How testable is Business Software?

Peter Schrammel

UNIVERSITY OF SUSSEX

diffblue
AI for Code

What is this talk about?

Desire for software that has fewer bugs

- Focus on non-safety-critical software
- Great techniques developed by formal methods, programming languages and software engineering communities
- Make developers' lives easier

Typical challenges in automated verification hampering adoption:

- State space explosion, scalability, ...

But there are other issues too...

Business-Critical Software

business critical

Sizeable software stack:

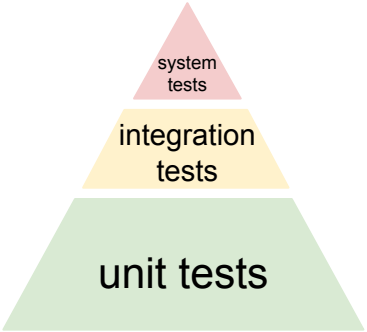
- Critical to perform daily operations
- Defects impact revenue, customer satisfation, ...

Pressures:

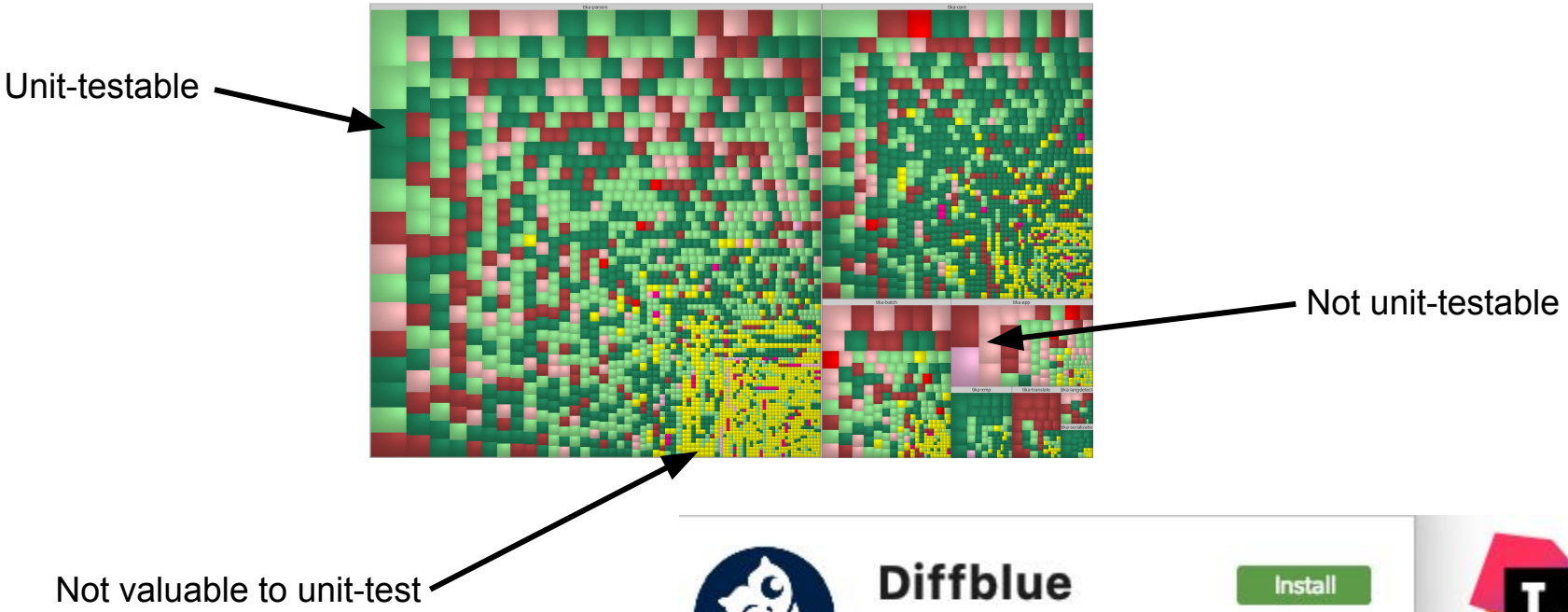
- Faster, cheaper, …
- Detect defects early (“shift left”)

Testing:

- Slow tests unsuitable for CI/CD
- Fast unit tests that can run early in the development cycle



How unit-testable is the code base?



What to expect from this talk?

How to map out testability of a code base

Similar issues apply to automated verification

Testability deficiencies are a signifcant issue

Support for overcoming these issues has high impact

(Binder 1994)

What is a unit test?

```
public class ProductTest {
  @Test
  public void testSend() {
    // Arrange the inputs and mocks
    Product product = new Product();
    product.addExpiryDate();
    // Act: call the method under test (MUT)
    boolean isExpired = product.isExpired();
    // Assert on the effects
    assertTrue(isExpired);
  }
}
```

Desirable properties:

- Runs fast (a few ms)
- Has no side effects on other tests

diffblue
AI for Code

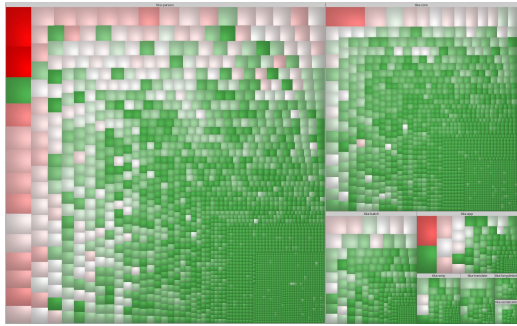
What is a unit test?

```
@Test
public void testPropertyMappingGlobalOverride() throws Exception {
  String propertyPrefix = AbstractMappingMetadataExtracter.PROPERTY_PREFIX_METADATA +
    DummyMappingMetadataExtracter.EXTRACTER_NAME +
    AbstractMappingMetadataExtracter.PROPERTY_COMPONENT_EXTRACT;

  ApplicationContext ctx = MiscContextTestSuite.getMinimalContext();
  Properties globalProperties = (Properties) ctx.getBean("global-properties");
  globalProperties.setProperty(
    propertyPrefix + "namespace.prefix.my",
    DummyMappingMetadataExtracter.NAMESPACE_MY);
  globalProperties.setProperty(
    propertyPrefix + DummyMappingMetadataExtracter.PROP_A,
    " my:a1, my:a2, my:c ");

  extracter.setApplicationContext(ctx);

  extracter.register();
  // Only mapped 'a'
  destination.clear();
  extracter.extract(reader, destination);

  assertEquals(
    DummyMappingMetadataExtracter.VALUE_A,
    destination.get(DummyMappingMetadataExtracter.QNAME_C));
}
```

https://github.com/Alfresco/alfresco-repository

diffblue
AI for Code

Not all code is equally critical



Goal: have tests for critical code

Critical code vs non-critical code

- Cyclomatic complexity (McCabe 1976) often used as a proxy

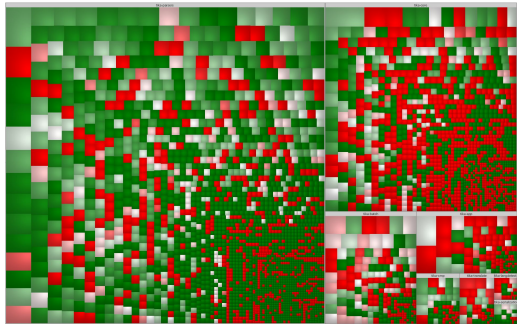
Unit-testable vs non-unit-testable code

- Testability analysis

this talk

Covered vs not-covered code

- Unit vs integration vs system test
- Test suite adequacy: coverage, mutation score



What does testability mean?

“If modularity is controlled so that the function of a module is **independent** of the source of its **input**, the destination of its **output**, and the **past history** of use of the module, the **difficulty of testing** the modules and structures assembled from the modules is greatly reduced.” *Nate Edwards, 1975*

What does testability mean?

“If modularity is controlled so that the function of a module is **independent** of the source of its **input**, the destination of its **output**, and the **past history** of use of the module, the **difficulty of testing** the modules and structures assembled from the modules is greatly reduced.” *Nate Edwards, 1975*

“The concept of [...] **testability of software** is defined by applying the concepts of **observability** and **controllability** to software. It is shown that a [...] testable program does not exhibit any **input-output inconsistencies** and supports **small test sets** in which test **outputs** are easily understood. **Metrics** that can be used to assess the **level of effort** required in order to **modify** a program so that it is [...] testable [...].” *Roy Freedman, 1991*

What does testability mean?

“Testability has two key facets: *controllability* and *observability*. To test a component, you must be able to **control its input** (and internal state) and **observe its output**. If you cannot control the input, you cannot be sure what has caused a given output. If you cannot observe the output of a component under test, you cannot be sure how a given input has been processed.”

Robert V. Binder, 1994

What does testability mean?

Controllability

- Control system: “Can steer into any desired state”
- Software:
  - Ability to **arrange** inputs of MUT to exercise a code path
  - Ability to control the effects of dependent components (mockability)
- Why not controllable? Non-determinism, unreachable code

Observability

- Control system: “State can be determined from the outputs”
- Software:
  - Ability to **assert** on relevant effects of the MUT
- Why not observable? Lack of accessibility and mockability

Mockability

Ability to inject objects that must be mocked in order to **control** and observe their interactions

```
public class Product {
  private LocalDateTime expiryDate;

  public void addExpiryDate() {
   this.expiryDate = LocalDateTime.now()
     .plus(30, DAYS);
  }
  public boolean isExpired() {
    return this.expiryDate
      .isBefore(LocalDateTime.now());
  }

}
```

```
public class ProductTest {
  @Test public void testSend() {
    // Arrange
    Product product = new Product();
    product.addExpiryDate();

    // Act & Assert
    assertTrue(product.isExpired());
  }
}
```

diffblue
AI for Code

Mockability

Ability to inject objects that must be mocked in order to **control** and observe their interactions

```
public class Product {
  private LocalDateTime expiryDate;

  public void addExpiryDate() {
   this.expiryDate = LocalDateTime.now()
     .plus(30, DAYS);
  }
  public boolean isExpired() {
    return this.expiryDate
      .isBefore(LocalDateTime.now());
  }

}
```

```
public class appTest {
  @Test public void testSend() {
    // Arrange
    Product product = new Product();
    product.addExpiryDate();
    Thread.sleep(31*24*3600);
    // Act & Assert
    assertTrue(product.isExpired());
  }
}
```

diffblue
AI for Code

Mockability

Ability to inject objects that must be mocked in order to **control** and observe their interactions

```
public class Product {
  private LocalDateTime expiryDate;
  private Clock clock = Clock.systemUTC();
  public void addExpiryDate() {
    this.expiryDate = LocalDateTime.now(clock)
      .plus(30, DAYS);
  }
  public boolean isExpired() {
    return this.expiryDate
      .isBefore(LocalDateTime.now(clock));
  }
  void setClock(Clock clock) { this.clock = clock;
}
}
```

Dependency injection

```
public class ProductTest {
  @Test public void testExpired() {
    // Arrange
    Product product = new Product();
    product.setClock(Clock.fixed(Instant.EPOCH));
    product.addExpiryDate();
    product.setClock(Clock.fixed(
      Instant.EPOCH.plus(31, DAYS)));
    // Act & Assert
    assertTrue(product.isExpired());
  }
}
```

diffblue
AI for Code

Mockability

Ability to inject objects that must be mocked in order to control and **observe** their interactions

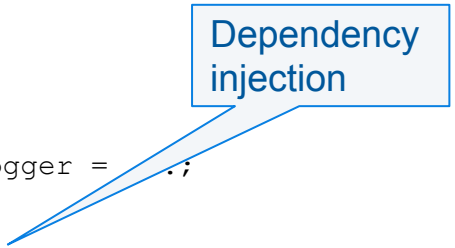
```
public class App {
  private static final logger = ...;
  private Client client;
  public App() {
    this.client = new Client();
  }
  public void send(Message m) {
    try {
      client.call(m);
    } catch (Exception e) {
      logger.error("send failed", e);
    }
  }
}
```

```
public class AppTest {
  @Test public void testSend() {
    // Arrange
    App app = new App();
    Message message = new Message("hello");
    // Act
    app.send(message);
    // Assert
    ???
  }
}
```

diffblue
AI for Code

Mockability

Ability to inject objects that must be mocked in order to control and **observe** their interactions




```
public class App {
  private static final logger =      .;
  private Client client;
  public App(Client client) {
    this.client = client;
  }
  public void send(Message m) {
    try {
      client.call(m);
    } catch (Exception e) {
      logger.error("send failed", e);
    }
  }
}
```

```
public class AppTest {
  @Test public void testSend() {
    // Arrange
    Client client = new Client();
    App app = new App(client);
    Message message =  new Message("hello");
    // Act
    app.send(message);
    // Assert
    assert(client...) ???
  }
}
```

diffblue
AI for Code

Mockability

Ability to inject objects that must be mocked in order to control and **observe** their interactions

```
public class App {
  private static final logger = ...;
  private Client client;
  public App(Client client) {
    this.client = client;
  }
  public void send(Message m) {
    try {
      client.call(m);
    } catch (Exception e) {
      logger.error("send failed", e);
    }
  }
}
```

```
public class AppTest {
  @Test public void testSend() {
    // Arrange
    Client client = mock(Client.class);
    App app = new app(client);
    Message message = new Message("hello");
    // Act
    app.send(message);
    // Assert
    verify(client).send(message);
  }
}
```

diffblue
AI for Code

Mockability

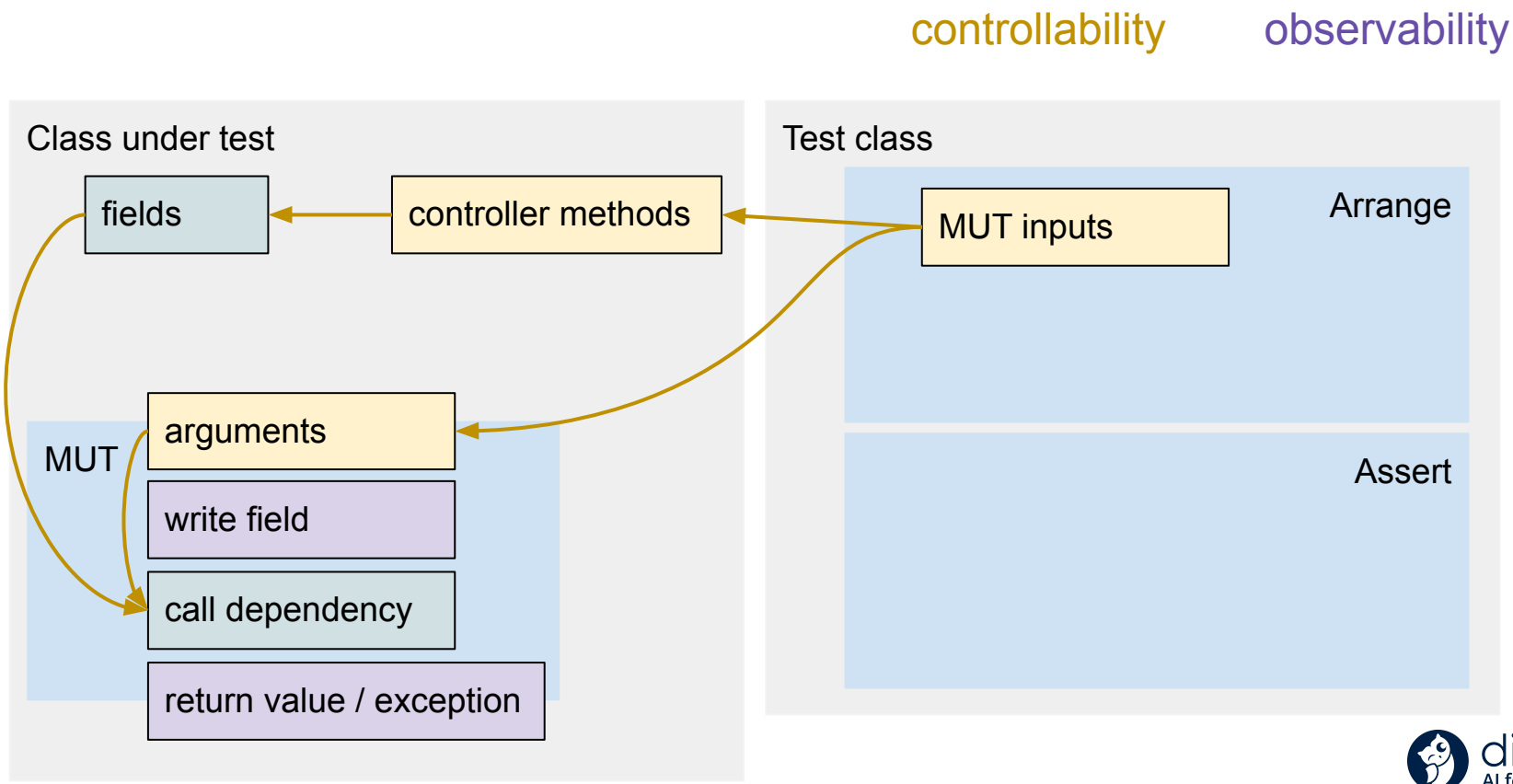
Ability to inject objects that must be mocked in order to **control** and **observe** their interactions

```
public class App {
  private static final logger = ...;
  private Client client;
  public App(Client client) {
    this.client = client;
  }
  public void send(Message m) {
    try {
      client.call(m);
    } catch (Exception e) {
      logger.error("send failed", e);
    }
  }
}
```

```
public class appTest {
  @Test public void testSendFailed() {
    // Arrange
    Client client = mock(Client.class);
    when(client.send(any())).thenThrow( new Exception());
    App app = new App(client);
    Message message =  new Message("hello");
    // Act
    app.send(message);
    // Assert
    assertThrows(Exception. class,  () -> app.send(message));
    verify(client).send(message);
  }
}
```

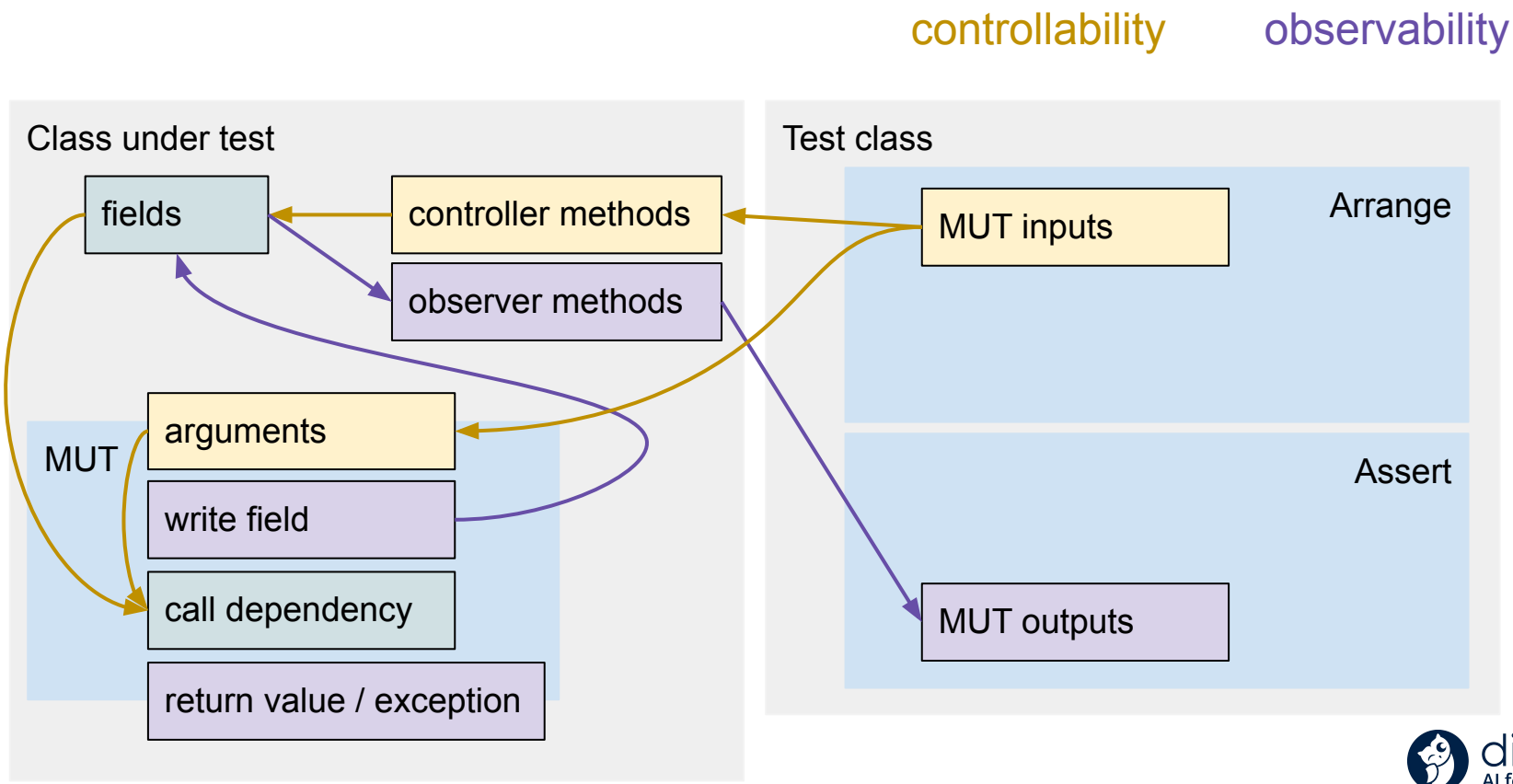
diffblue
AI for Code

Testability



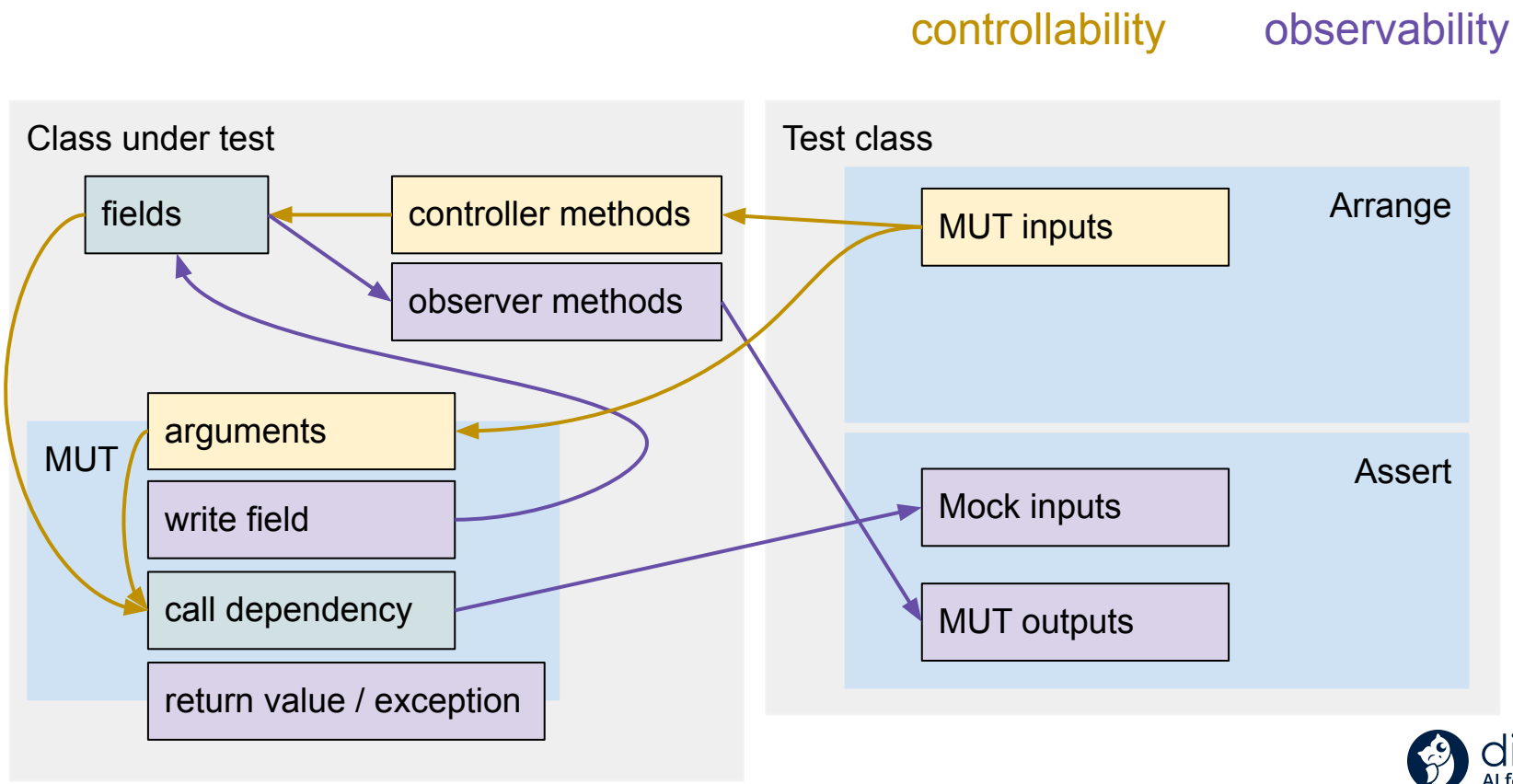
diffblue
AI for Code

Testability



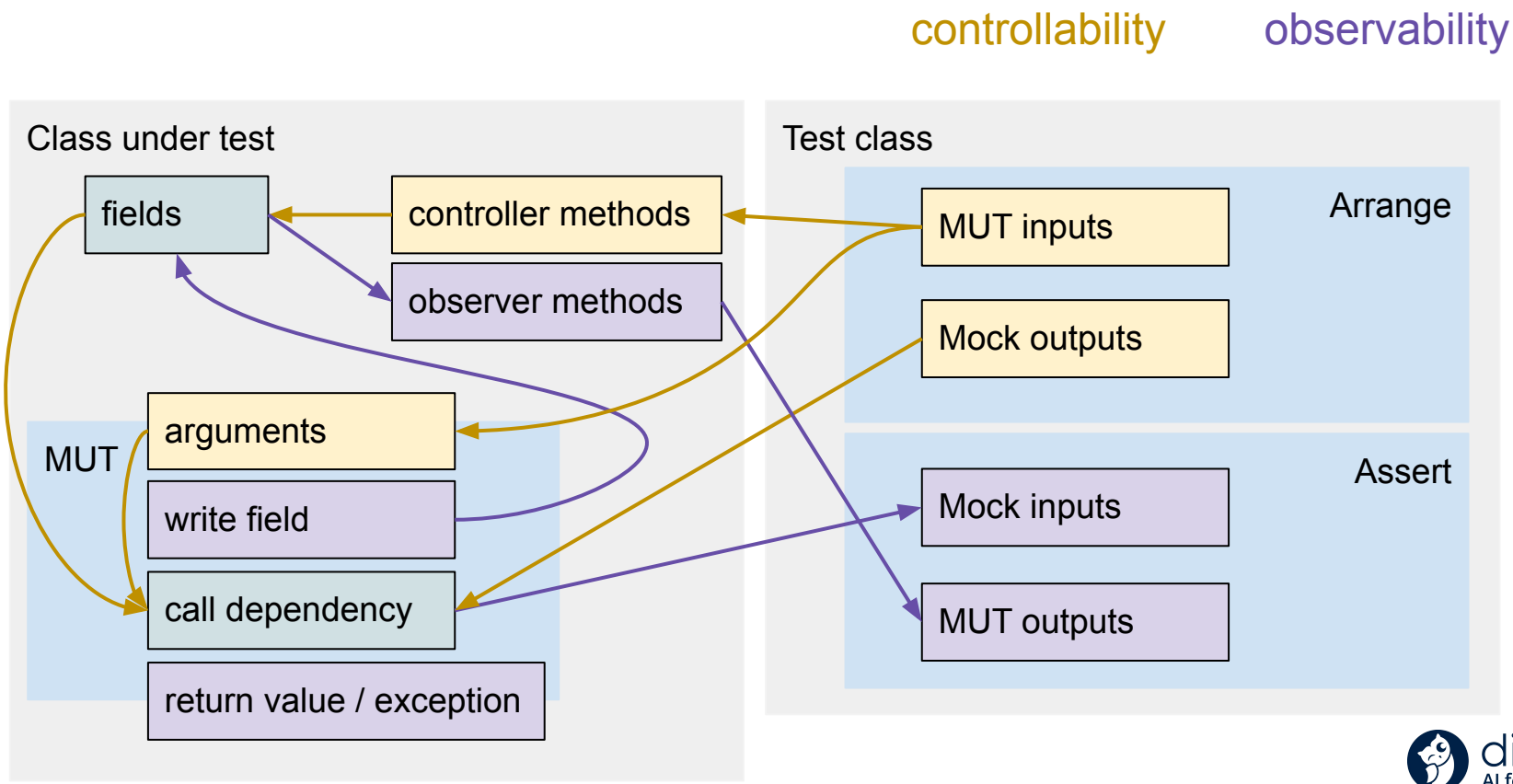
diffblue
AI for Code

Testability



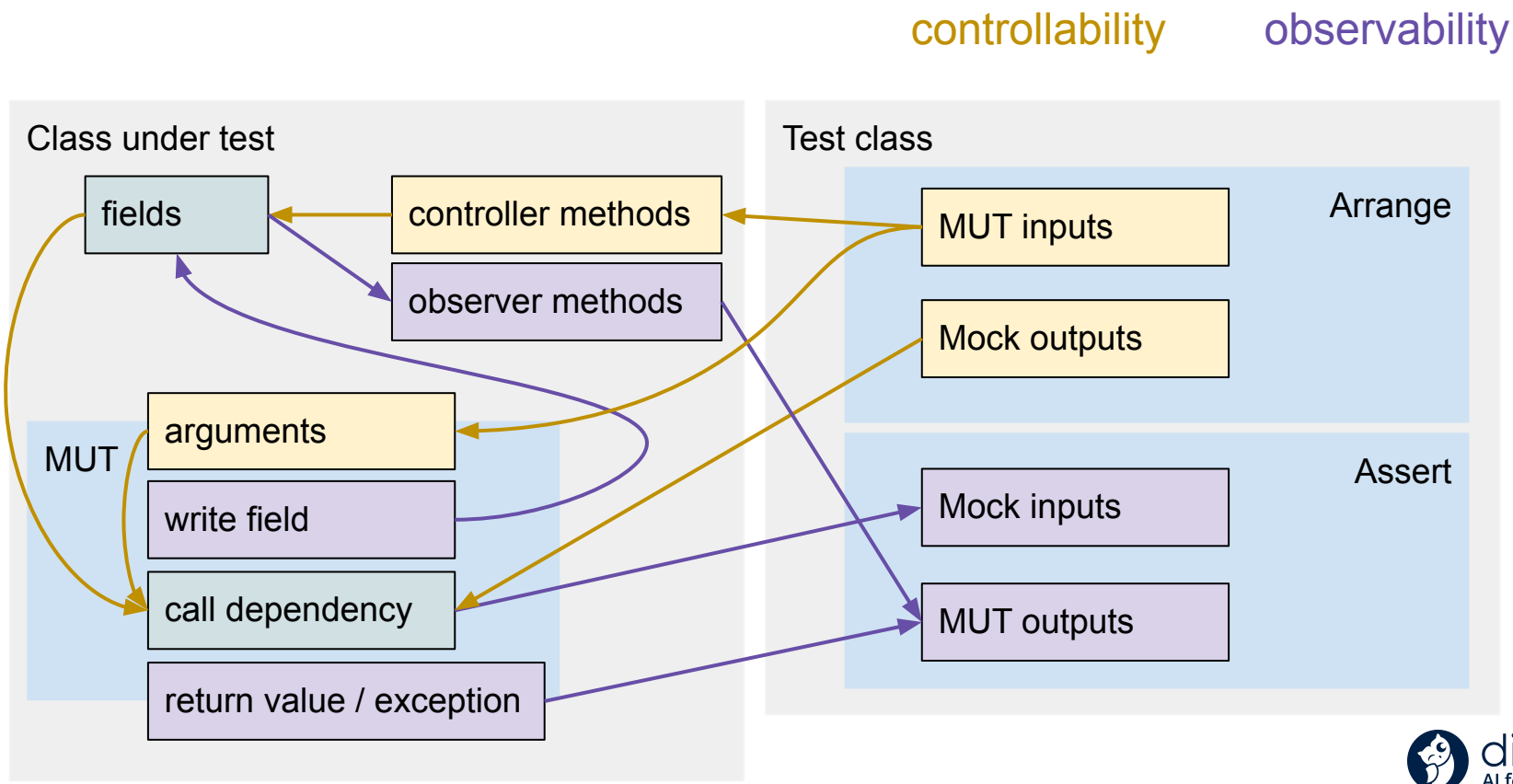
diffblue
AI for Code

Testability



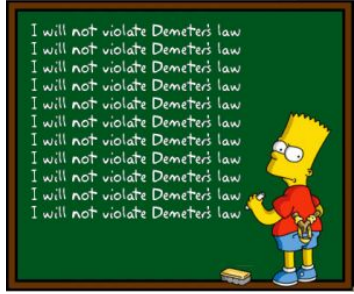
diffblue
AI for Code

Testability



diffblue
AI for Code

Testability Metrics



Try to find correlations between

- software quality metrics (coupling, number of fields, complexity of methods, etc)
- and difficulty / effort to write tests (e.g. Terragni et al 2020)
- Give quantitative predictions

Our goal:

- Give precise diagnostic information
- Explain for each method where and what the problem is
- Assist in fixing it, potentially fix it automatically
- Our test generation tool will perform better

diffblue
AI for Code

How unit-testable is business software?

Static analysis:

- On the byte code (.class files)
- Under-approximate “not valuable to unit-test” and “not unit-testable”

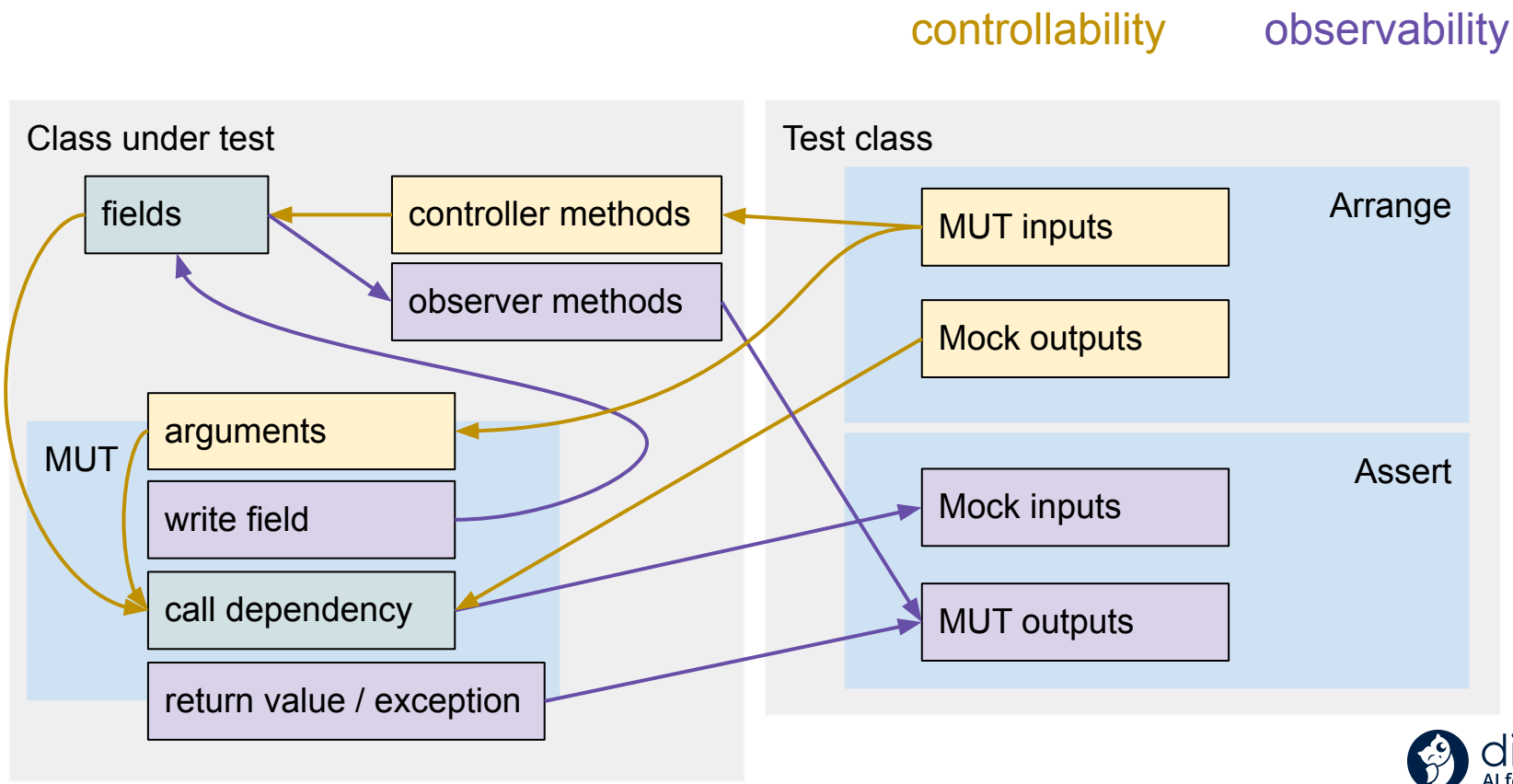
Analysis of Java software packages:

- 40 repositories with 442 modules
- 8.2 MLOC Java, 98k classes (with dependencies much more)

Various areas:

- Business workflows, data processing, distributed computing, data storage

Testability



diffblue
AI for Code

Mockability Analysis

We under-approximate the set of non-mockable methods.

A method is **non-mockable** if

- It **must be mocked** (because it is non-deterministic), or
- It has a call to a **non-mockable** static method, or
- It has a call to a **non-mockable** instance method on an object that is **non-injectable**

An object is **non-injectable** if

- It cannot be supplied through inputs

How unit-testable is business software?

On average:

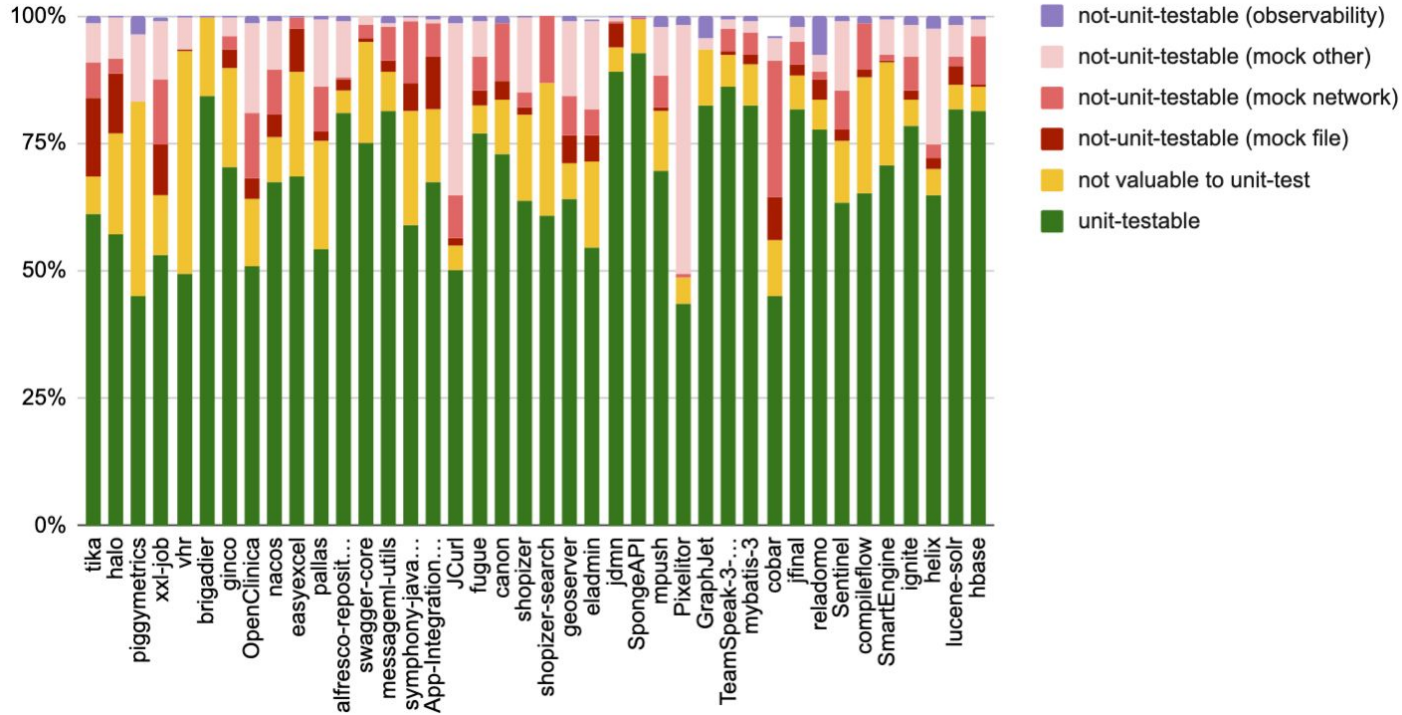
21% not unit-testable

6% not valuable to unit-test

73% unit-testable

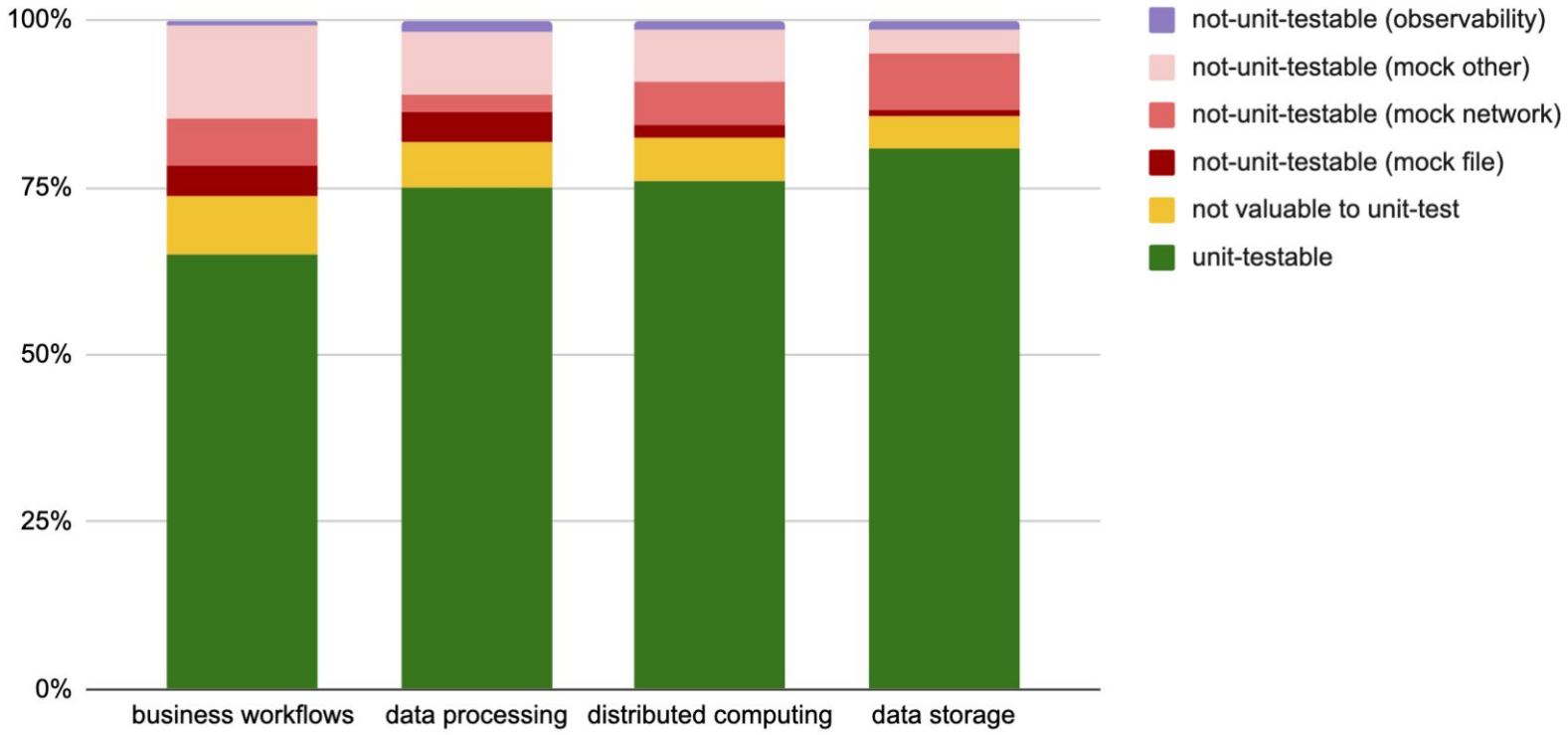
Very high variability

on module level (0-100%)



diffblue
AI for Code

How unit-testable is business software?



diffblue
AI for Code

Diagnostic Information

```
public class MailServiceImpl implements MailService {
  ...
  public void testConnection() {
    JavaMailSender javaMailSender = getMailSender();
    if (javaMailSender instanceof JavaMailSenderImpl) {
      JavaMailSenderImpl mailSender = (JavaMailSenderImpl) javaMailSender;
      try {
        mailSender.testConnection();
      } catch (MessagingException e) {
        throw new EmailException("无法连接到邮箱服务器，请检查邮箱配置.[" +
          e.getMessage() + "]", e);
      }
    }
  }
  private JavaMailSender getMailSender() {
    ...
  }
}
```

https://github.com/halo-dev/halo

diffblue
AI for Code

Diagnostic Information

```
public class MailServiceImpl implements MailService {
  ...
  public void testConnection() {
    JavaMailSender javaMailSender = getMailSender();
    if (javaMailSender instanceof JavaMailSenderImpl) {
      JavaMailSenderImpl mailSender = (JavaMailSenderImpl) javaMailSender;
      try {
        mailSender.testConnection();
      } catch (MessagingException e) {
        throw new EmailException("无法连接到邮箱服务器，请检查邮箱配置.[" +
          e.getMessage() + "]", e);
      }
    }
  }
  pri
    .
  }
}
```

https://github.com/halo-dev/halo

```
INFO T012 MailServiceImpl.testConnection:()V
INFO   There are calls to methods that should be mocked because they perform
INFO   file system operations, but we cannot mock them.
INFO   Methods that cannot be mocked:
INFO     org.springframework.mail.javamail.JavaMailSenderImpl.testConnection:()V
INFO     org.springframework.mail.javamail.JavaMailSenderImpl.connectTransport:()Lja…
...
INFO     java.io.FileDescriptor.<init>:()V
```

Assumptions and Limitations

Allow dirty tricks?

- Reflection
- Byte code manipulations

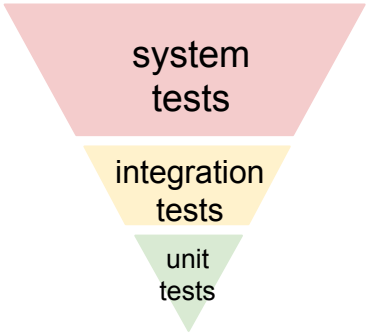
When is something still a unit test?

What should be mocked?

- Files?, network, threads, time, random

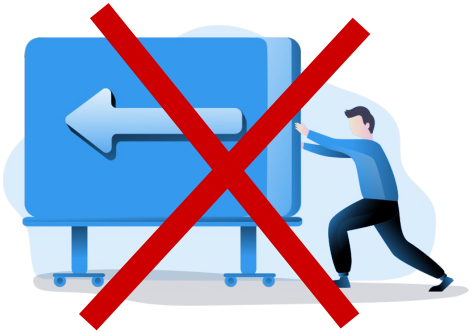
Non-deterministic tests with determinstic verdict?

What are the implications of testability on test efficiency?



Lack of unit-testability

- Tendency to have a higher proportion of system and integration tests
- Slow CI
  - Test selection
  - Nightly test runs
  - Later defect detection
  - No shift-left possible



What are the implications of testability on coverage metrics?

Focus on critical code coverage:

- Projects have 5-40% trivial code
- Easy to increase coverage by 10% without any added value
- 80% coverage is bad if the remaining 20% are critical

Focus on badly unit-testable, critical code:

- Areas of risk in the code base that need attention

What's the role of design for testability?

Common workarounds for lack of testability

- Lack of controllability:
  - Use of bytecode rewriting (e.g. Powermock)
  - Integration test (e.g. with database, emulation of external client)
    → Slow CI
- Lack of observability:
  - Use of reflection
  - Ad-hoc weakening of encapsulation
  - Assertions on log file content
    → Further reduction of code quality

(Binder 1994)

What should actually be done?

- Consider requirements on the **testing interface** when designing functional interface

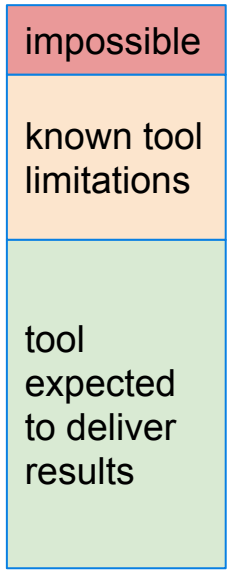
What are the implications of testability on verification tools?

Verification harnesses are very similar to unit tests

- Automated test generation essentially produces harness automatically
- Lack of testability is also an impediment for automated software verification
- Code designed for testability expected easier to handle

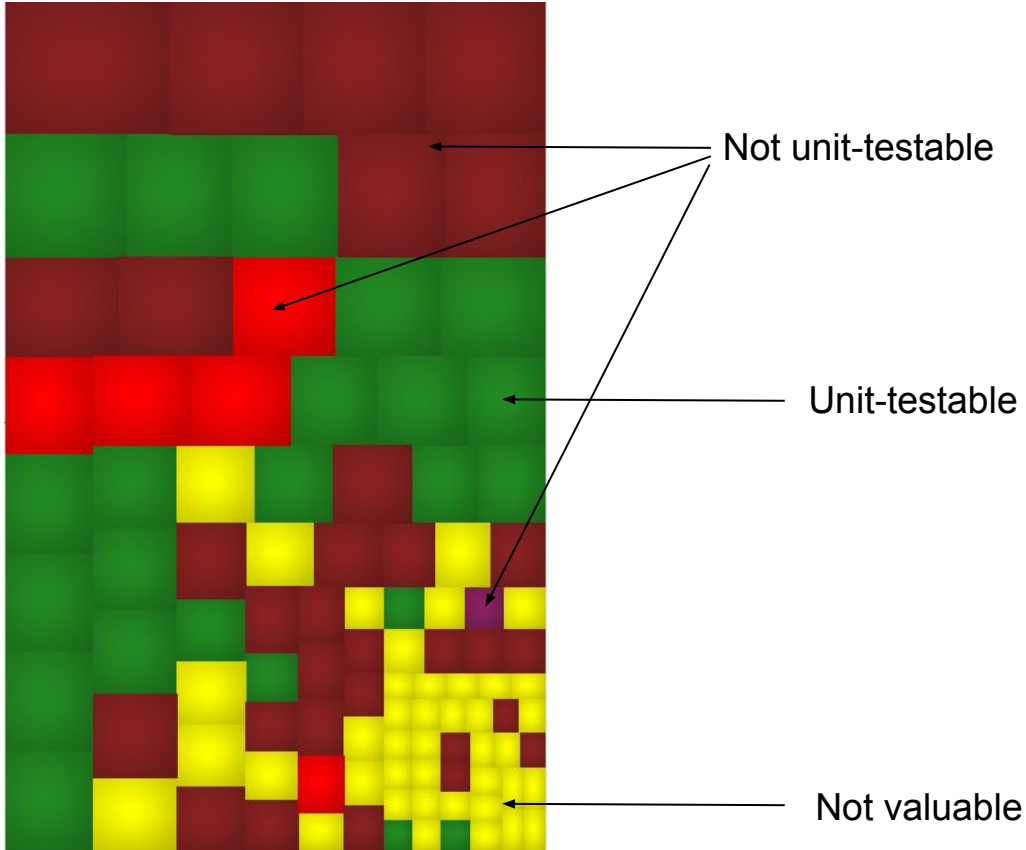
“Verifiability” analysis

- Could estimate upper bounds on what a verification tool can be expected to achieve
- Point out limitations when dealing with real world projects



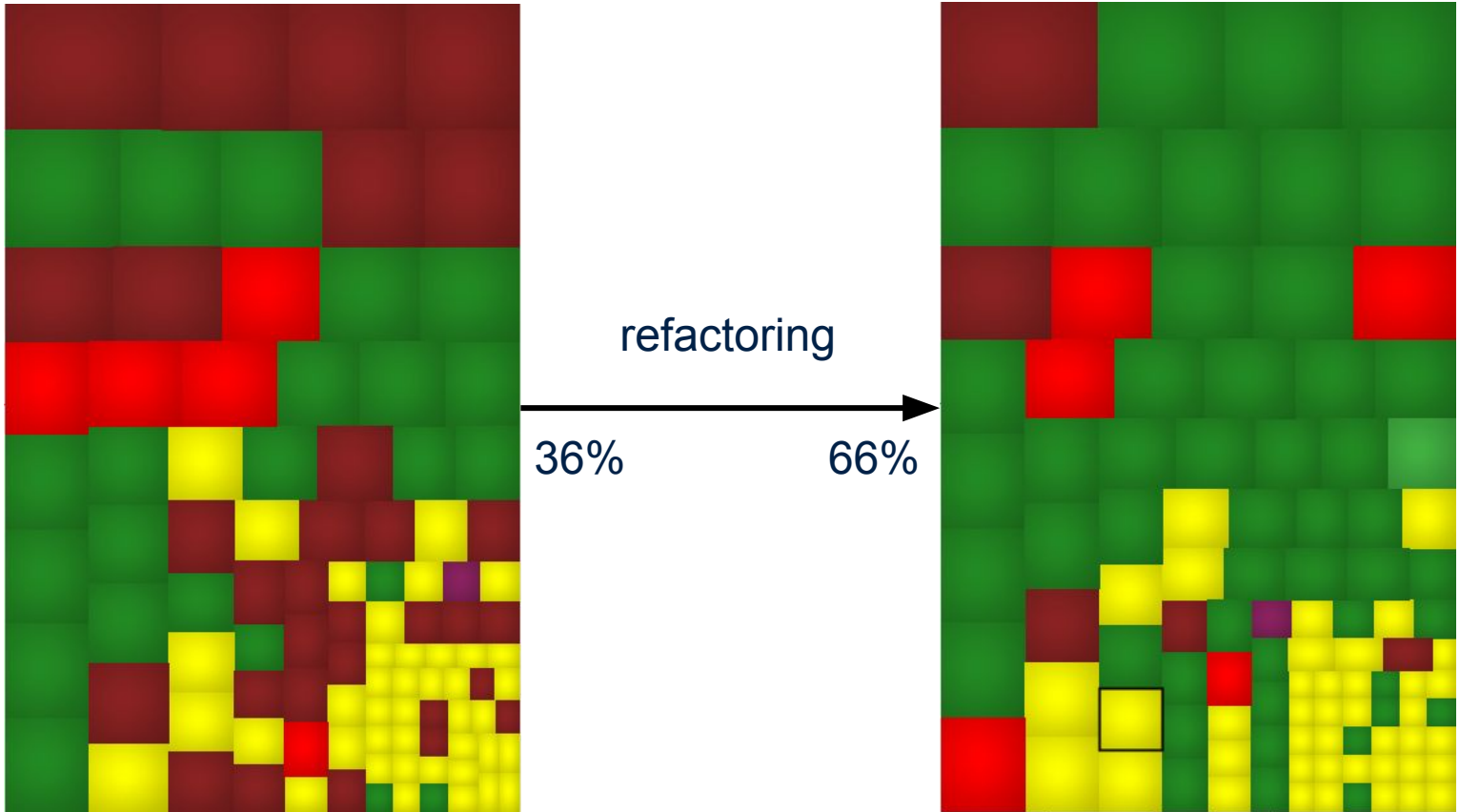
diffblue
AI for Code

What can we do to improve testability?



diffblue
AI for Code

What can we do to improve testability?



Control non-determinism by improving injectability

Improve observability of relevant effects

diffblue
AI for Code

Take-aways

Business software is business-critical.

Unit-testing is important to move fast.

Test coverage metrics have to take into account criticality.

Testability depends on controllability and observability.

Lack of testability affects automated software verification and test generation tools

Automated refactoring/advice to improve testability of business software?

References

Kalman, R. E., “On the General Theory of Control Systems”. Proceedings of the First International Congress on Automatic Control, Butterworth, London, 1960, pp. 481-493.

Edwards, N. P., “The effect of certain modular design principles on software testability”. ACM SIGPLAN Notices, 10(6):401--410, April 1975.

McCabe, T., “A Complexity Measure”, IEEE Trans. Software Eng. 2 (4): 308-320, 1976.

Freedman, R.S., “Testability of Software Components”, IEEE Transactions on Software Engineering, Vol. 17(6), 1991.

Binder, R. V., “Design for testability in object-oriented systems”. Communications of the ACM, Sept 1994.

Binder, R. V., “Testing Object-Oriented Systems: Models, Patterns, and Tools.: Addison Wesley, 1999.

Jungmayr, S., “Testability measurement and software dependencies”, In Proceedings of the 12th International Workshop on Software Measurement, October 7-9, 2002, Magdeburg, Germany, pp. 179-202.

Bruntink, M., Deursen, A.V., “An empirical study into class testability”, Journal of Systems and Software, 2006.

Zhao, L., “A new approach for software testability analysis”. ICSE, 2006.

Khan, R. A., Mustafa, K., “Metric Based Testability Model for Object-Oriented Design (MTMOOD)”. ACM SIGSOFT Software Engineering Notes, volume 34, number 2, March 2009.

V. Chowdhary, “Practicing Testability in the Real World”, International Conference on Software Testing, Verification and Validation, IEEE Computer Society Press, 2009.

Kout, A., Toure, F., Badri, M., “An empirical analysis of a testability model for object-oriented programs”, ACM SIGSOFT Software Engineering Notes, August 2011.

Garousi, V., Felderer, M., Kılıçaslan, F. N., “A Survey on Testability”, Information and Software Technology, Vol. 108, pp. 35-64, April 2019.

Terragni, V., Toure, F., Pezze, M., “Measuring Software Testability Modulo Test Quality”. ICPC, 2020.

Further resources and results: https://bit.ly/2ZUNMOY