

# CS 6015: Software Engineering

Spring 2024

Lecture 13: Test automation

# Last Week

- Parsing cont. (Project related)
- MSDScript project overview
- Power of variables
- Libraries (Lab 5)

# This Week

- Test automation
- Design patterns

# Recall Testing

- Unit tests cover what you thought might go wrong
- **Drawback?**
  - Are we aware of different test combinations?
  - How to generate different strings from a pool of characters?
- **We still need that kind of testing- but we need to think of generating many different combinations of the input**

# Test generation

- **Test generation/automation**
  - Generating random inputs.
  - Results of the output are compared against expected output to verify that the test output passed or failed
- Generated tests can find what you didn't think about

# Test generation

Test generation needs:

- a way to generate inputs
- a driver to send the inputs and receive outputs
- a way to decide whether the output was good

*Test oracle*

# Test generation

Test generation needs:

- a way to generate inputs
- a driver to send the inputs and receive outputs
- a way to decide whether the output was good

Use another implementation  
*differential testing*

# Test generation

- Saves time
- Quick to find bug candidates: couple of minutes to perform a testing session.
- Ensures consistency and reduces human error
- Reduces variance in test quality from different individuals
- Run tests more frequently and anytime

# Test generation

## **Which tests should be automated**

- Test cases which are executed repeatedly
- Tests which are difficult to perform manually
- Time consuming tests

## **Tests not suitable for automation**

- Test cases which are faster to test manually rather than develop automated tests for them



# Testing: Example

<https://www.sqlite.org/testing.html>

- Four independently developed test harnesses
- 100% branch test coverage in an as-deployed configuration
- Millions and millions of test cases
- Out-of-memory tests
- I/O error tests
- Crash and power loss tests
- Fuzz tests
- Boundary value tests
- Disabled optimization tests
- Regression tests
- Malformed database tests
- Extensive use of assert() and run-time checks
- Valgrind analysis
- Undefined behavior checks
- Checklists

**The project has 590 times as  
much test code and test scripts !**

# Fuzzing

- or fuzz testing
- Type of test generation
- Provides random generated input to a program

**Fuzz testing**



**Program did not crash**

```
// Makes a string of up to 31 random bytes
static std::string random_bytes() {
    std::string word = "";
    for (int i = rand() % 32; i-- > 0; )
        word += rand() % 256;
    return word;
}
```

# Fuzzing

- or fuzz testing
- Type of test generation
- Provides random generated input to a program

```
// Makes a string of up to 31 random bytes
static std::string random_bytes() {
    std::string word = "";
    for (int i = rand() % 32; i-- > 0; )
        word += Returns a "random" int
    return word;
}
```

# Fuzzing


- or fuzz testing
- Type of test generation
- Provides random generated input to a program

```
// Makes a string of up to 31 random bytes
static std::string random_bytes() {
    std::string word = "";
    for (int i = rand() % 32; i-- > 0; )
        word += Use srand(clock()) to generate varying values
    return word;
}
```

# Fuzzing

- or fuzz testing
- Type of test generation
- Provides random generated input to a program

```
// Makes a string of up to 31 random bytes
static std::string random_bytes() {
    std::string word = "";
    for (int i = rand() % 32; i-- > 0; )
        word += rand();
    return word;
}
```



# Fuzzing

- or fuzz testing
- Type of test generation
- Provides random generated input to a program

```
// Makes a string of up to 31 random bytes
static std::string random_bytes() {
    std::string word = "";
    for (int i = rand() % 32; i-- > 0; )
        word += rand() % 256;
    return word;
}
```

*A number between 0 and 255*

# Fuzzing

- Fuzzing is a good idea for testing parsers, but just generating random strings is unlikely to generate many interesting MSDscript expressions

**Project related**



# Generating Expressions

```
<expr> = <number>
        | ( <expr> )
        | <expr> + <expr>
        | <expr> * <expr>
        | <variable>
        | _let <variable> = <expr> _in <expr>
```

Possible strategy:

- randomly pick a case
- for <number>, randomly pick one
- for <variable>, randomly generate one
- for others, recur for nested <expr>

# Generating Expressions

```
<expr> = <number>
| ( <expr> )
| <expr> + <expr>
| <expr> * <expr>
| <variable>
| _let <variable> = <expr> _in <expr>
```

Generate **Expr** values or strings?

By generating strings, we can make the test generator more separate from the code it's trying to test

# Simple Generator

First try — just generate numbers

```
std::string random_expr_string() {  
    return std::to_string(rand());  
}
```

Could check:

- `--interp` mode prints the same number
- `--print` mode prints the same number
- `--pretty-print` mode prints the same number
- exit code is always 0

# Simple Generator

```
std::string random_expr_string() {  
    if ((rand() % 10) < 6)  
        return std::to_string(rand());  
    else  
        return random_expr_string() + "+" + random_expr_string();  
}
```

# Simple Generator

```
std::string random_expr_string() {  
    if ((rand() % 10) < 6)  
        return std::to_string(rand());  
    else  
        return random_expr_string() + "+" + random_expr_string();  
}
```

60% of the time

# Simple Generator

```
std::string random_expr_string() {  
    if ((rand() % 10) < 6)  
        return std::to_string(rand());  
    else  
        return random_expr_string() + "+" + random_expr_string();  
}
```

60% of the time

40% of the time

Even without tracking the expected sum, but could check:

- **--interp** mode prints *some* number
- **--print** mode prints some expression that interps to the same number
- **--pretty-print** mode prints some expression that interps to the same number and pretty-prints exactly the same
- exit code is always 0

# Trying Generated Expressions

## Overall:

- Generate an expression string
- Send string as input to **msdscript**
- Check **msdscript** output and exit code

Inside the **msdscript** implementation:

- We take control of input and output using **std::istream&** and **std::ostream&** arguments

From the **outside**:

- We need a way to run a program
- Send it input to **std::cin**,
- and capture its output to **std::cout**

# Test-Runner Helper

Provided by **exec.cpp**:

```
ExecResult exec_program(int argc, char **argv, std::string in);
```

```
class ExecResult {  
public:  
    int exit_code;  
    std::string out;  
    std::string err;  
};
```



# Test-Runner Helper

Provided by **exec.cpp**:

```
ExecResult exec_program(int argc, char **argv, std::string in);
```

```
class ExecResult {  
public:  
    int exit_code;  
    std::string out;  
    std::string err;  
};
```

`argv[0]` is the program to run

one string as input

# Test-Runner Helper

Provided by **exec.cpp**:

```
ExecResult exec_program(int argc, char **argv, std::string in);
```

`argv[0]` is the program to run

one string as input

```
class ExecResult {
```

```
public:
```

```
    int exit_code;
```

```
    std::string out;
```

```
    std::string err;
```

```
};
```

two strings as output  
but either might be empty

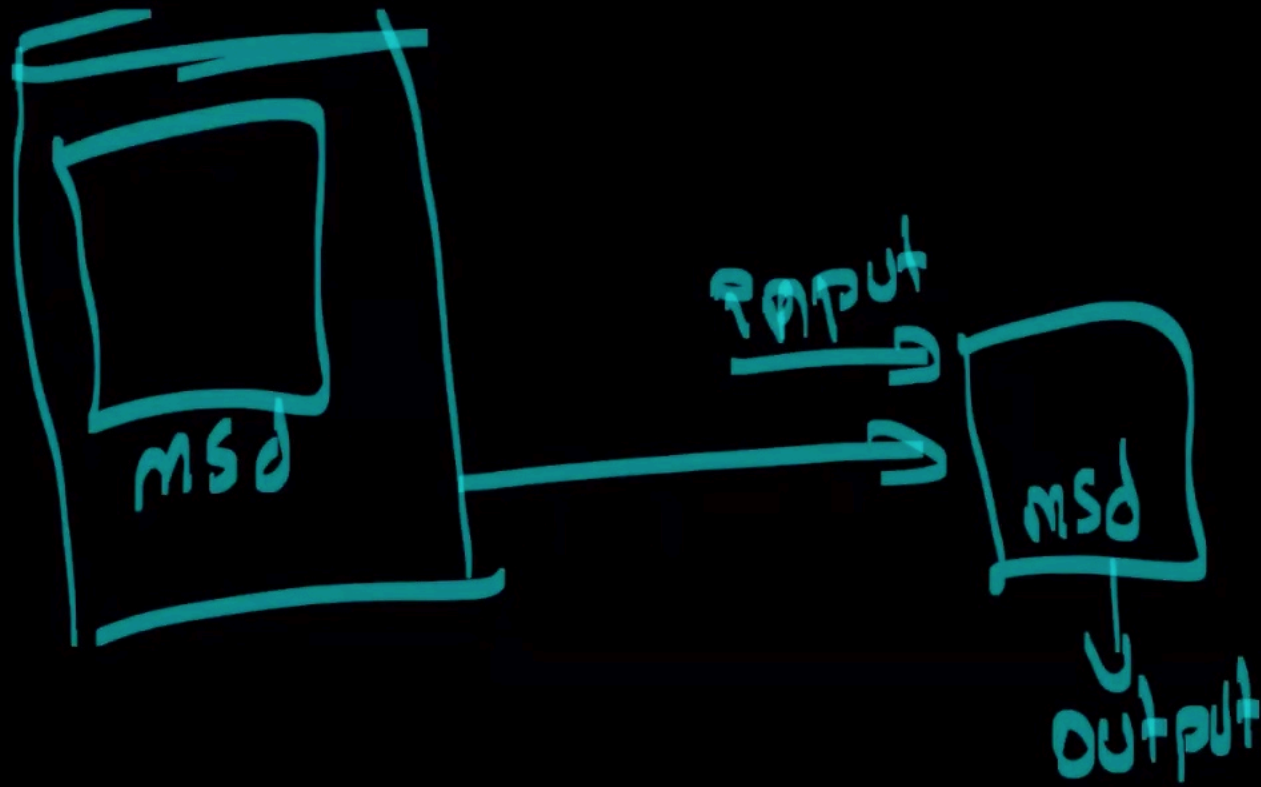
# Test-Runner Helper

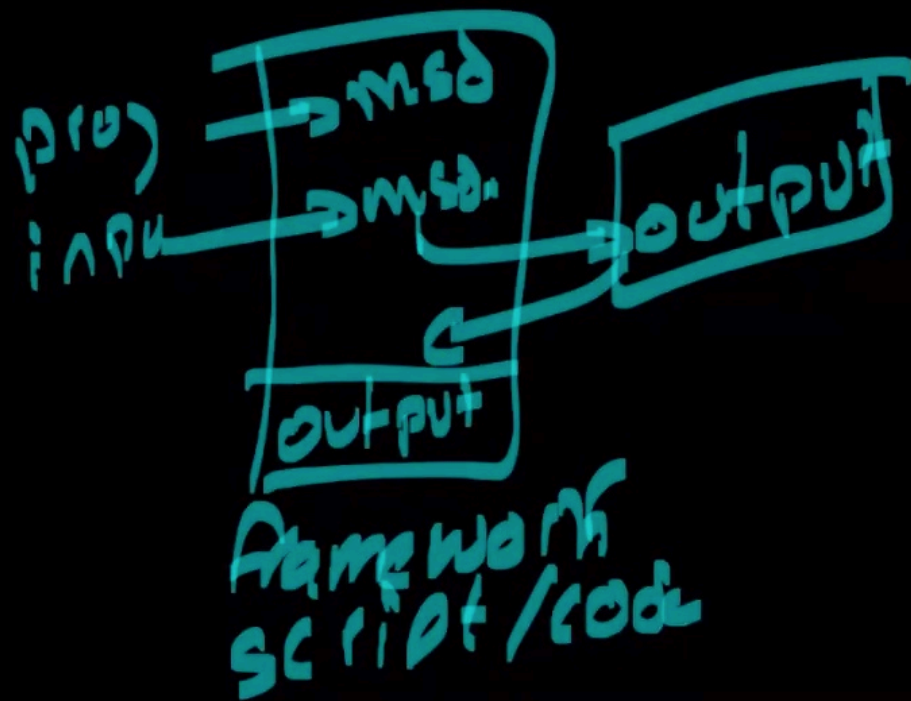
```
const char * const wc_argv[] = { "/usr/bin/wc", "-w" };

ExecResult wc_result = exec_program(2, wc_argv, "a b c");

if (wc_result.exit_code != 0)
    std::cerr << "non-zero exit: " << wc_result.exit_code << "\n";

if (wc_result.out != "      3\n")
    std::cerr << "bad wc result\n";
```







MSD CODE  
expr.cpp  
test.cpp

rstream

↓  
parser.cpp

↓  
generate  
using new.  
→ interp

↪ ostream.

```
nabil@nabil-macbookpro Lecture 13 Test generation % c++ exec.cpp e_run_wc.cpp -o
w
nabil@nabil-macbookpro Lecture 13 Test generation % ./w
nabil@nabil-macbookpro Lecture 13 Test generation % c++ exec.cpp e_run_wc.cpp -o
w
nabil@nabil-macbookpro Lecture 13 Test generation % ./w
bad wc result
nabil@nabil-macbookpro Lecture 13 Test generation % cd msdscript
nabil@nabil-macbookpro msdscript % c++ run_msdscript2.cpp exec.cpp -o prog1
nabil@nabil-macbookpro msdscript % ./prog1
```

# Another Simple Test Driver

```
int main(int argc, char **argv) {
    const char * const interp_argv[] = { "msdscript", "--interp" };
    const char * const print_argv[] = { "msdscript", "--print" };

    for (int i = 0; i < 100; i++) {
        std::string in = random_expr_string();
        std::cout << "Trying" << in << "\n";

        ExecResult interp_result = exec_program(2, interp_argv, in);
        ExecResult print_result = exec_program(2, print_argv, in);

        ExecResult interp_again_result = exec_program(2, interp_argv, print_result.out);
        if (interp_again_result.out != interp_result.out)
            throw std::runtime_error("different result for printed");
    }

    return 0;
}
```



# Simple Test Driver

```
int main(int argc, char **argv) {
    const char * const interp1_argv[] = { "msdscript", "--interp" };
    const char * const interp2_argv[] = { "msdscript2", "--interp" };

    for (int i = 0; i < 100; i++) {
        std::string in = random_expr_string();
        std::cout << "Trying " << in << "\n";

        ExecResult interp1_result = exec_program(2, interp1_argv, in);
        ExecResult interp2_result = exec_program(2, interp2_argv, in);

        if (interp1_result.out != interp2_result.out)
            throw std::runtime_error("different results");
    }

    return 0;
}
```

# Project

**Test generation for the msdscript project does  
not use any of the msdscript implementation**

