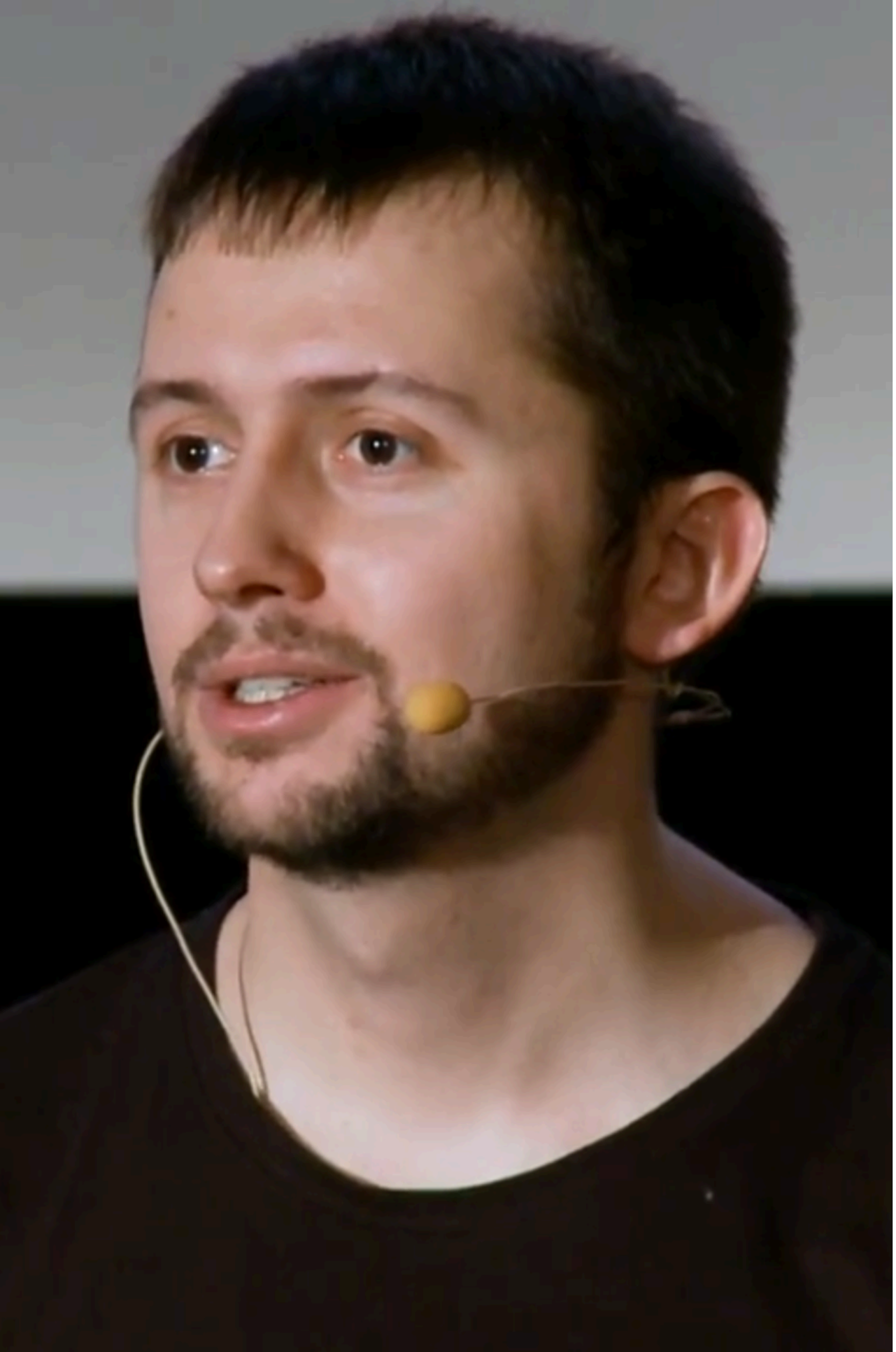
GETTING STARTED WITH GOOGLE TEST

Introduction to test automation

AGENDA

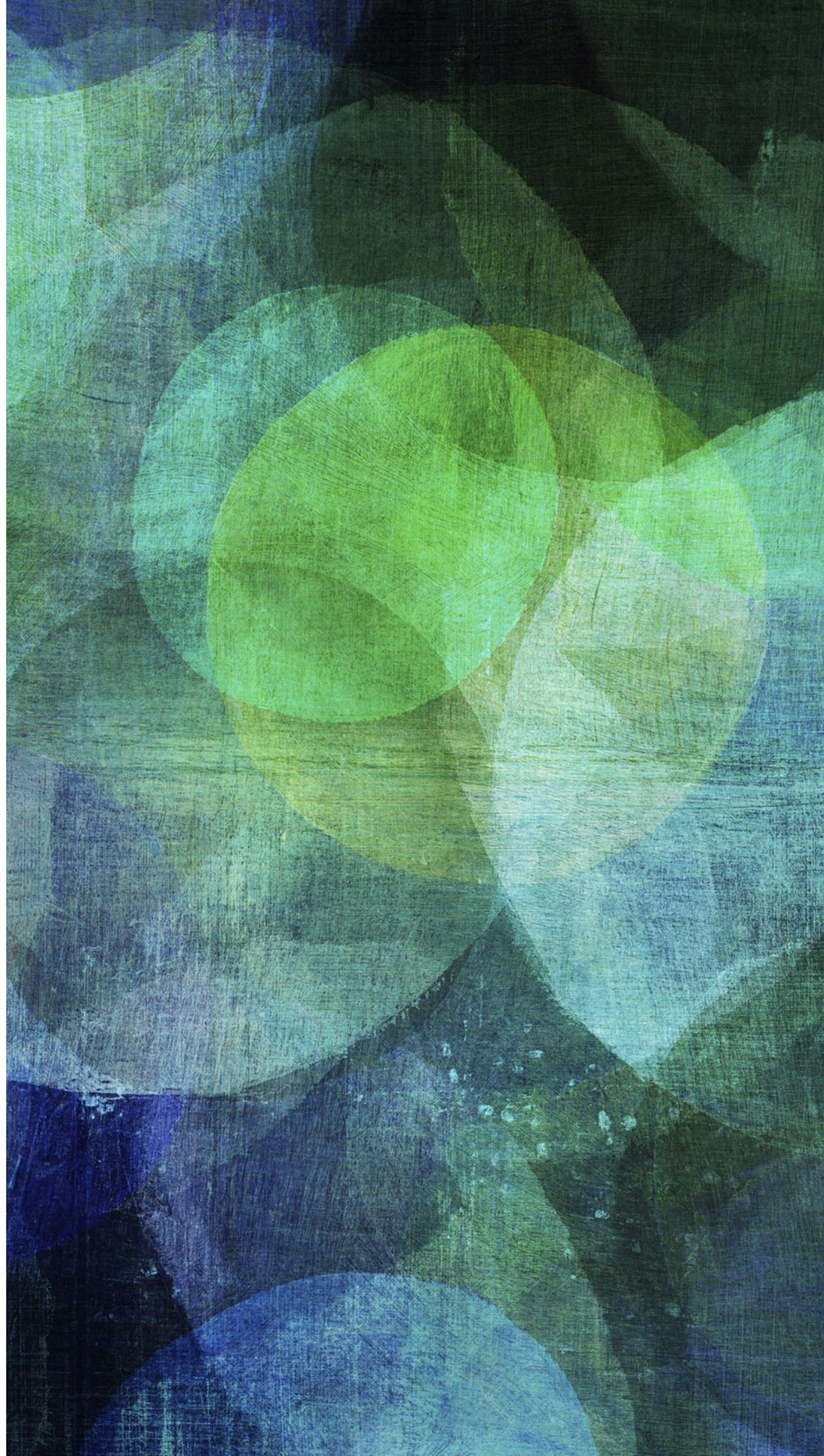
- motivation for test automation
- basic assertions
- mocks and dependency injection
- matchers & actions
- expectation's saturation
- conclusions



MICHAŁ KOWALCZYK

- 5 years as C++ developer
- 3 years experience with Google Test
- Senior Software Engineer @ TomTom
- Michi - Map rendering engine
- Likes conferences
- Is decorating his flat

MOTIVATION



INNOCENT REFACTORING

```
ProcessResult Processor::process(const bool tryFullProcess)
{
  ProcessResult result;

  const bool shouldRunLongProcess = canRunFullProcess() && tryFullProcess;

  internalShortProcessPhase1(result);

  if (shouldRunLongProcess)
  {
    internalLongProcess(result);
  }

  internalShortProcessPhase2(result);

  return result;
}
```

INNOCENT REFACTORING

```
ProcessResult Processor::process(const bool tryFullProcess)
{
  ProcessResult result;

  const bool shouldRunLongProcess = canRunFullProcess() && tryFullProcess;

  internalShortProcessPhase1(result);

  if (shouldRunLongProcess)
  {
    internalLongProcess(result);
  }

  internalShortProcessPhase2(result);

  return result;
}
```

INNOCENT REFACTORING

```
ProcessResult Processor::process(const bool tryFullProcess)
{
  ProcessResult result;

  const bool shouldRunLongProcess = tryFullProcess && canRunFullProcess();

  internalShortProcessPhase1(result);

  if (shouldRunLongProcess)
  {
    internalLongProcess(result);
  }

  internalShortProcessPhase2(result);

  return result;
}
```

INNOCENT REFACTORING

```
bool Process::canRunFullProcess()
{
  const bool result = hasALotOfResources();

  if (result)
  {
    prepareResourcesForLongProcess();
  }
  else
  {
    prepareResourcesForShortProcess();
  }

  return result;
}
```

INNOCENT REFACTORING

```
bool Process::canRunFullProcess()
{
  const bool result = hasALotOfResources();

  if (result)
  {
    prepareResourcesForLongProcess();
  }
  else
  {
    prepareResourcesForShortProcess();
  }

  return result;
}
```

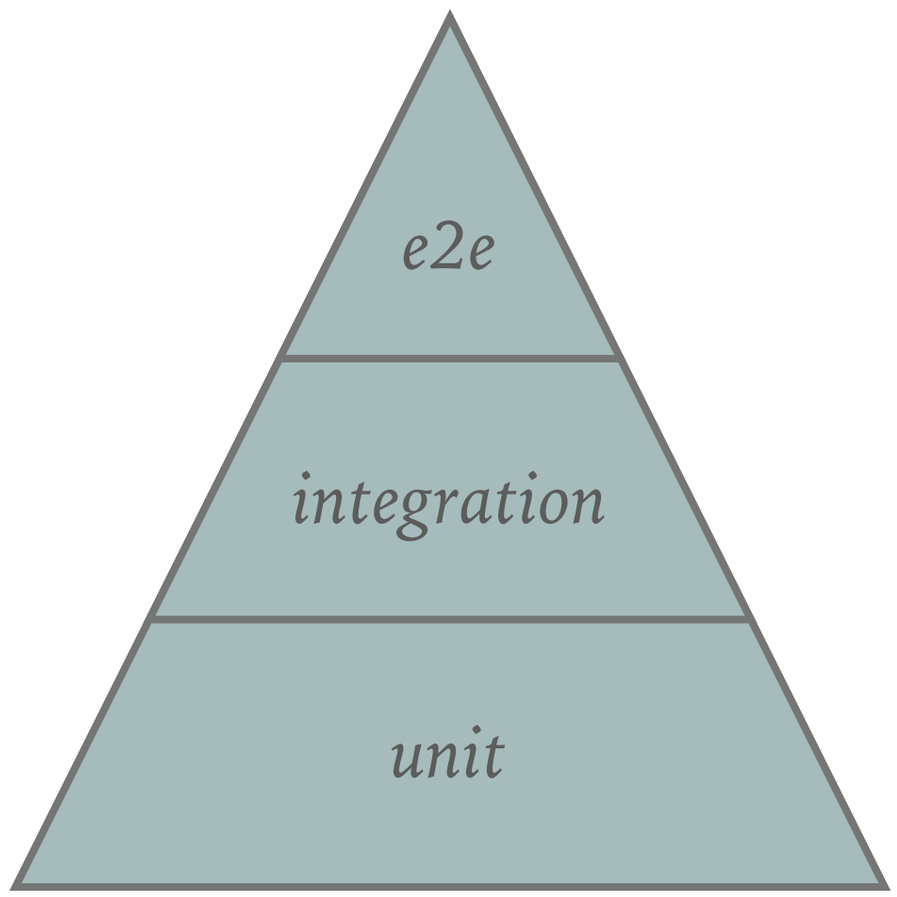
UNIT TEST INTRODUCTION

TEST LEVELS

- Unit tests
- Integration tests
- Acceptance tests

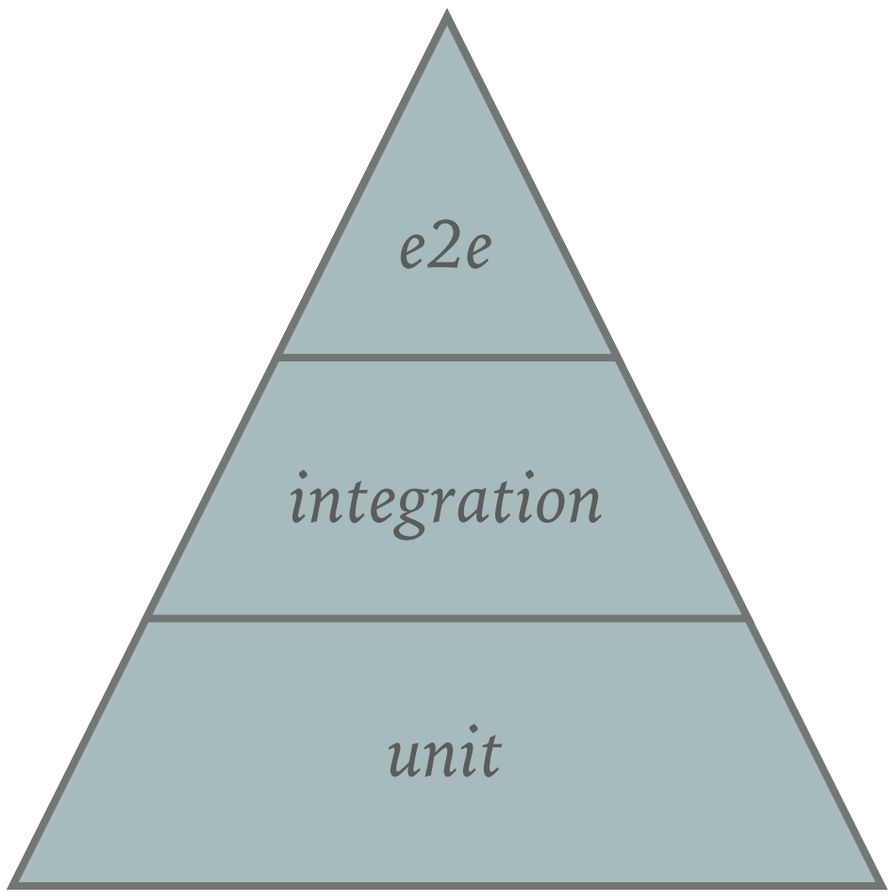
TEST LEVELS

- Unit tests
- Integration tests
- Acceptance tests



TEST LEVELS

- Unit tests (22741)
- Integration tests (1336)
- Acceptance tests (124)



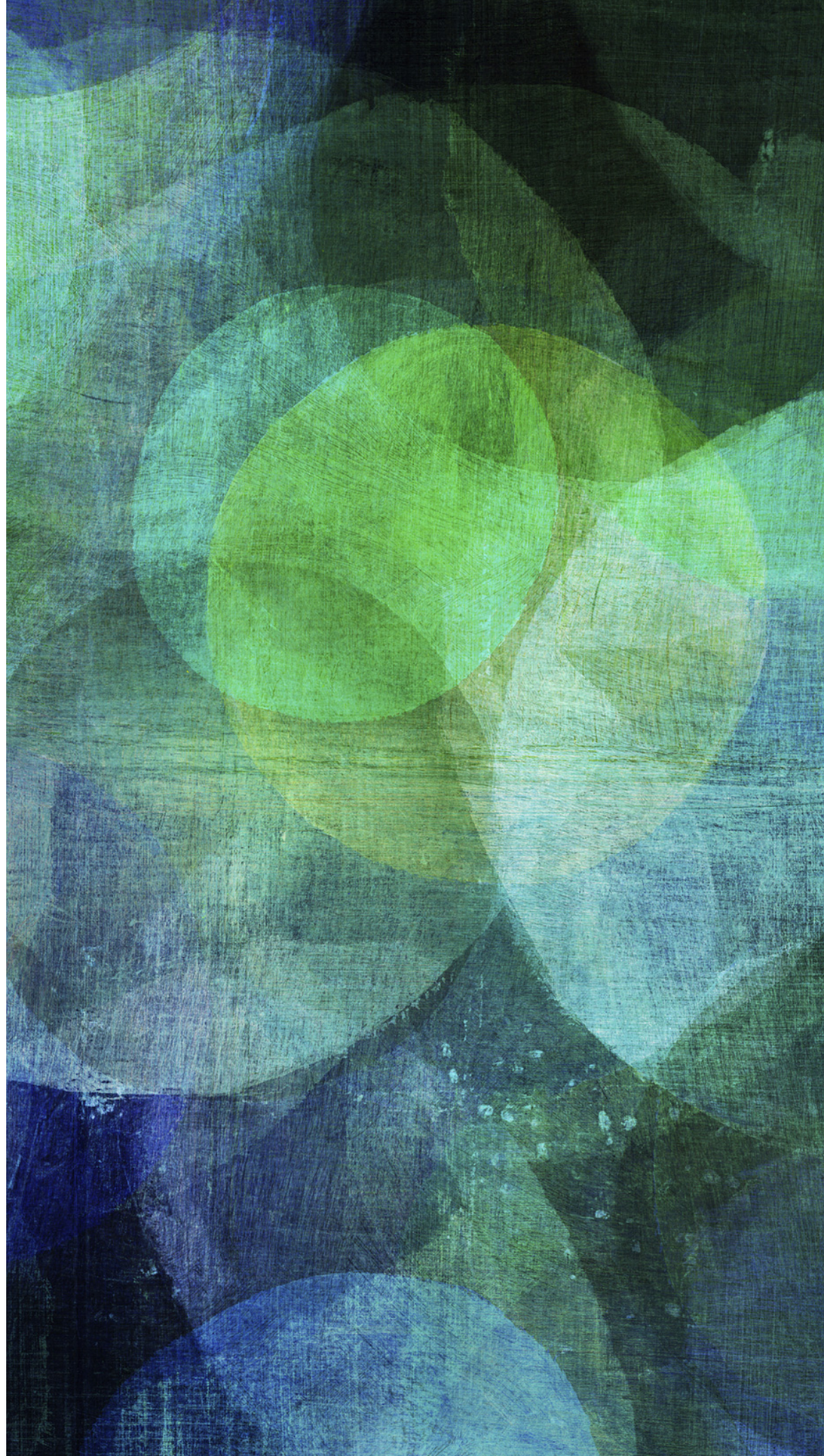
BENEFITS

- Shorten feedback loop
- Increased confidence
- Refactoring
- Better design
- Higher quality
- Less bugs
- Samples of use

ABOUT GOOGLE TEST

- GTest / GMock
- C++
- Multiplatform
- Used in LLVM, OpenCV, wide variety of Google projects
- xUnit based

BASIC ASSERTIONS



BASIC CONCEPTS

- Test (Test Case)
- Test Case (Test Suite)
- Fixtures
- (Non) fatal failures
- Assertions / expectations

ASSERTIONS

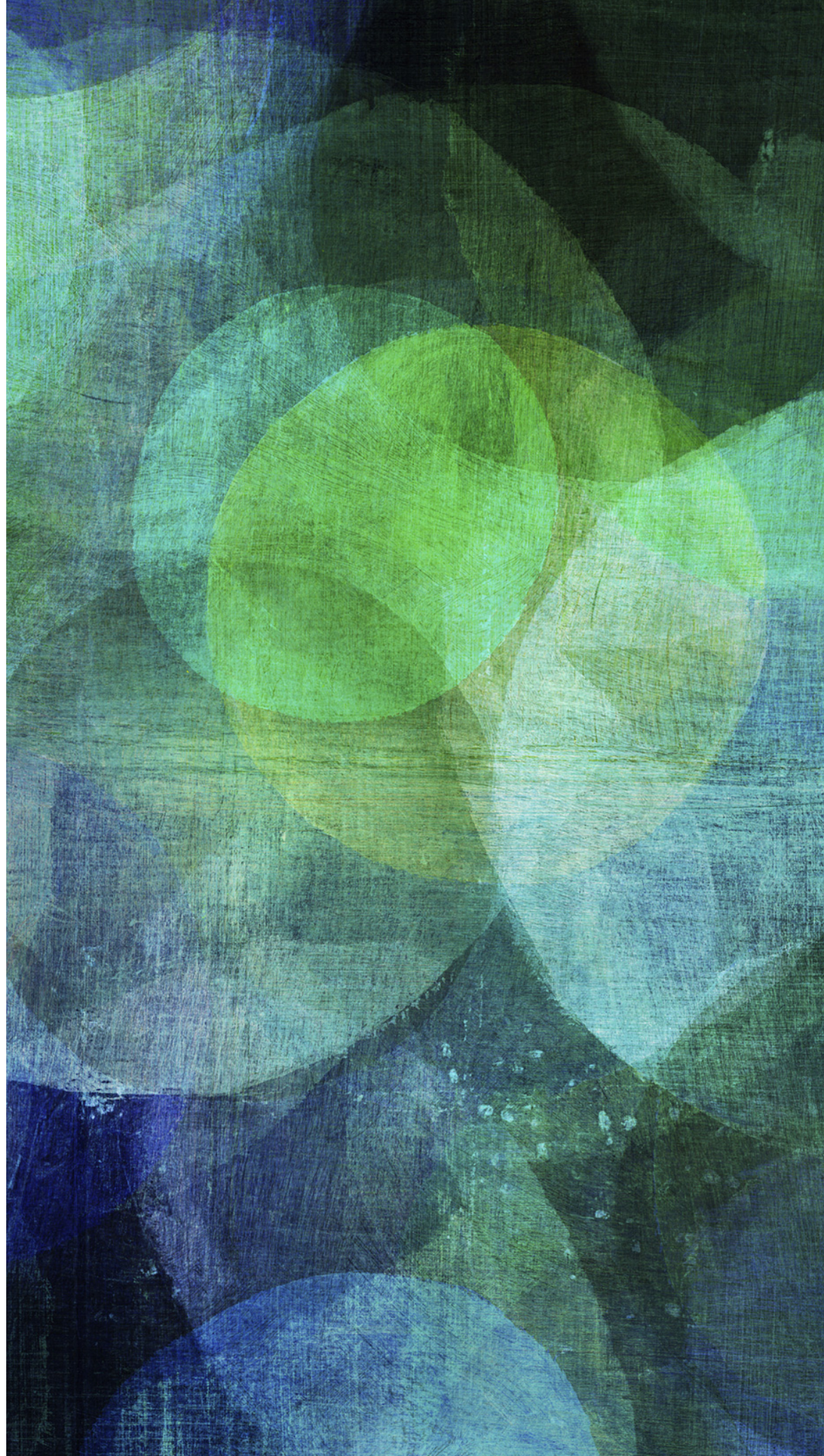
- ASSERT/EXPECT_TRUE, *e.g. EXPECT_TRUE(isPrime(7));*
- ASSERT/EXPECT_FALSE, *e.g. EXPECT_FALSE(isPrime(6));*
- ASSERT/EXPECT_EQ, *e.g. EXPECT_EQ(42, theAnswer);*
- ASSERT/EXPECT_NE, *e.g. EXPECT_NE();*
- ASSERT/EXPECT_STREQ, *e.g. EXPECT_STREQ("\n", nl.c_str())*
- ASSERT/EXPECT_STRNE, *e.g. EXPECT_STREQ("\t", nl.c_str())*
- ASSERT/EXPECT_LT/GT/LE/GE, *e.g. EXPECT_LT(fibb(2), fibb(3));*
- ASSERT/EXPECT_NEAR, *e.g. EXPECT_NEAR(3.14, M_PI, 0.01);*
- ASSERT/EXPECT_THROW, *e.g. EXPECT_THROW(vec.at(10));*
- ASSERT/EXPECT_NO_THROW, *e.g. EXPECT_NO_THROW(vec.at(0));*

LET'S CREATE A UNIT TEST

WHEN TO USE?

- Global function
- Static functions
- Classes with no dependencies
- Classes with cheap dependencies

MOCKS AND DEPENDENCY INJECTION



DEPENDENCIES HARD TO TEST

- I/O
  - Filesystem
  - Database
  - Interprocess communication
  - Network
- GPU
- Long running tasks
- Non deterministic behaviours
  - Asynchronous
  - Random

DEPENDENCY INJECTION

- Prefer interfaces over concrete classes
- In production use concrete class that derives from the interface
- In tests use mock class that derives from the interface
- That applies to static polymorphism too!

INTRODUCE AN INTERFACE

```
class FileInterface
{
public:
   virtual ~FileInterface() {}

   virtual void open() = 0;
   virtual void close() = 0;

   virtual bool isOpen() const = 0;

   virtual void read(std::string& text) = 0;
   virtual void write(const std::string& text) = 0;
};
```

CREATE A CONCRETE CLASS

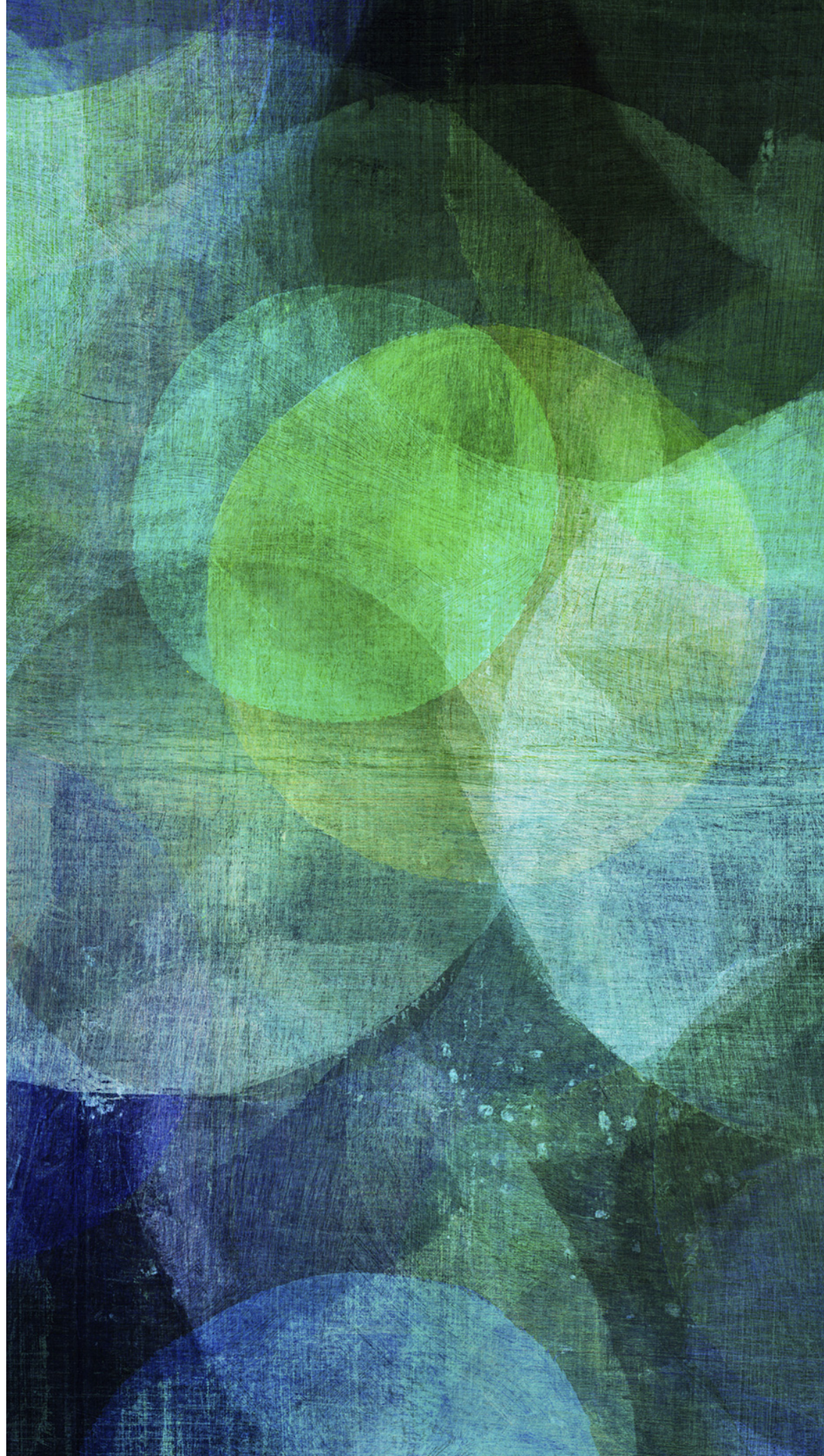
```
class File : public FileInterface
{
public:
  ~File() override;

  void open() override;
  void close() override;

  bool isOpen() const override;

  void read(std::string& text) override;
  void write(const std::string& text) override;
};
```

CREATE A MOCK

```
class MockFile : public FileInterface {
public:
   MOCK_METHOD0(open,
        void());
   MOCK_METHOD0(close,
        void());
   MOCK_CONST_METHOD0(isOpen,
        bool());
   MOCK_METHOD1(read,
        void(std::string& text));
   MOCK_METHOD1(write,
        void(const std::string& text));
};
```

LET'S USE MOCKS IN TESTS!

MATCHERS & ACTIONS



MATCHERS

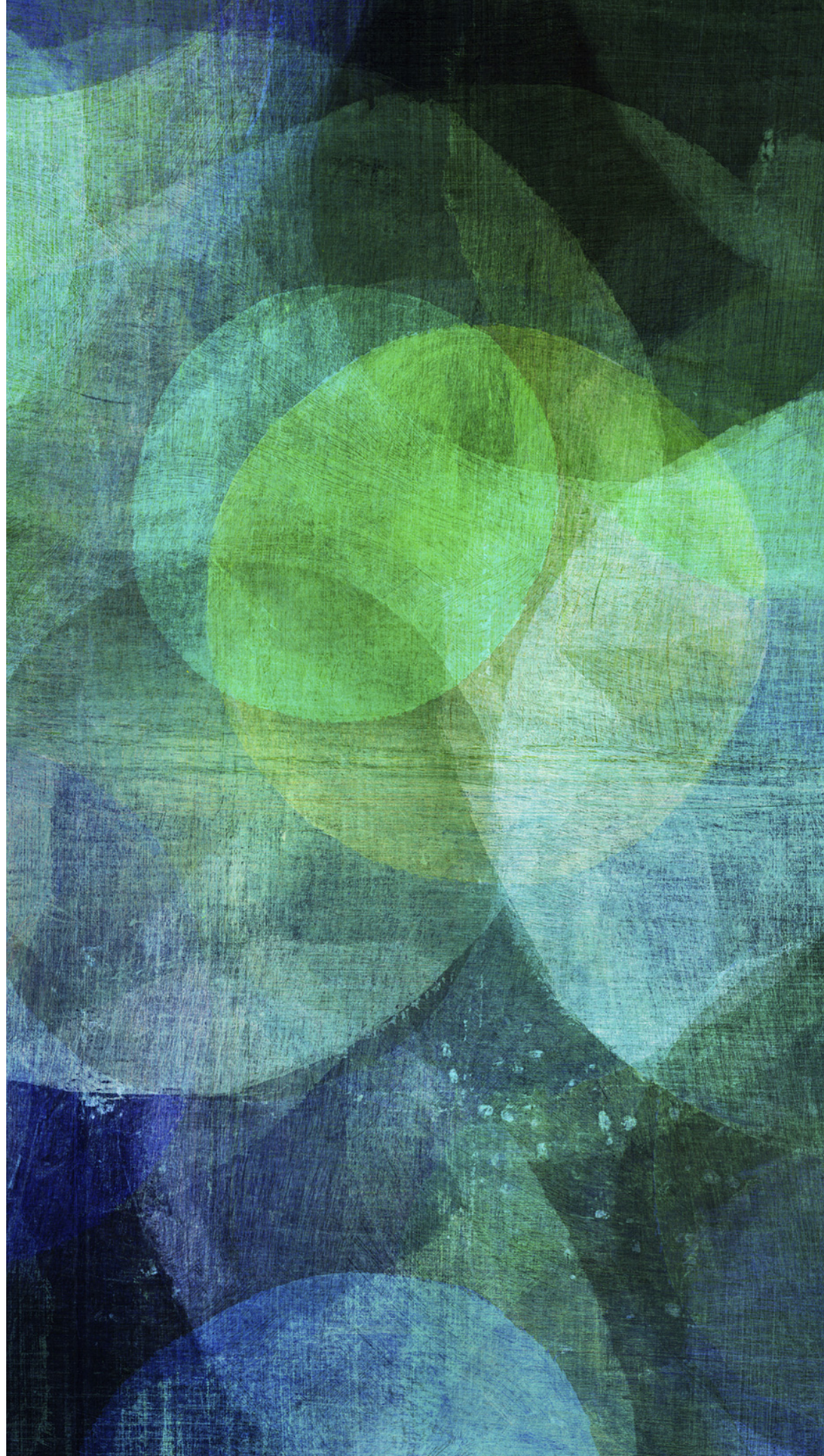
- Matches an arg's value to an expectation
- May be composites
- Extensible
- Examples:
  - Eq(value)
  - StrEq(string)
  - Not(matcher)
  - _

ACTIONS

- Define what happen when mock is invoked
- May be composites
- Extensible
- Examples:
  - Return(value)
  - Invoke(function)
  - SaveArg<N>(pointer)

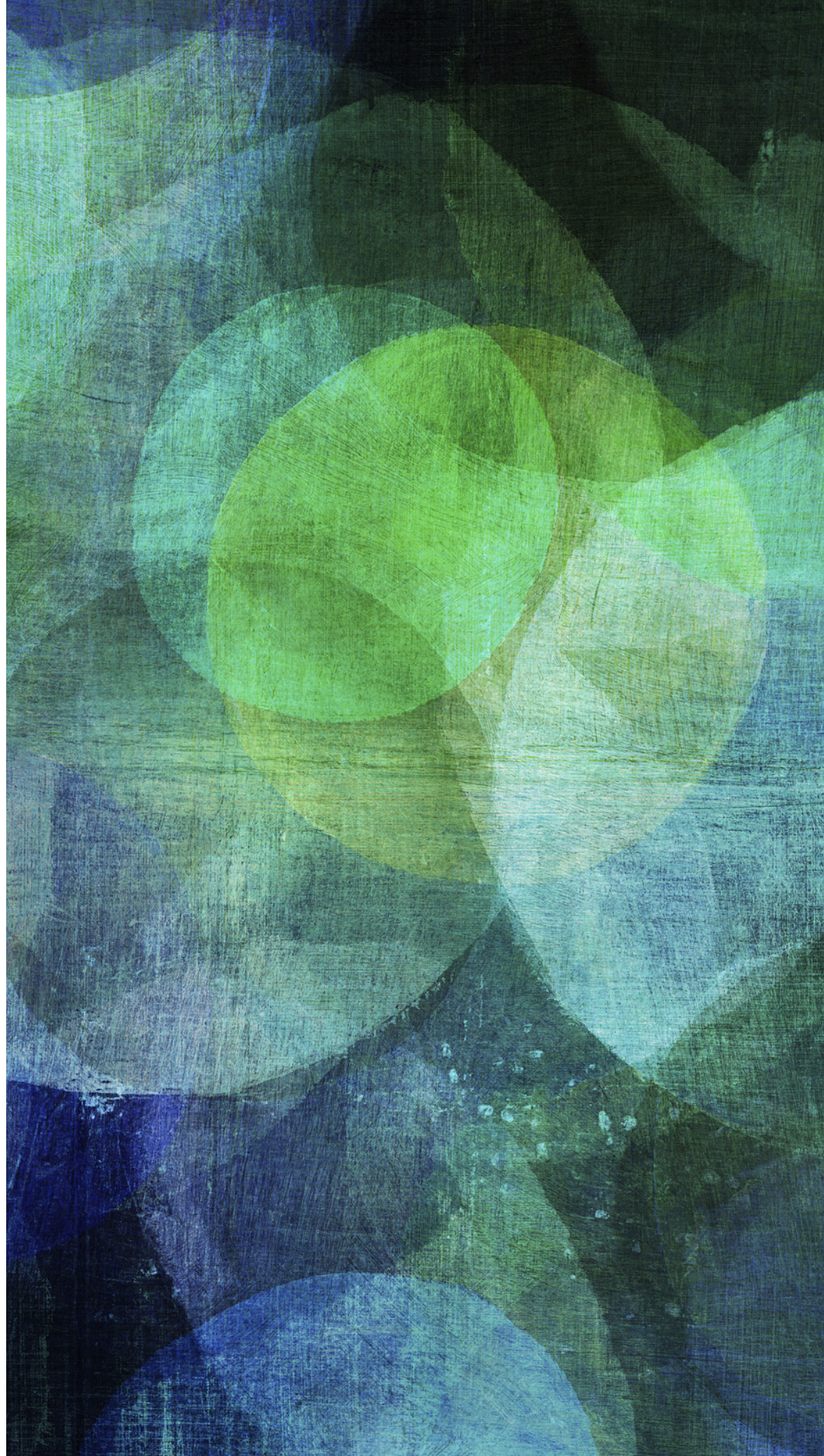
LET'S ADD MATCHERS AND ACTIONS!

SATURATION OF EXPECTATION



WHEN MOCKS GET COMPLICATED

FINAL WORDS



MISCELLANEOUS

- Executable documentation
- Strict and nice mocks
- Don’t fix tests by adding EXPECT_CALLs

LINKS

- github.com/google/googletest
- github.com/michal-kowalczyk
- xunitpatterns.com

THANK YOU

@michal_k_k