CMPT 473
Software Quality Assurance

Regression Testing

Nick Sumner

The Story So Far

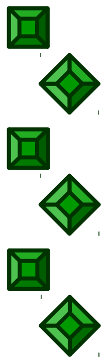
- We have seen how to measure the quality of software

The Story So Far

- We have seen how to measure the quality of software (and even improve it a bit)

The Story So Far

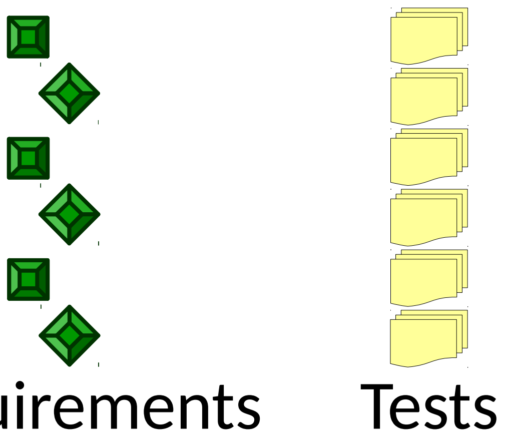
- We have seen how to measure the quality of software
  - Establish quality requirements



Requirements

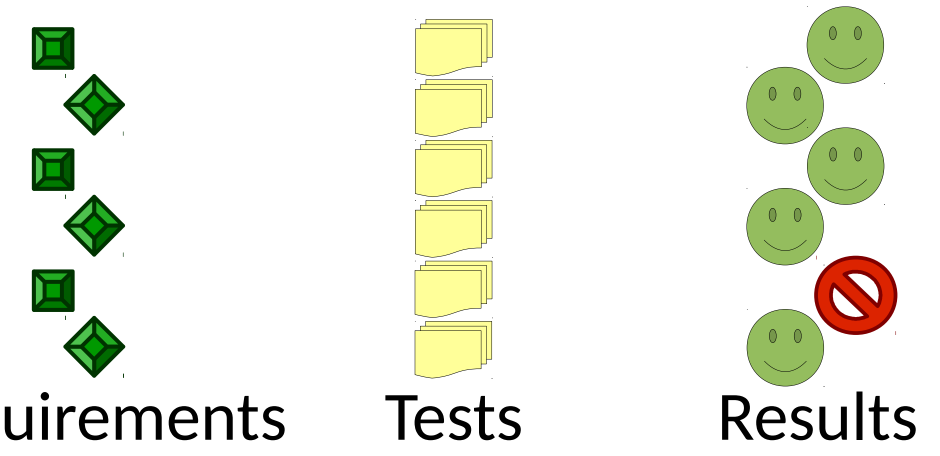
The Story So Far

- We have see how to measure the quality of software
  - Establish quality requirements
  - Build a test suite



The Story So Far

- We have seen how to measure the quality of software
  - Establish quality requirements
  - Build a test suite
  - Run it to identify missed requirements



The Story So Far

- We have seen how to measure the quality of software
  - Establish quality requirements
  - Build a test suite
  - Run it to identify missed requirements
- Are the quality requirements in real software static/ fixed?

The Story So Far

- We have seen how to measure the quality of software
  - Establish quality requirements
  - Build a test suite
  - Run it to identify missed requirements
- Are the quality requirements in real software static/ fixed?
- Software evolves

The Story So Far

- We have seen how to measure the quality of software
  - Establish quality requirements
  - Build a test suite
  - Run it to identify missed requirements
- Are the quality requirements in real software static/ fixed?
- Software evolves
  - The testing process should support and facilitate change

Regression Testing

- *Regression Testing*

What is it?

Regression Testing

- *Regression Testing*
  - Retesting software as it evolves to ensure previous functionality

Regression Testing

- *Regression Testing*
  - Retesting software as it evolves to ensure previous functionality
- Useful as a tool for *ratcheting* software quality

Regression Testing

- *Regression Testing*
  - Retesting software as it evolves to ensure previous functionality
- Useful as a tool for *ratcheting* software quality

What is a ratchet?

Regression Testing

- *Regression Testing*
  - Retesting software as it evolves to ensure previous functionality
- Useful as a tool for *ratcheting* software quality

What is a ratchet?

Regression Testing

- *Regression Testing*
  - Retesting software as it evolves to ensure previous functionality
- Useful as a tool for *ratcheting* software quality
- Regression tests further enable making changes

Why Use Regression Testing

- As software evolves, previously working functionality can fail.

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.

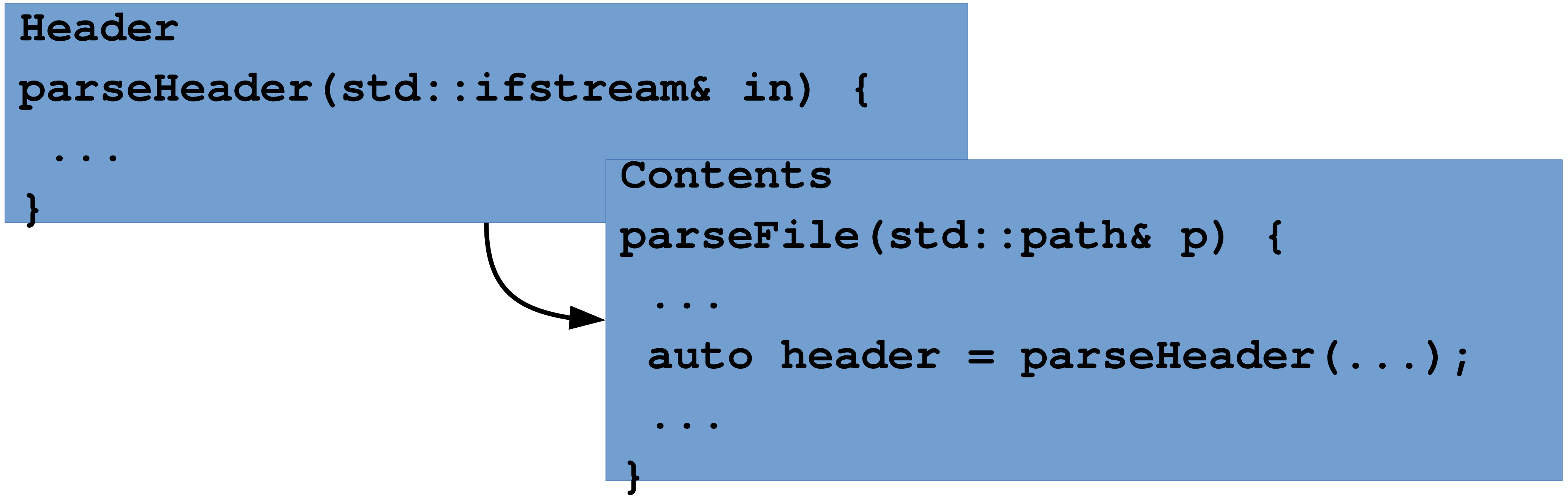
Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.

```
Contents
parseFile(std::path& p) {
  ...
  auto header = parseHeader(...);
  ...
}
```

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.

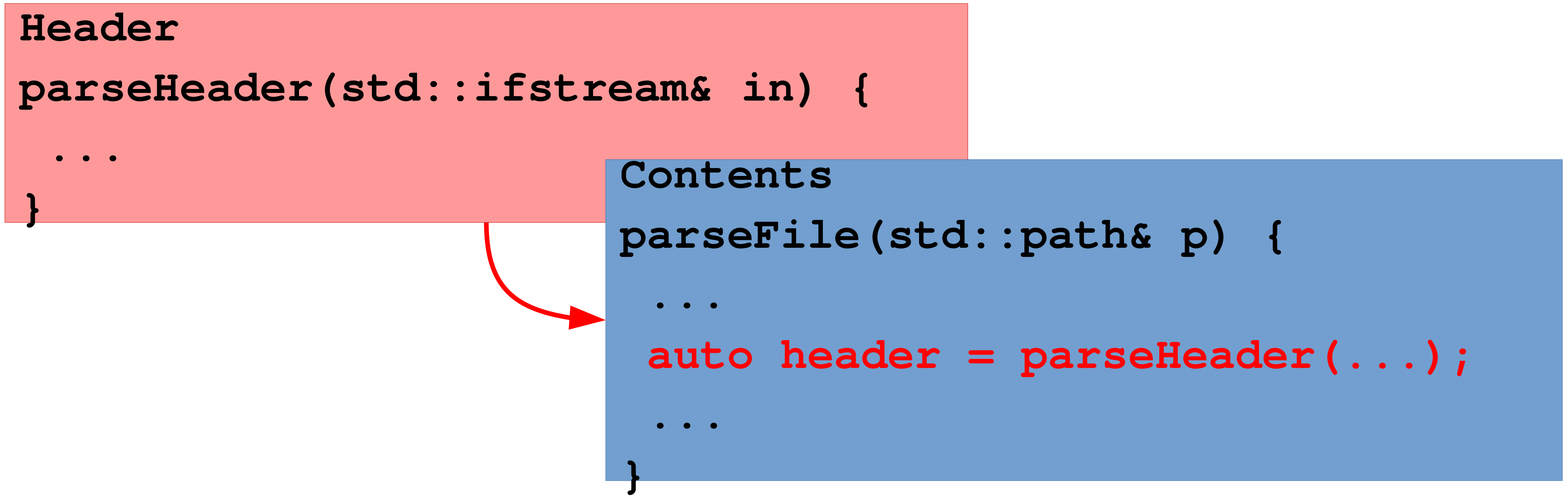


```
Header
parseHeader(std::ifstream& in) {
  ...
}
```

```
Contents
parseFile(std::path& p) {
  ...
  auto header = parseHeader(...);
  ...
}
```

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.



```
Header
parseHeader(std::ifstream& in) {
  ...
}
```

```
Contents
parseFile(std::path& p) {
  ...
  auto header = parseHeader(...);
  ...
}
```

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.
  - New environments can introduce unexpected behavior in components that originally work.

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.
  - New environments can introduce unexpected behavior in components that originally work.
- **Most testing is regression testing**

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.
  - New environments can introduce unexpected behavior in components that originally work.
- **Most testing is regression testing**
- Ensuring previous functionality can require large test suites. Are they always realistic?

Why Use Regression Testing

- As software evolves, previously working functionality can fail
  - Software is complex & interconnected.
  - Changing one component can unintentionally impact another.
  - New environments can introduce unexpected behavior in components that originally work.
- **Most testing is regression testing**
- Ensuring previous functionality can require large test suites. Are they always realistic?

How often did you run regression tests in co-ops/internships?

What Is A Regression Test Suite

- Three common components for regression suites:

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs
    - Some components are bug prone
    - Helps to identify when previous fixes were inadequate.

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs
    - Some components are bug prone
    - Helps to identify when previous fixes were inadequate.
  - Unit tests

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs
    - Some components are bug prone
    - Helps to identify when previous fixes were inadequate.
  - Unit tests
    - Especially useful for refactoring

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs
    - Some components are bug prone
    - Helps to identify when previous fixes were inadequate.
  - Unit tests
    - Especially useful for refactoring
  - General system tests

What Is A Regression Test Suite

- Three common components for regression suites:
  - Tests for previously fixed bugs
    - Some components are bug prone
    - Helps to identify when previous fixes were inadequate.
  - Unit tests
    - Especially useful for refactoring
  - General system tests
- Regression tests are usually a selected subset of tests generated for other purposes.

Regression Testing In Practice

- Too many & too frequent to do by hand
  - Automate it:

    e.g. JUnit suites, commit hooks, nightlies

Regression Testing In Practice

- Too many & too frequent to do by hand
  - Automate it:

    e.g. JUnit suites, commit hooks, nightlies
- Over time, regression suites grow even larger
  - Cannot run every time you commit
  - Cannot run every night

Regression Testing In Practice

- Too many & too frequent to do by hand
  - Automate it:

    e.g. JUnit suites, commit hooks, nightlies
- Over time, regression suites grow even larger
  - Cannot run every time you commit
  - Cannot run every night
- Can grow the test bed as well, but that costs $ as well…

Regression Testing In Practice

- Too many & too frequent to do by hand
  - Automate it:
    e.g. JUnit suites, commit hooks, nightlies
- Over time, regression suites grow even larger
  - Cannot run every time you commit
  - Cannot run every night
- Can grow the test bed as well, but that costs $ as well...

How else can we address this problem?

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests

But this is more or less where we started…

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests
- Sometimes not all tests need to run with each commit

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests
- Sometimes not all tests need to run with each commit
  - Run a subset of sanity or *smoke tests* for commits

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests
- Sometimes not all tests need to run with each commit
  - Run a subset of sanity or *smoke tests* for commits

These mostly validate the build process & core behaviors.

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests
- Sometimes not all tests need to run with each commit
  - Run a subset of sanity or *smoke tests* for commits
  - Run more thorough tests nightly

Limiting Regression Suites

- Be careful not to add redundant test to the test suite.
  - Every bug may indicate a useful behavior to test
  - Test adequacy criteria can limit the other tests
- Sometimes not all tests need to run with each commit
  - Run a subset of sanity or *smoke tests* for commits
  - Run more thorough tests nightly
  - "   " weekly
  - "   " preparing for milestones/ integration

Limiting Regression Testing

- Can we be smarter about which test we run & when?

What else could we do?

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software
- Can decide which tests to run on demand

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software
- Can decide which tests to run on demand
  - **Conservative**: run all tests
  - **Cheap**: run tests with test requirements related to the changed lines

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software
- Can decide which tests to run on demand
  - **Conservative**: run all tests
  - **Cheap**: run tests with test requirements related to the changed lines

Is the cheap approach enough?

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software
- Can decide which tests to run on demand
  - **Conservative**: run all tests
  - **Cheap**: run tests with test requirements related to the changed lines
  - **Middle ground**: Run those tests affected by how changed propagate through the software?

Limiting Regression Testing

- Can we be smarter about which test we run & when?
- Change Impact Analysis
  - Identify how changes affect the rest of software
- Can decide which tests to run on demand
  - **Conservative**: run all tests
  - **Cheap**: run tests with test requirements related to th
  - **Middle ground**: Run those tests affected by how changed propagate through the software?

In practice, tools can assist in finding out which tests need to be run

Failure

- Eventually, tests will fail. What do you do?

Failure

- Eventually, tests will fail. What do you do?

Honestly. What do you do?
We are no longer *measuring* quality.

Failure

- Eventually, tests will fail. What do you do?
  - It depends…

Failure

- Eventually, tests will fail. What do you do?
  - It depends…
- If the new and old versions should be equivalent:

Why might this happen?

Failure

- Eventually, tests will fail. What do you do?
  - It depends…
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct

Failure

- Eventually, tests will fail. What do you do?
  - It depends...
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct

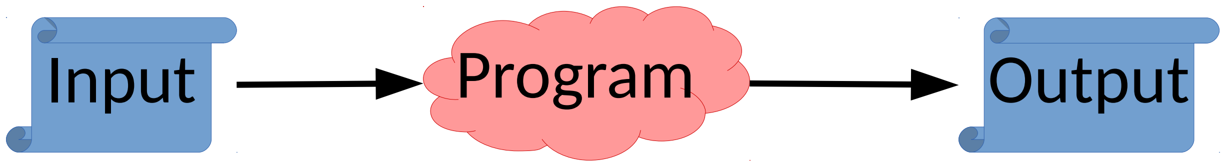
This yields the ratcheting power of regression tests!

Failure

- Eventually, tests will fail. What do you do?
  - It depends…
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise:

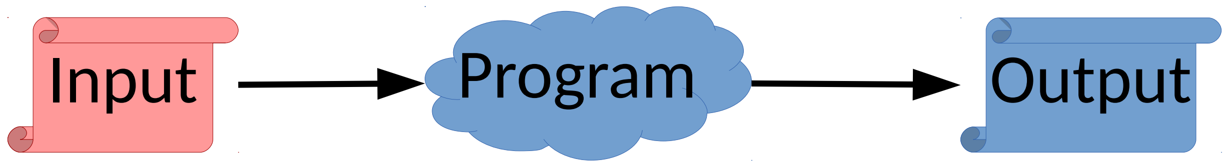
Failure

- Eventually, tests will fail. What do you do?
  - It depends...
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise: (at least one of)
  - The software has a bug to fix



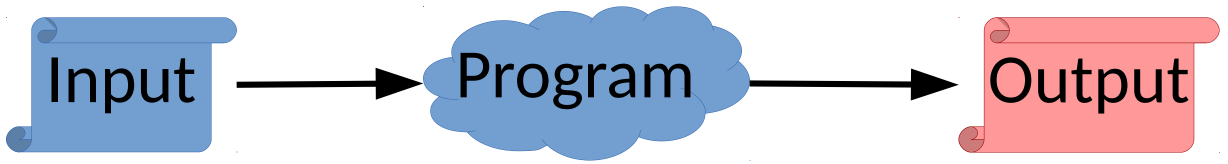
Failure

- Eventually, tests will fail. What do you do?
  - It depends...
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise: (at least one of)
  - The software has a bug to fix
  - Test inputs are stale and must be fixed



Failure

- Eventually, tests will fail. What do you do?
  - It depends...
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise: (at least one of)
  - The software has a bug to fix
  - Test inputs are stale and must be fixed
  - The expected behavior has changed & must be fixed



Failure

- Eventually, tests will fail. What do you do?
  - It depends...
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise:
  - The software has a bug to fix
  - Tes
  - The ust be fixed

Keeping these cases separate is important.
How can we do that?

Failure

- Eventually, tests will fail. What do you do?
  - It depends…
- If the new and old versions should be equivalent:
  - A failing tests indicates misbehavior to correct
- Otherwise:
  - The software has a bug to fix
  - Test inputs are stale and must be fixed
  - The expected behavior has changed & must be fixed
- Maintaining regression tests is ***costly***

Burdens

Burdens of scale

Burdens

Burdens of scale

- Running the tests

Burdens

Burdens of scale

- Running the tests
- Interpreting the results

Burdens

Burdens of scale

- Running the tests
- Interpreting the results
- Updating tests

Burdens

Burdens of scale

- Running the tests
- Interpreting the results
- Updating tests
- Adding new tests

Burdens

Burdens of scale

- Running the tests
- Interpreting the results
- Updating tests
- Adding new tests

Addressing these burdens is a major focus of *automated testing* and *testability*

Summary

- Regression testing retests software to ensure previous functionality.

Summary

- Regression testing retests software to ensure previous functionality.
- It increases the confidence of refactoring & supports ratcheting software quality

Summary

- Regression testing retests software to ensure previous functionality.
- It increases the confidence of refactoring & supports ratcheting software quality
- The major trade-off comes from the scale of the regression test suite.

Summary

- Regression testing retests software to ensure previous functionality.
- It increases the confidence of refactoring & supports ratcheting software quality
- The major trade-off comes from the scale of the regression test suite.
  - Judgment on making trade offs for regression testing are important for lowering costs

Summary

- Regression testing retests software to ensure previous functionality.
- It increases the confidence of refactoring & supports ratcheting software quality
- The major trade-off comes from the scale of the regression test suite.
  - Judgment on making trade offs for regression testing are important for lowering costs
  - You may remove tests from the regression suite over time

Summary

- Regression testing retests software to ensure previous functionality.
- It increases the confidence of refactoring & supports ratcheting software quality
- The major trade-off comes from the scale of the regression test suite.
  - Judgment on making trade offs for regression testing are important for lowering costs
  - You may remove tests from the regression suite over time

We may also look at techniques for generalizing unit tests to find new bugs...