploiting information from the software configuration management system.

  Cordeiro *et al.*, 2010.

- "***Continuous Verification***." Adoption of verification activities throughout the development process rather than relying on a testing phase towards the end of development.

  Chang *et al.*, 1997.

Continuous Formal Verification

- "***Continuous Formal Verification***." Formal verification of s2n, the open-source TLS implementation used in numerous Amazon services.

  Chudnov *et al.*, 2018.

- "***Continuous Reasoning***." Formal reasoning about a (changing) codebase is done in a fashion which mirrors the iterative, continuous model of software development that is increasingly practiced in industry.

  O'Hearn *et al.*, 2005.

- "***Continuous Verification***." Detect design errors as quickly as possible by exploiting information from the software configuration management system.

  Cordeiro *et al.*, 2010.

- "***Continuous Verification***." Adoption of verification activities throughout the development process rather than relying on a testing phase towards the end of development.

  Chang *et al.*, 1997.

Continuous Formal Verification

I. Decompose software systems into submodules of manageable complexity.

Chaki *et al.*, ICSE, 2003.

II. Design verification tasks for each submodule using general inputs, e.g., taking advantage of symbolic execution.

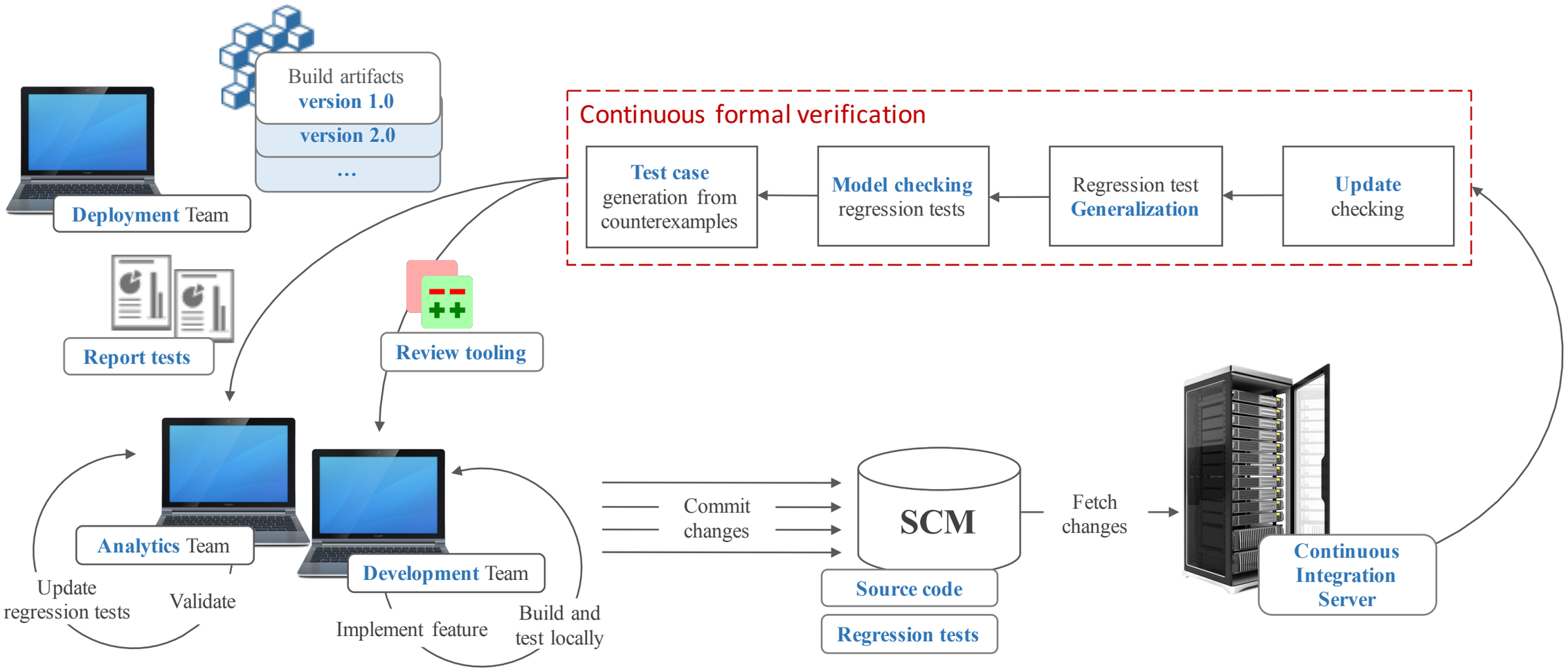
Cadar *et al.*, ICSE, 2011.

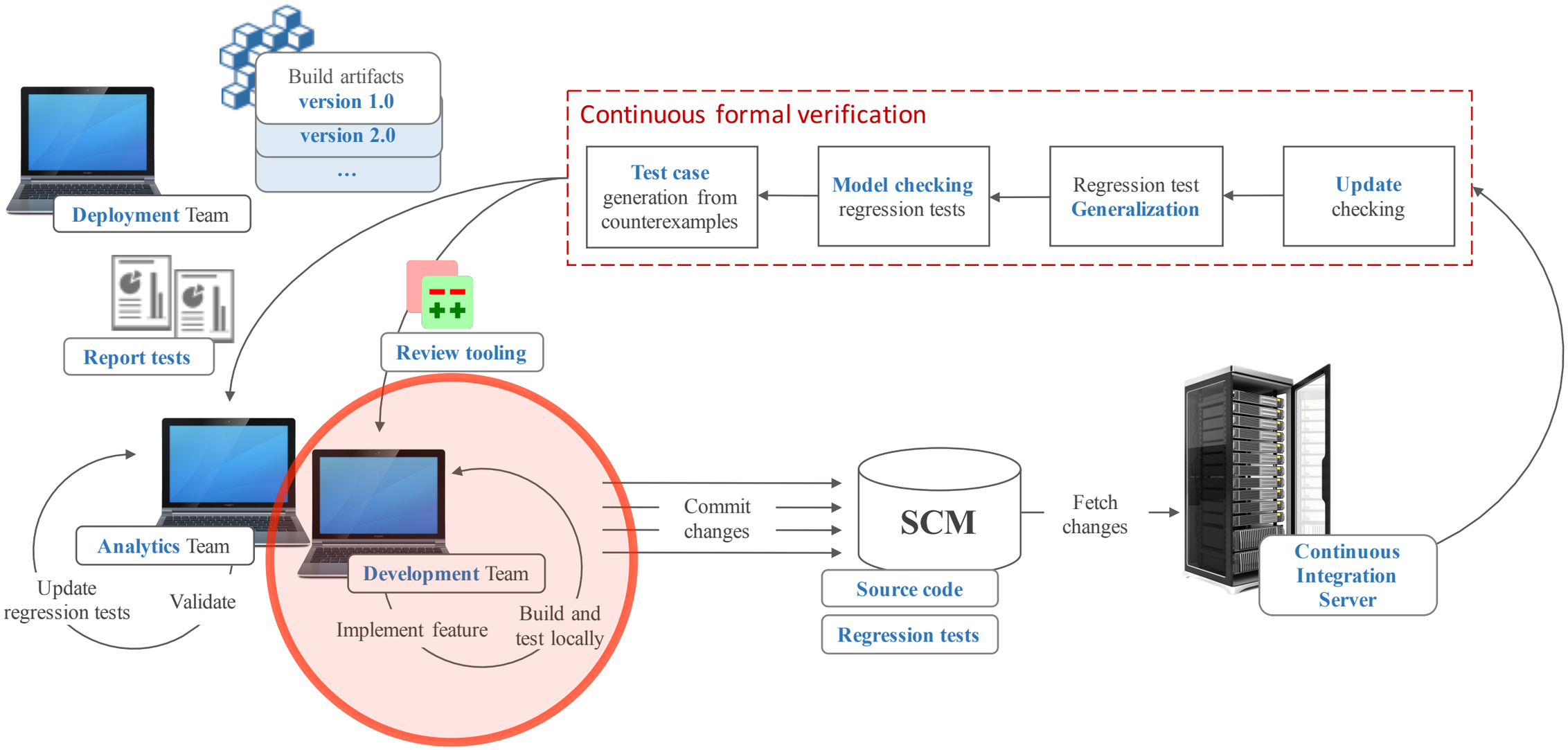
III. Given a code change, we must compute the blast radios in the software system and run all verification tasks for the relevant modules.

Fedyukovich *et al.*, TACAS, 2013.

IV. Report results at diff-time with witnesses for failed verification tasks.

Beyer *et al.*, ICSE, 2004.

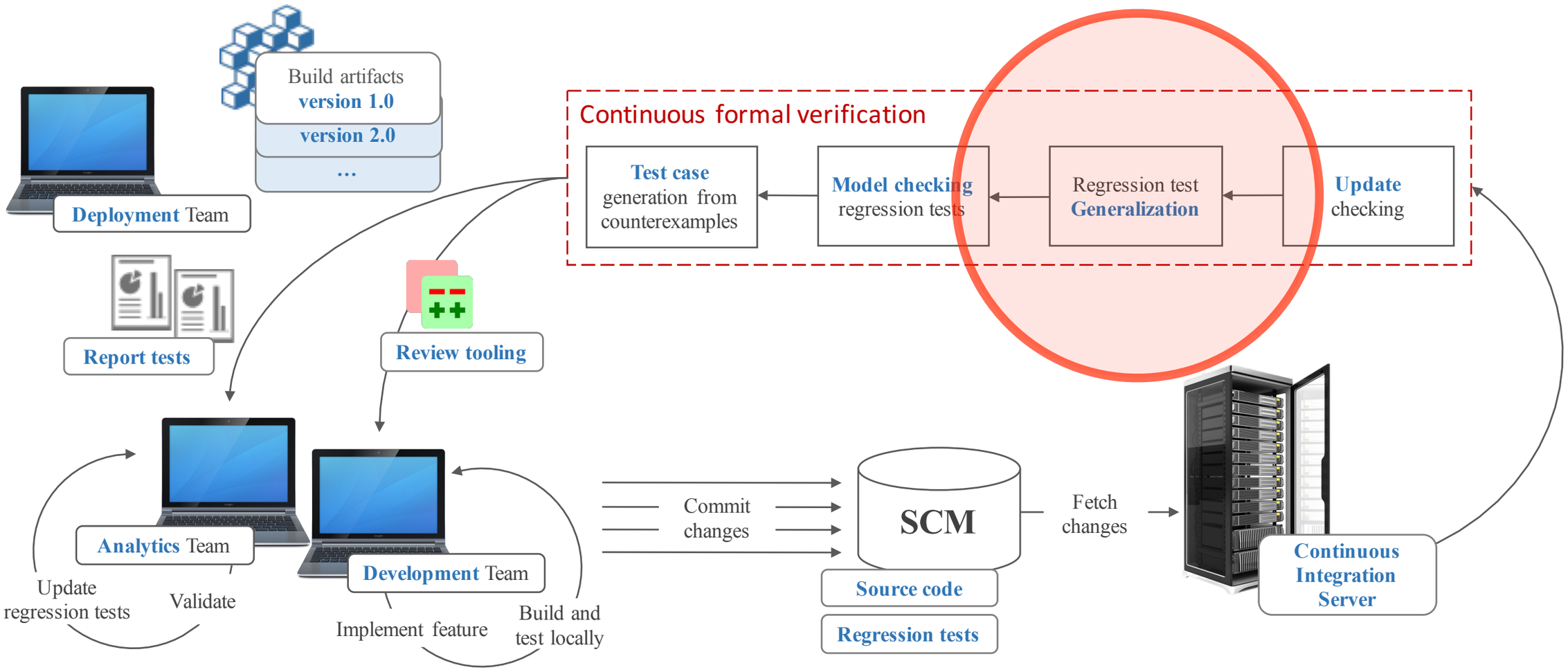

Build artifacts
version 1.0
version 2.0
...
Deployment Team
Continuous formal verification
Test case generation from counterexamples
Model checking regression tests
Regression test Generalization
Update checking
Report tests
Review tooling
Analytics Team
Development Team
Update regression tests
Validate
Implement feature
Build and test locally
Commit changes
SCM
Source code
Regression tests
Fetch changes
Continuous Integration Server



I. Decompose software systems into submodules of manageable complexity.

regression/unit testing

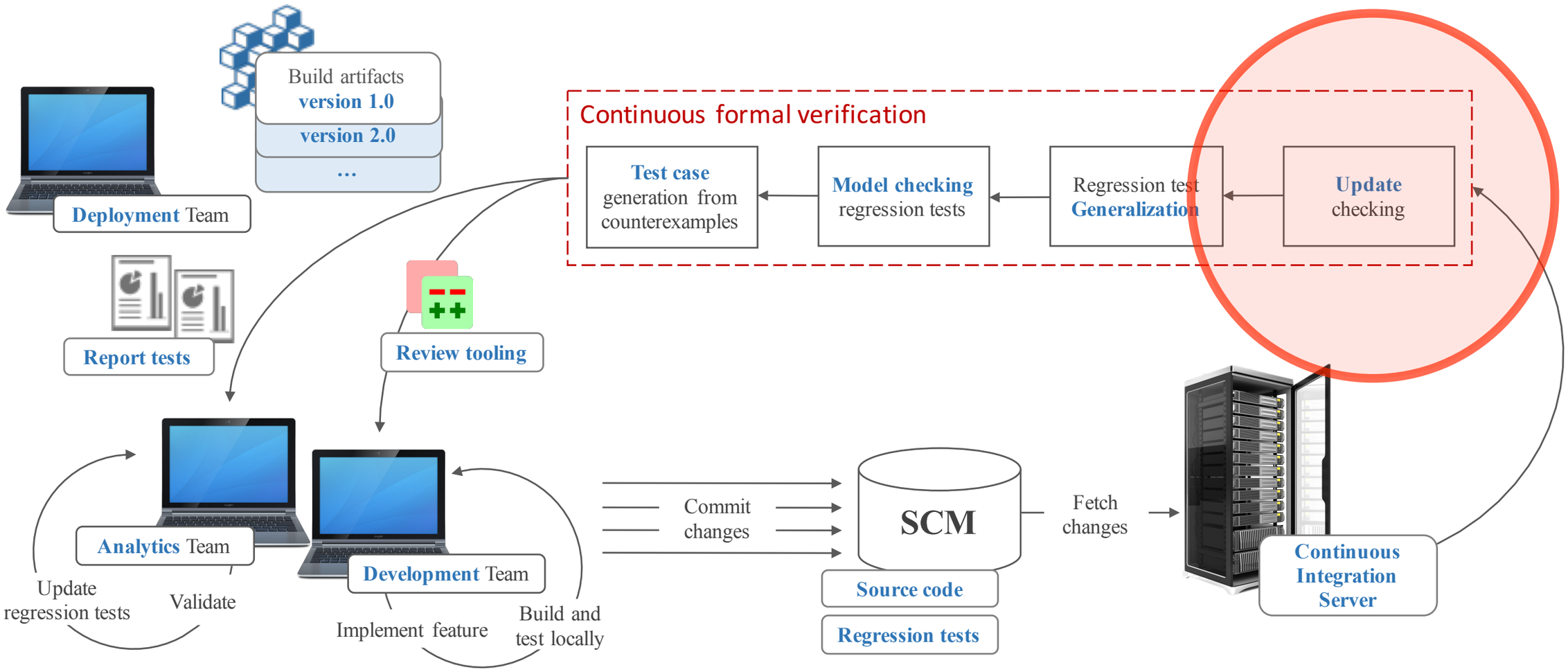
Chaki *et al.*, ICSE, 2003.



II. Design verification tasks for each submodule using general inputs, e.g., taking advantage of symbolic execution.

model checking + symbolic execution

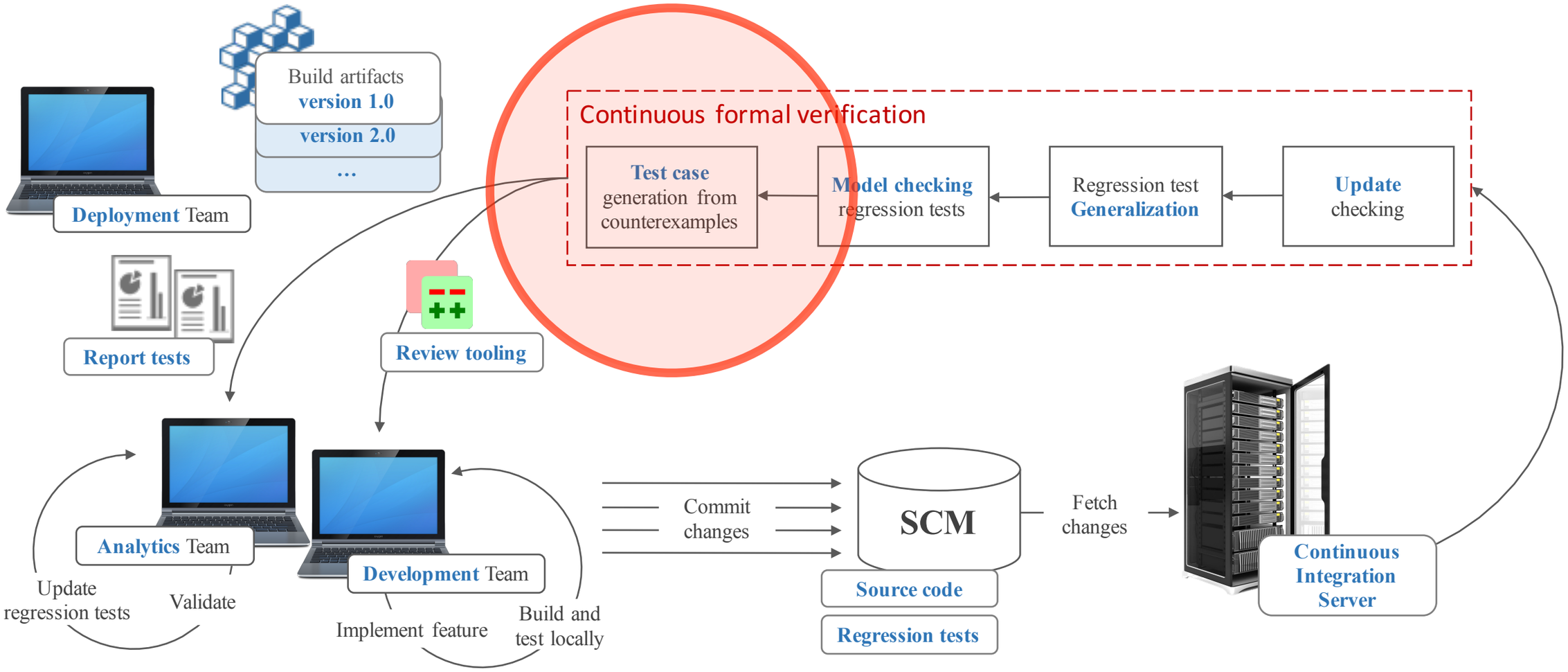
Cadar *et al.*, ICSE, 2011.



III. Given a code change, we must compute the blast radios in the software system and run all verification tasks for the relevant modules.

update checking

Fedyukovich *et al.*, TACAS, 2013.



IV. Report results at diff-time with witnesses for failed verification tasks.

test case generation

Beyer *et al.*, ICSE, 2004.

Open challenges

Continuous Formal Verification

i. Modularity and complexity must be treated differently based on the software project

API vs. Systems

ii. Different properties required different approaches

Temporal Properties vs. Functional Properties

iii. Most of the software projects do not have a formal specification

Running Example

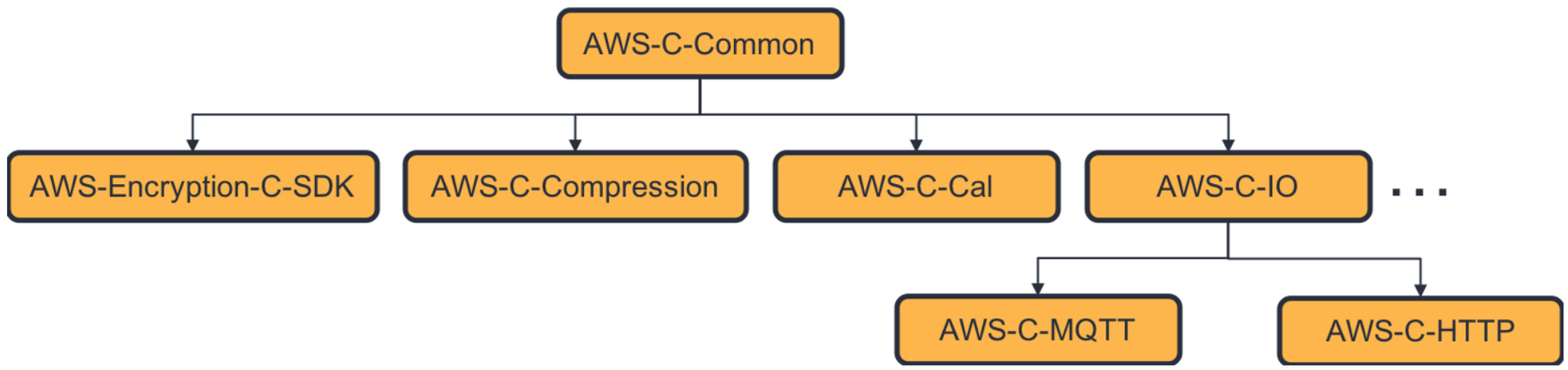
- Core c99 package for AWS SDK for C.

  Includes cross-platform primitives, configuration, data structures, and error handling.

- It already contains 171 proof harnesses, which were manually developed

  So how can we scale this process?

aws Labs



Generate proof harnesses from regression tests

```c
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        uint8_t arr_a[] = {'a', 'b', 'c'};
        const char *str_a = "abc";
        const char *str_b = "xyz";
        const char *str_c = "abcd";
        const char *empty = "";

        /* Simple */
        ASSERT_TRUE(aws_array_eq_c_str(arr_a, 3, str_a));
        ASSERT_FALSE(aws_array_eq_c_str(arr_a, 3, str_b));
        ASSERT_FALSE(aws_array_eq_c_str(arr_a, 3, str_c));

        /* Referencing self */
        ASSERT_TRUE(aws_array_eq_c_str(str_a, 3, str_a));
        ASSERT_FALSE(aws_array_eq_c_str(str_a, 2, str_a));

        /* Check length 0 */
        ASSERT_TRUE(aws_array_eq_c_str(arr_a, 0, empty));
        ASSERT_FALSE(aws_array_eq_c_str(arr_a, 0, str_a));

        /* NULL array is OK if length is 0 */
        ASSERT_TRUE(aws_array_eq_c_str(NULL, 0, empty));
        ASSERT_FALSE(aws_array_eq_c_str(NULL, 0, str_a));
    }

    return 0;
}
```

Initialize inputs

Check multiple scenarios

Generate proof harnesses from regression tests

```c
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Initialize inputs

Check all possible scenarios

Generate proof harnesses from regression tests

```c
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Generate proof harnesses from regression tests

```c
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Generate proof harnesses from regression tests

```
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Generate proof harnesses from regression tests

```c
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Generate proof harnesses from regression tests

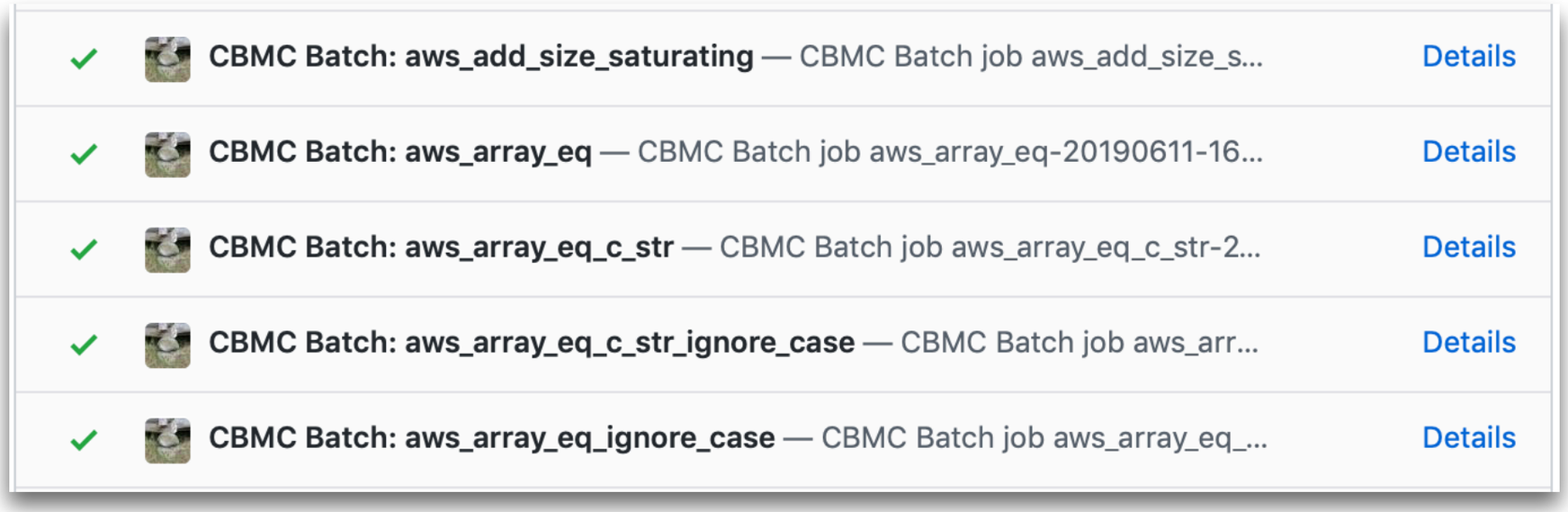
```
AWS_TEST_CASE(test_array_eq_c_str, s_test_array_eq_c_str)
static int s_test_array_eq_c_str(struct aws_allocator *allocator, void *ctx) {
    (void)ctx;
    (void)allocator;

    {
        size_t array_len;
        __CPROVER_assume(array_len <= MAX_BUFFER_SIZE);
        array = fn_can_fail_malloc(array_len);

        const char *nondet_str = fn_nondet_bool() ? NULL : fn_nondet_str(MAX_BUFFER_SIZE);

        /* pre-conditions */
        __CPROVER_assume(array || (array_len == 0));
        __CPROVER_assume(c_str);

        /* operation under verification */
        if (aws_array_eq_c_str(array, array_len, c_str)) {
            /* asserts equivalence */
            assert(array_len == str_len);
            if (array_len > 0) {
                assert_bytes_match((uint8_t *)array, (uint8_t *)nondet_str, array_len);
            }
        } else {
            /* assert non-equivalence */
        }
    }
    return 0;
}
```

Generate proof harnesses from regression tests

```
const char *fn_nondet_str(size_t max_size) {
    size_t cap;
    __CPROVER_assume(cap > 0 && cap <= max_size);
    const char *str = bounded_malloc(cap);
    __CPROVER_assume(str[cap - 1] == 0);
    return str;
}
```

Predicate example

Generate proof harnesses from regression tests



Continuous Integration

Open challenges

Automatically generate proof harnesses from regression tests

i. Deal with false negatives as the non-deterministic choice of values for program variables may force the exploration of paths that are infeasible in the original program.

ii. One may combine techniques to automatically generate tests based on counterexamples or source code.

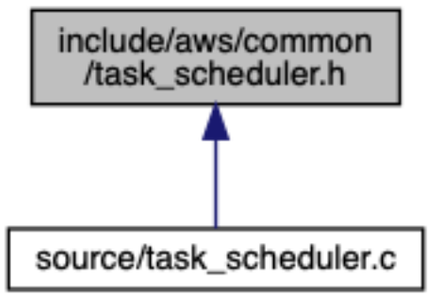
iii. We will also increase the power of this analysis by using conditional verifiers.

Checking for Relevant Code Changes

- The equivalence check will happen in two steps:

  (1) fast and imprecise abstract syntax tree (AST) structural equivalence check;

  (2) slow and precise formal check e.g. bounded model checking (BMC).

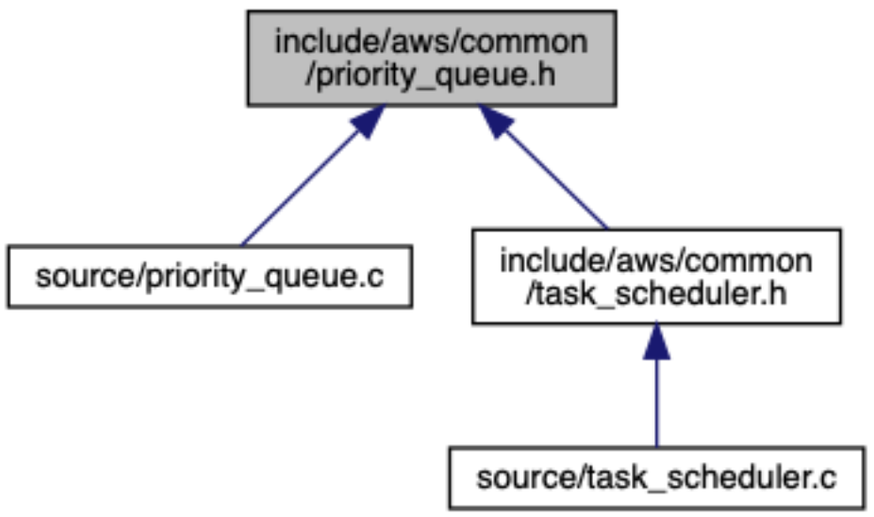


Checking for Relevant Code Changes

- The equivalence check will happen in two steps:

  (1) fast and imprecise abstract syntax tree (AST) structural equivalence check;

  (2) slow and precise formal check e.g. bounded model checking (BMC).

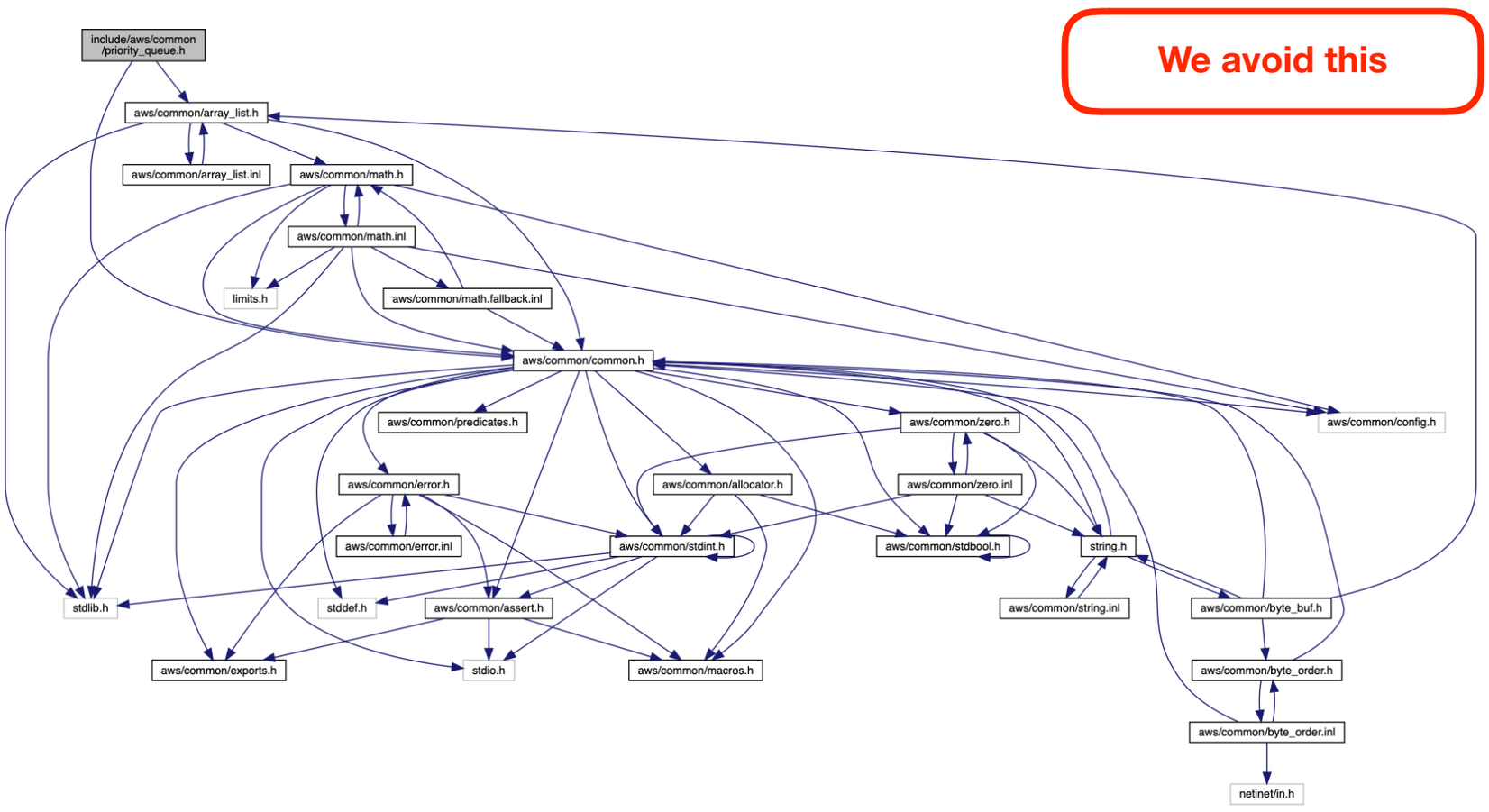


Checking for Relevant Code Changes

- The equivalence check will happen in two steps:

  (1) fast and imprecise abstract syntax tree (AST) structural equivalence check;

  (2) slow and precise formal check e.g. bounded model checking (BMC).



Open challenges

Efficiently detect relevant code changes and compute blast radius

i. There are many techniques that could be applied to perform equivalence checking such as SYMDIFF and CORK tools or through directed incremental symbolic execution (DiSE).

ii. Compare update checking with or tree diffs.

Conclusions

We make a call to the FM community to contribute to CFV, an approach with the potential to detect software vulnerabilities at scale

i. We are currently developing an automated software tool to tackle the key challenges of equivalence checking and test case generalization, so it can be applied to large open-source projects.

ii. We are also working in close collaboration with software developers at Samsung with the goal of integrating our automated reasoning tool into their workflow, thus increasing the adoption of formal methods in industry.

10th Workshop on Tools for Automatic Program Analysis
26th Static Analysis Symposium
3rd World Congress on Formal Methods

Boost Continuous Formal Verification in Industry

Position Paper

Felipe R. Monteiro, Mikhail R. Gadelha, and Lucas C. Cordeiro

Federal University of Amazonas

SIDIA Instituto de Ciência e Tecnologia

University of Manchester