#UFADEVCONF

Юнит-тестирование унаследованного кода: безопасный рефакторинг

Ястребов Виктор

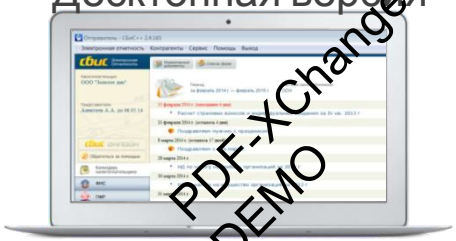
к.т.н., ведущий разработчик компании “Тензор”

http://dc.ufacoder.com - 2018

Компания Тензор

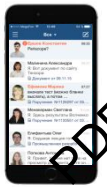
- СБИС – сеть деловых коммуникаций и обмена электронными документами между компаниями, госорганами и обыкновенными людьми
- Свыше 1 000 разработчиков
- Свыше 1 000 000 пользователей

Десктопная версия



Веб-решение

Приложение



О чем будем говорить

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

О чем будем говорить

- Принципы создания тестируемого кода

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

О чем будем говорить

- Принципы создания тестируемого кода
- Понятие поддельных объектов

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

PDF-XChange Editor DEMO

О чем будем говорить

- Принципы создания тестируемого кода
- Понятие поддельных объектов
- Разбор на примере

PDF-XChange Editor DEMO

О чем будем говорить

- Принципы создания тестируемого кода
- Понятие поддельных объектов
- Разбор на примере
- Общие рекомендации и практические приемы

Допущения

- Примеры с использованием C++

Допущения

- Примеры с использованием C++
- Код упрощен

Допущения

- Примеры с использованием C++
- Код упрощен

- Используется Google Test

https://github.com/google/googletest/tree/master/googletest

Пример унаследованного кода

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename ) ;
      }
   private:
      DatabaseManager    dbManager;
           WebService    webService;
};
```

Пример унаследованного кода

```
class EntityAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ){
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename ) ;
      }
   private:
      DatabaseManager    dbManager;
           WebService    webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

Пример унаследованного кода

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename ) ;
      }
   private:
      DatabaseManager    dbManager;
         WebService      webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

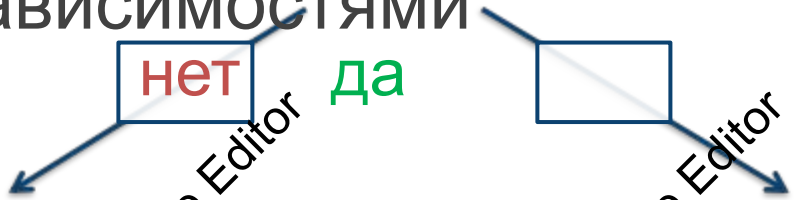
```
class DatabaseManager {
   public:
      bool IsValid( std::string ename ) {
         /* логика, включающая
         операции чтения из базы данных*/
      }
};
```

Внешняя зависимость

Взаимодействие есть, а контроля нет

Интеграционный тест VS юнит-тест

Полный контроль над внешними зависимостями

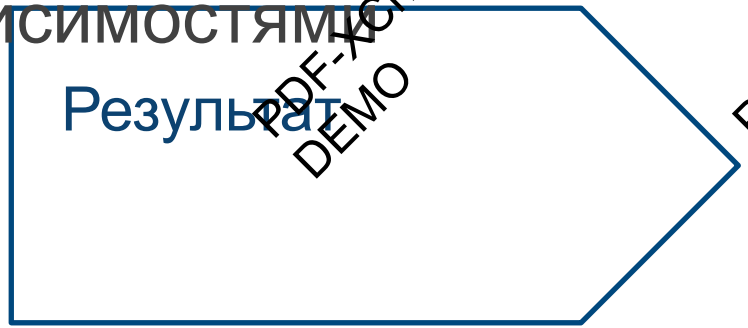


Интеграционный тест

Юнит-тест

Признаки хорошего юнит-теста

Полный контроль над внешними зависимостями

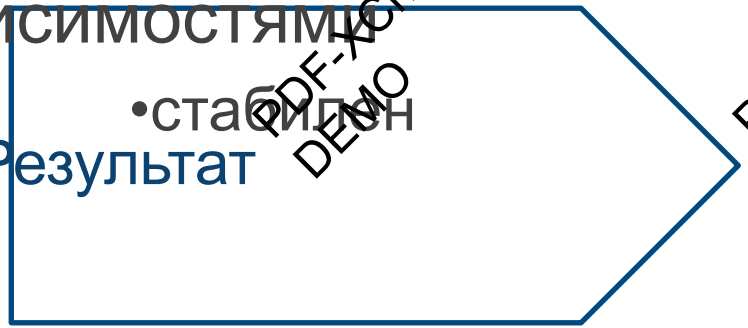


Признаки хорошего юнит-теста

Полный контроль над внешними зависимостями

- стабилен

Результат

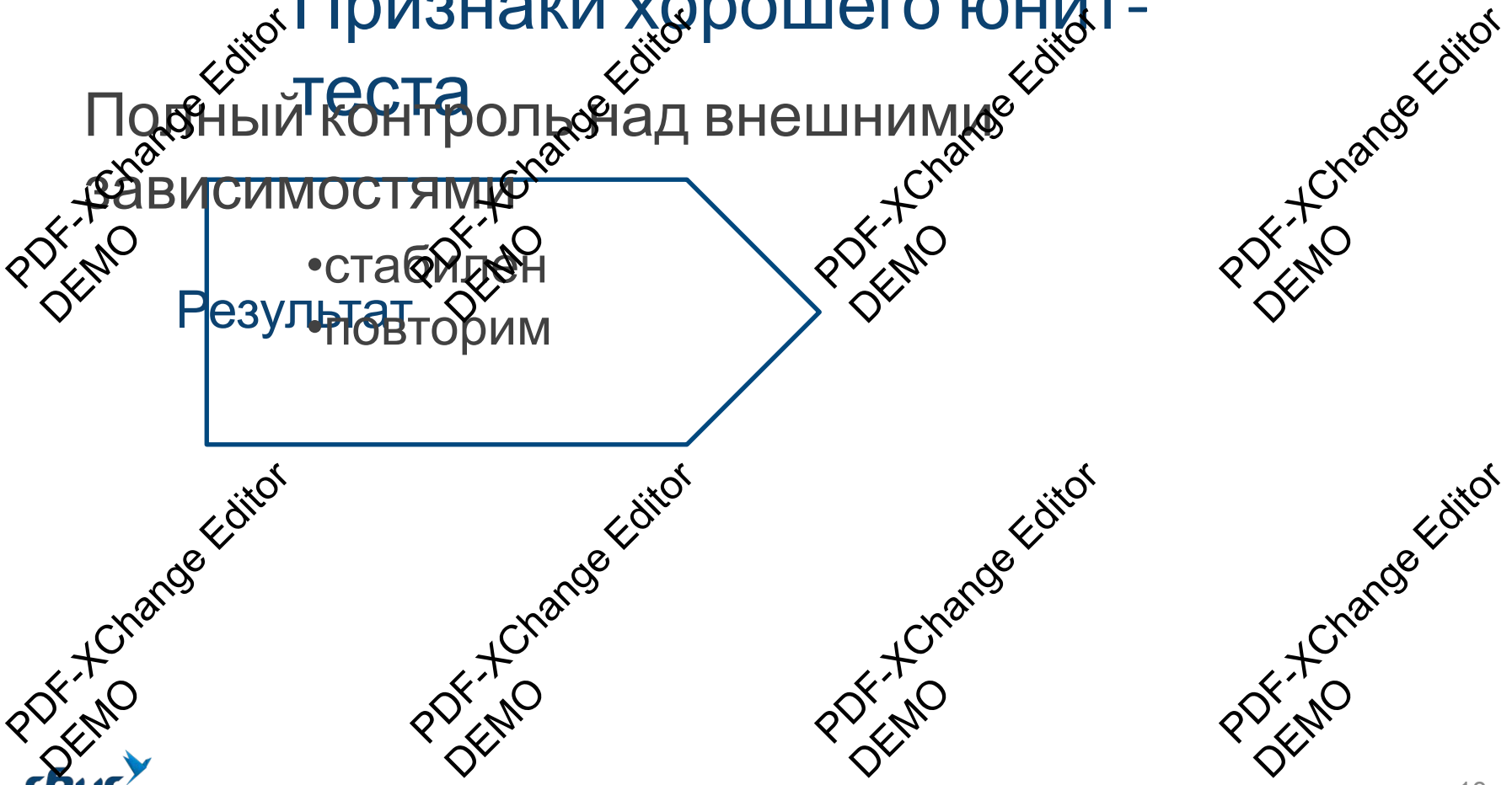


Признаки хорошего юнит-теста

Полный контроль над внешними зависимостями

Результат

- стабилен
- повторим



Признаки хорошего юнит-теста

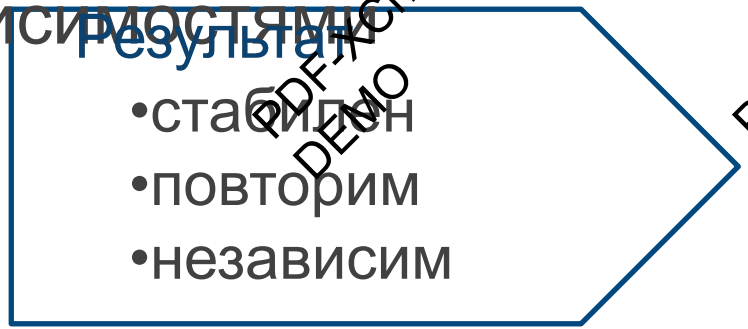
Полный контроль над внешними зависимостями

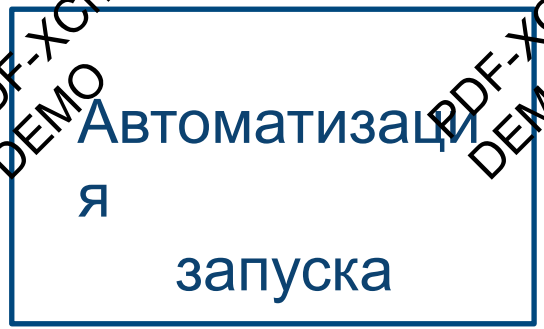
Результат

- стабилен
- повторим
- независим

Признаки хорошего юнит-теста

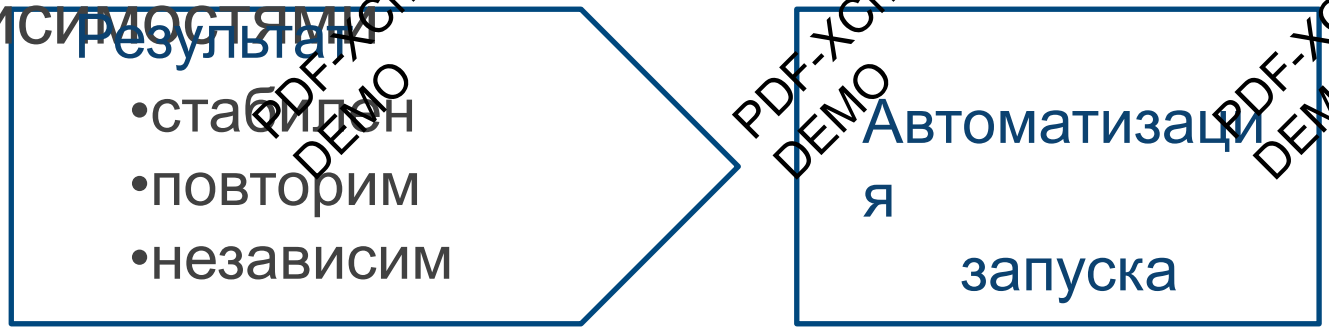
Полный контроль над внешними зависимостями





Признаки хорошего юнит-теста

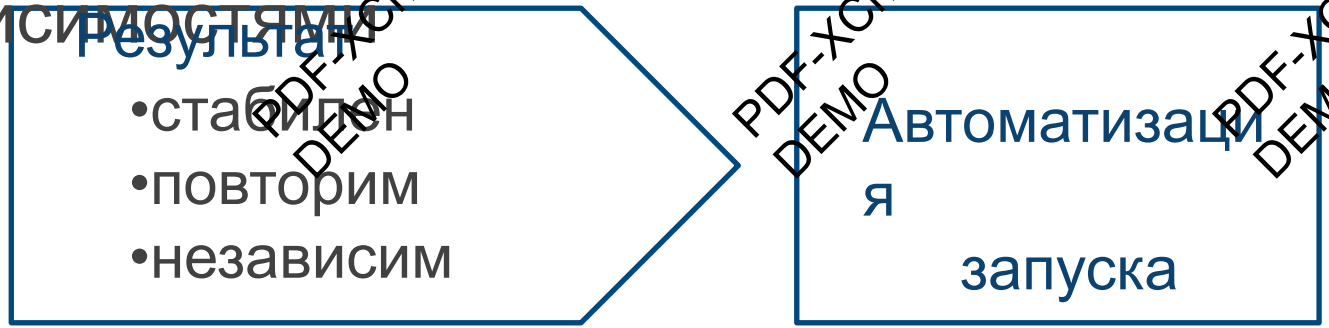
Полный контроль над внешними зависимостями



•Малое время выполнения

Признаки хорошего юнит-теста

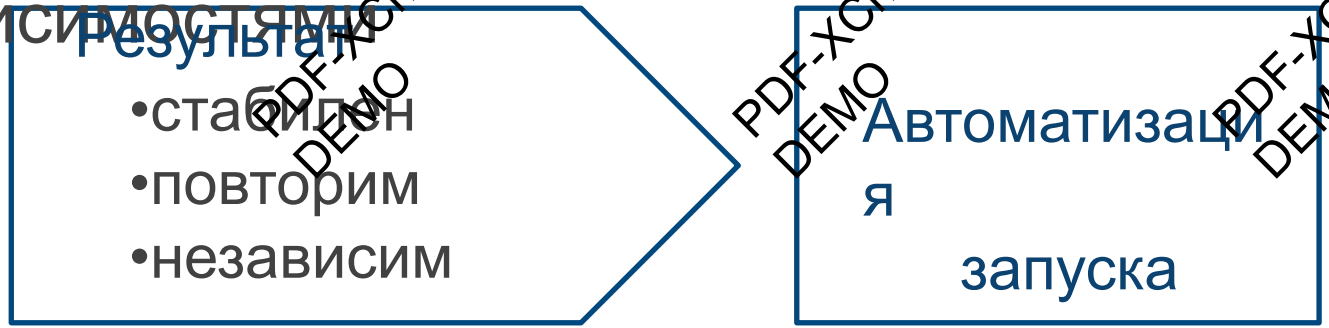
Полный контроль над внешними зависимостями



- Малое время выполнения
- Малое время компиляции

Признаки хорошего юнит-теста

Полный контроль над внешними зависимостями



- Малое время выполнения
- Малое время компиляции
- Простота чтения

Исходный код

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager    dbManager;
           WebService    webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

```
class DatabaseManager {
   public:
      bool IsValid( std::string ename ) {
         /* логика, включающая
         операции чтения из базы данных*/
      }
};
```

Исходный код

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager    dbManager;
      WebService         webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

Внешняя зависимость

```
class DatabaseManager {
   public:
      bool IsValid( std::string ename ) {
         /* логика, включающая
         операции чтения из базы данных*/
      }
};
```

Исходный код

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager    dbManager;
          WebService     webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

```
class DatabaseManager {
   public:
      bool IsValid( std::string ename ) {
         /* логика, включающая
         операции чтения из базы данных*/
      }
};
```

Внешняя зависимость

Исходный код

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager    dbManager;
           WebService    webService;
};
```

```
class WebService {
   public:
      void LogError( std::string msg ) {
         /* логика, включающая
         работу с сетевым соединением*/
      }
};
```

```
class DatabaseManager {
   public:
      bool IsValid( std::string ename ) {
         /* логика, включающая
         операции чтения из базы данных*/
      }
};
```

Внешние зависимости

Как сделать унаследованный код тестируемым?

Понятие программного шва

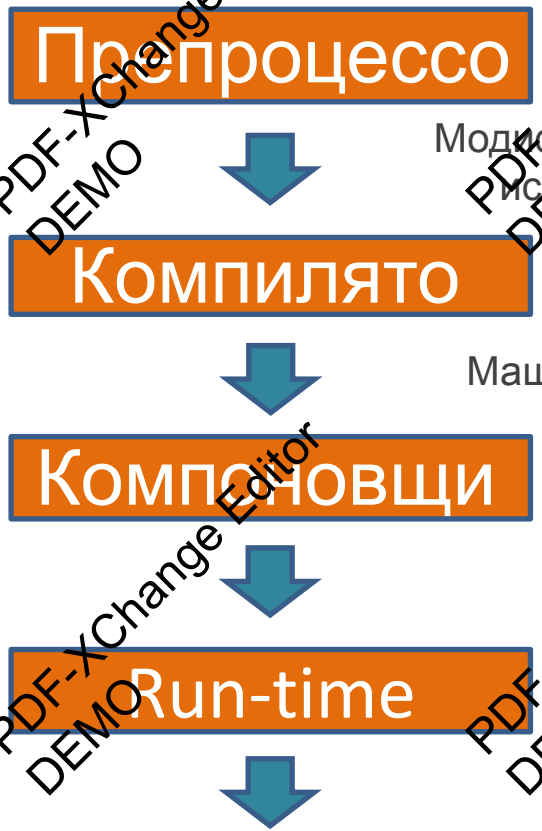
- Шов — место в котором можно изменить поведение программы, не правя ее в этом месте
- Виды швов:
  - Предварительной обработки
  - Этапа компиляции
  - Компоновочный шов
  - Объектный шов

Слои тестирования

Слои тестирования

Модифицированный исходный код

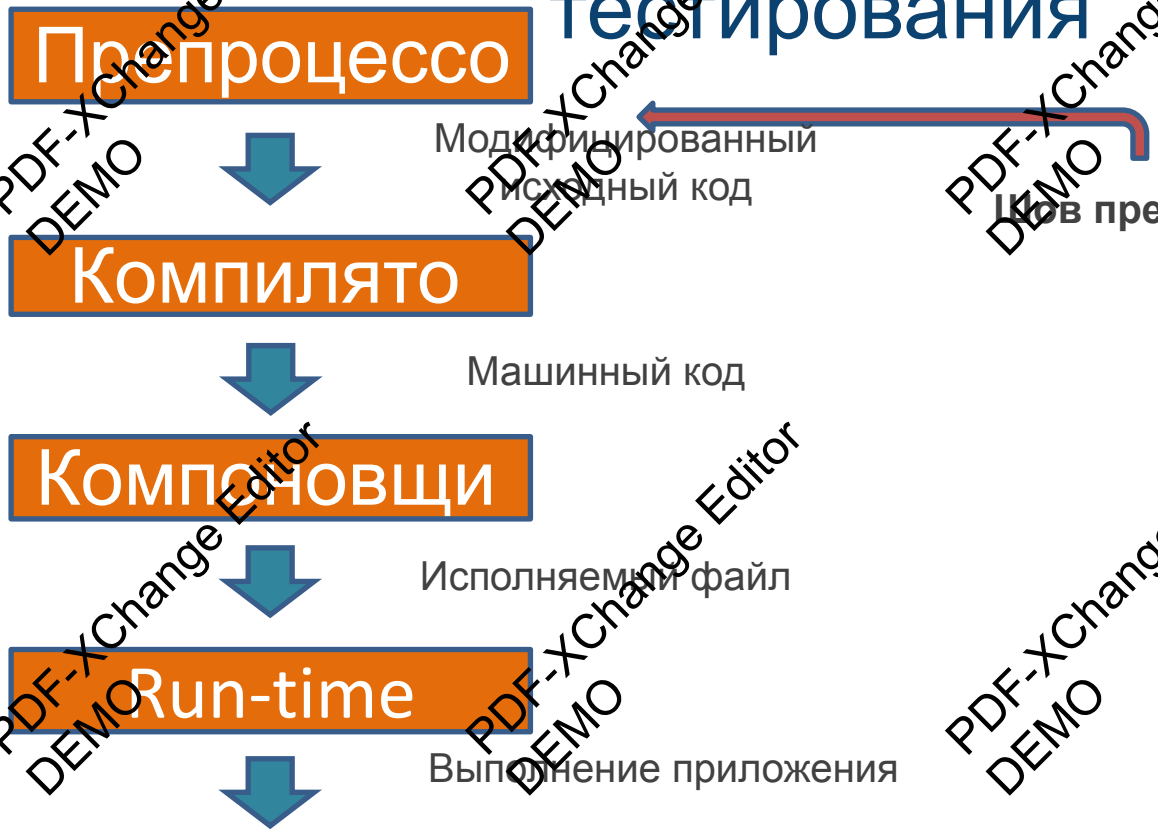
Слои тестирования



Слои тестирования

Слои тестирования

Слои тестирования



Цикл предварительной обработки

Слои тестирования

Модифицированный исходный код

Шов предварительной обработки

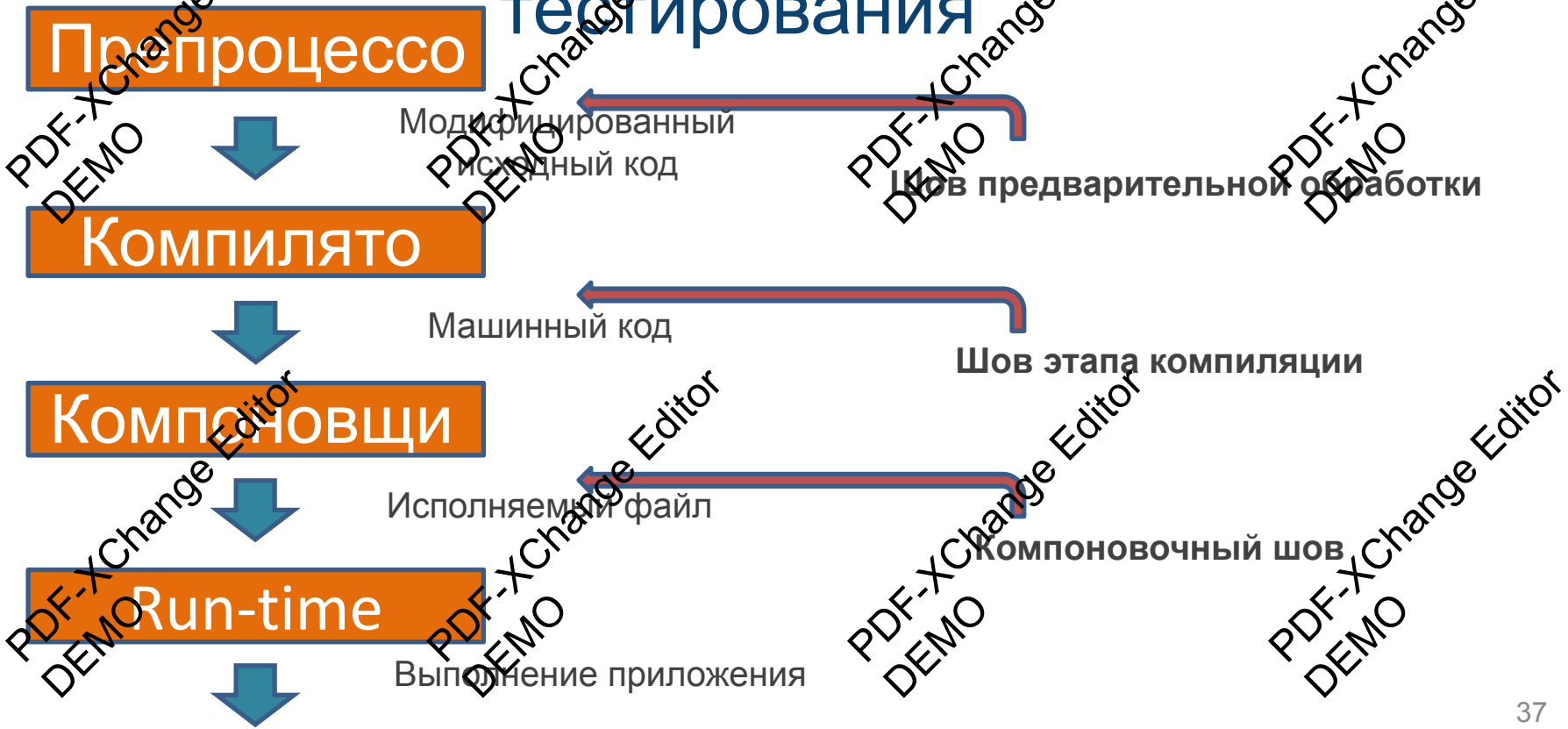
Машинный код

Шов этапа компиляции

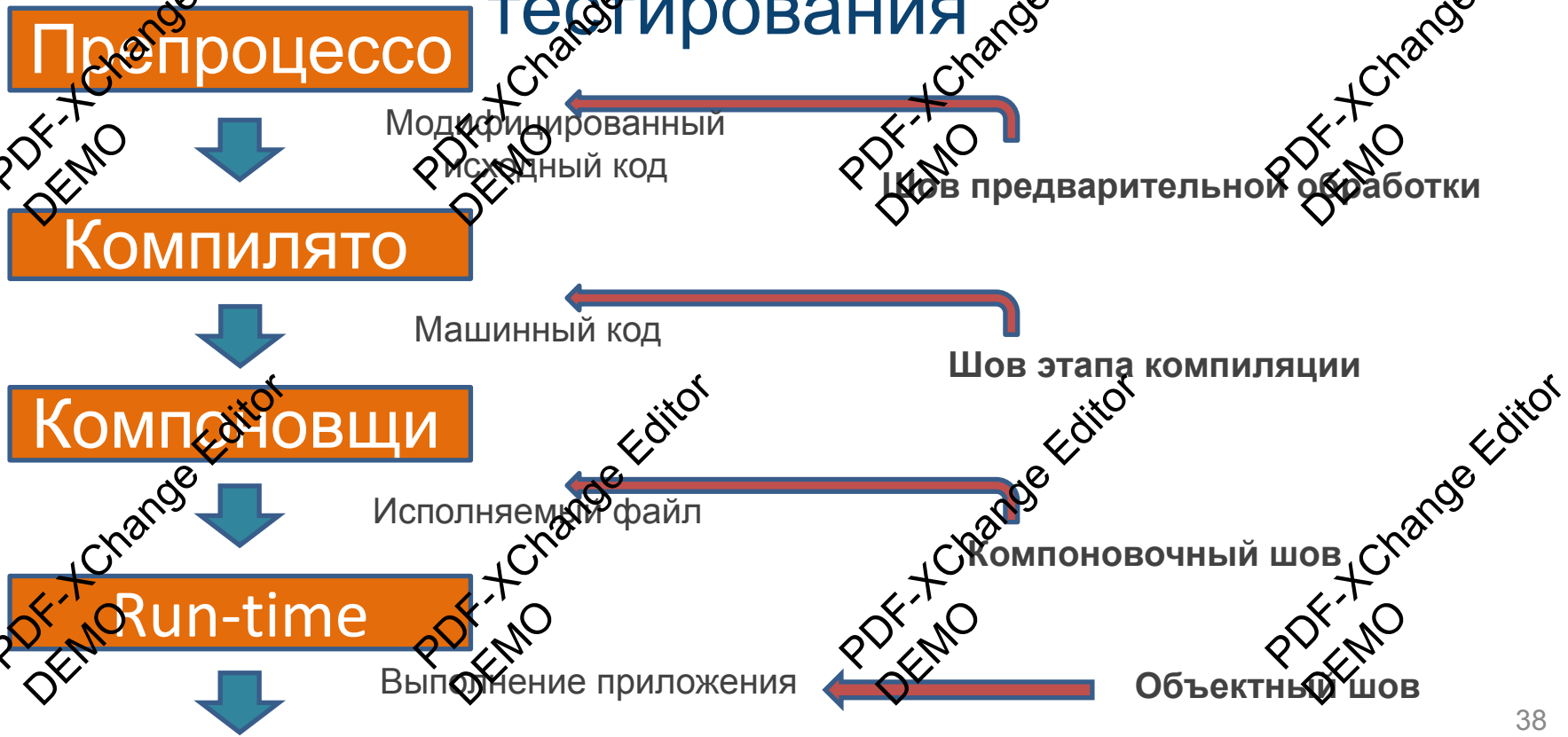
Исполняемый файл

Выполнение приложения

Слои тестирования



Слои тестирования



Слои тестирования

Модифицированный исходный код

Шов предварительной обработки

Машинный код

Шов этапа компиляции

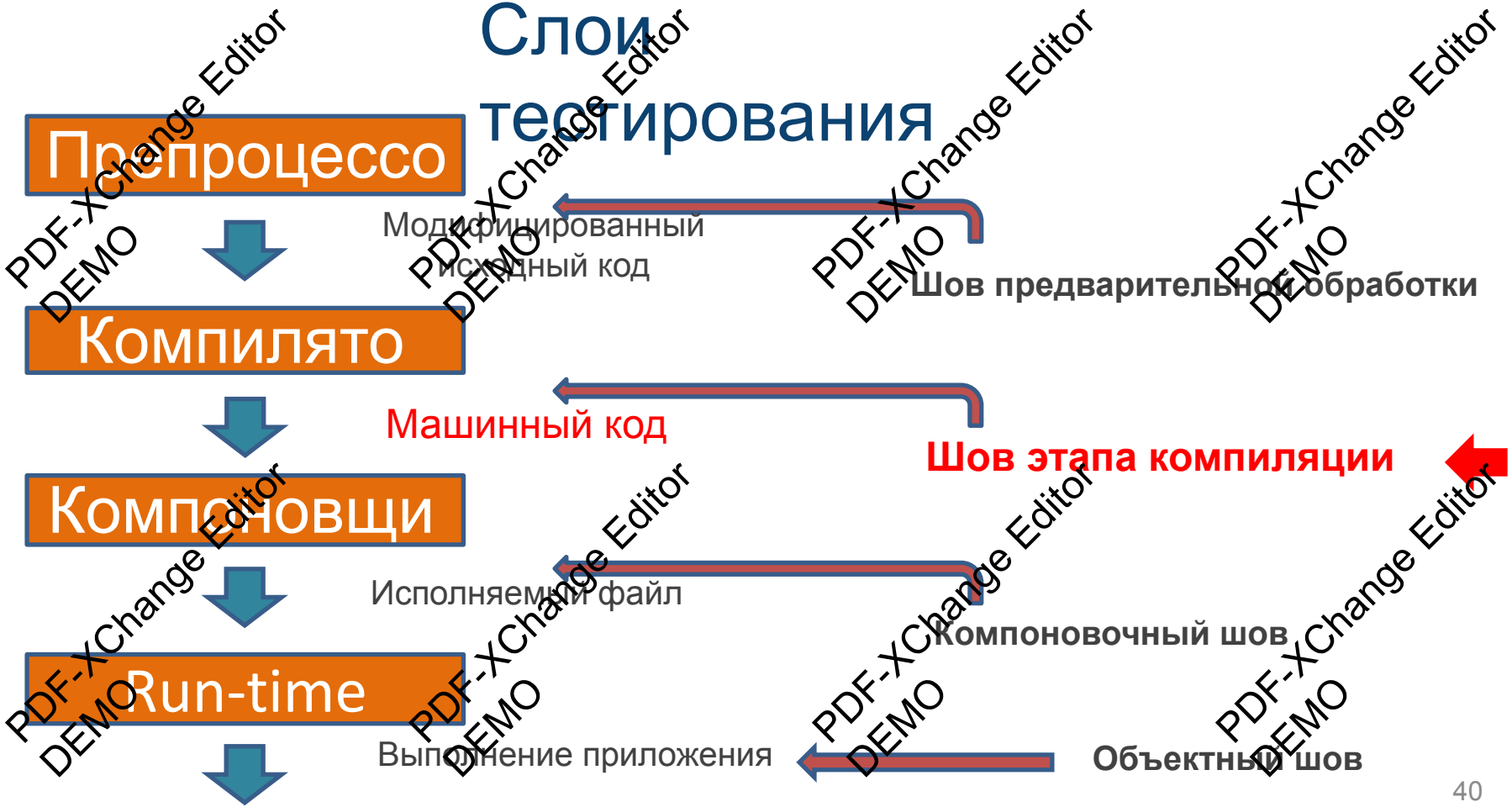
Исполняемый файл

Компоновочный шов

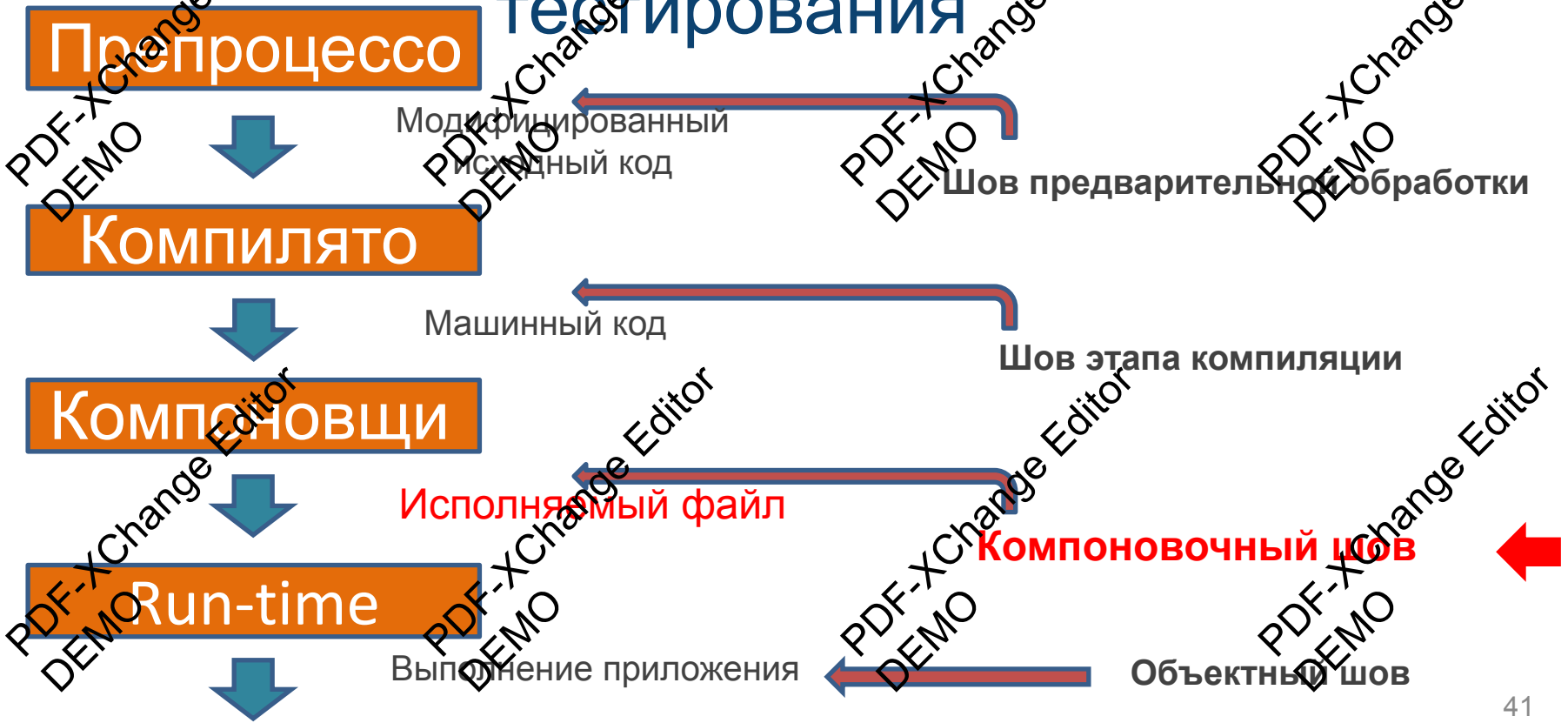
Выполнение приложения

Объектный шов

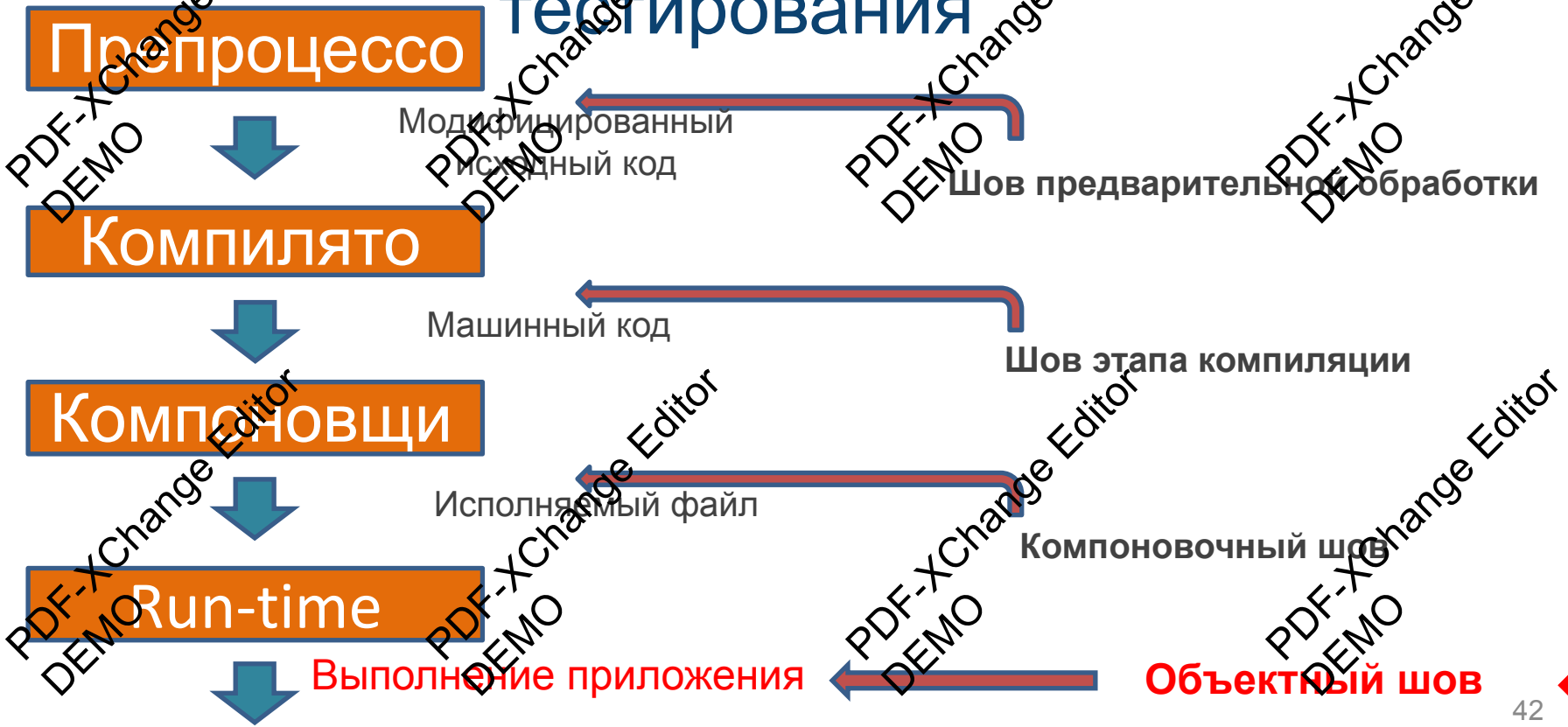
Слои тестирования



Слои тестирования

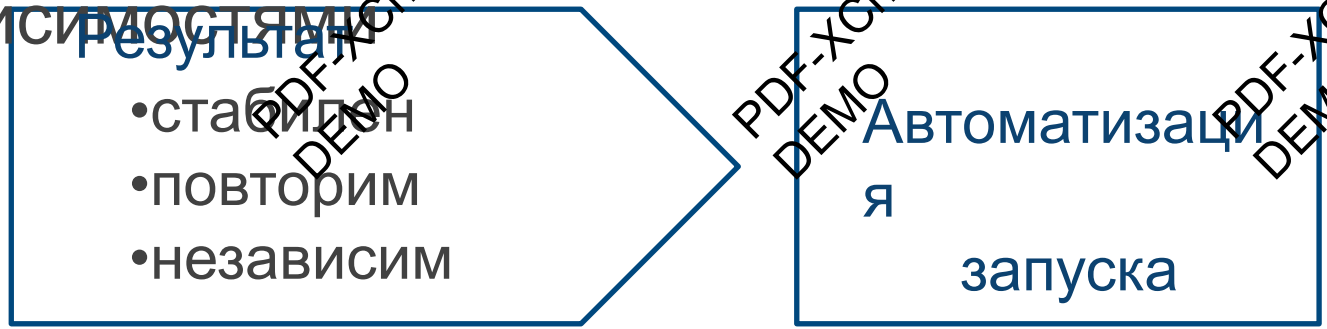


Слои тестирования



Признаки хорошего юнит-теста

Полный контроль над внешними зависимостями



- Малое время выполнения
- Малое время компиляции
- Простота чтения

Принцип наименования юнит-теста

[ИмяТестируемойРабочейЕдиницы]_[СценарийТеста]_[ОжидаемыйРезультат]

Sum_ByDefault_ReturnsZero()

Sum_WhenCalled_CallsTheLogger()

Структура юнит-теста

```
TEST_F( CalculatorTest,
        Sum_ByDefault_ReturnsZero )
{
    Calculator calc;

    int last_sum = calc.Sum();

    ASSERT_EQ( 0, last_sum );
}
```

Структура юнит-теста

```
TEST_F( CalculatorTest,
    Sum_ByDefault_ReturnsZero )
{
  Calculator calc;

  int last_sum = calc.Sum();

  ASSERT_EQ( 0, last_sum );
}
```

1. Arrange

Структура юнит-теста

```
TEST_F( CalculatorTest,
    Sum_ByDefault_ReturnsZero )
{
  Calculator calc;

  int last_sum = calc.Sum();

  ASSERT_EQ( 0, last_sum );
}
```

1. Arrange

2. Act

Структура юнит-теста

```
TEST_F( CalculatorTest,
    Sum_ByDefault_ReturnsZero )
{
  Calculator calc;

  int last_sum = calc.Sum();

  ASSERT_EQ( 0, last_sum );
}
```

1. Arrange
2. Act
3. Assert

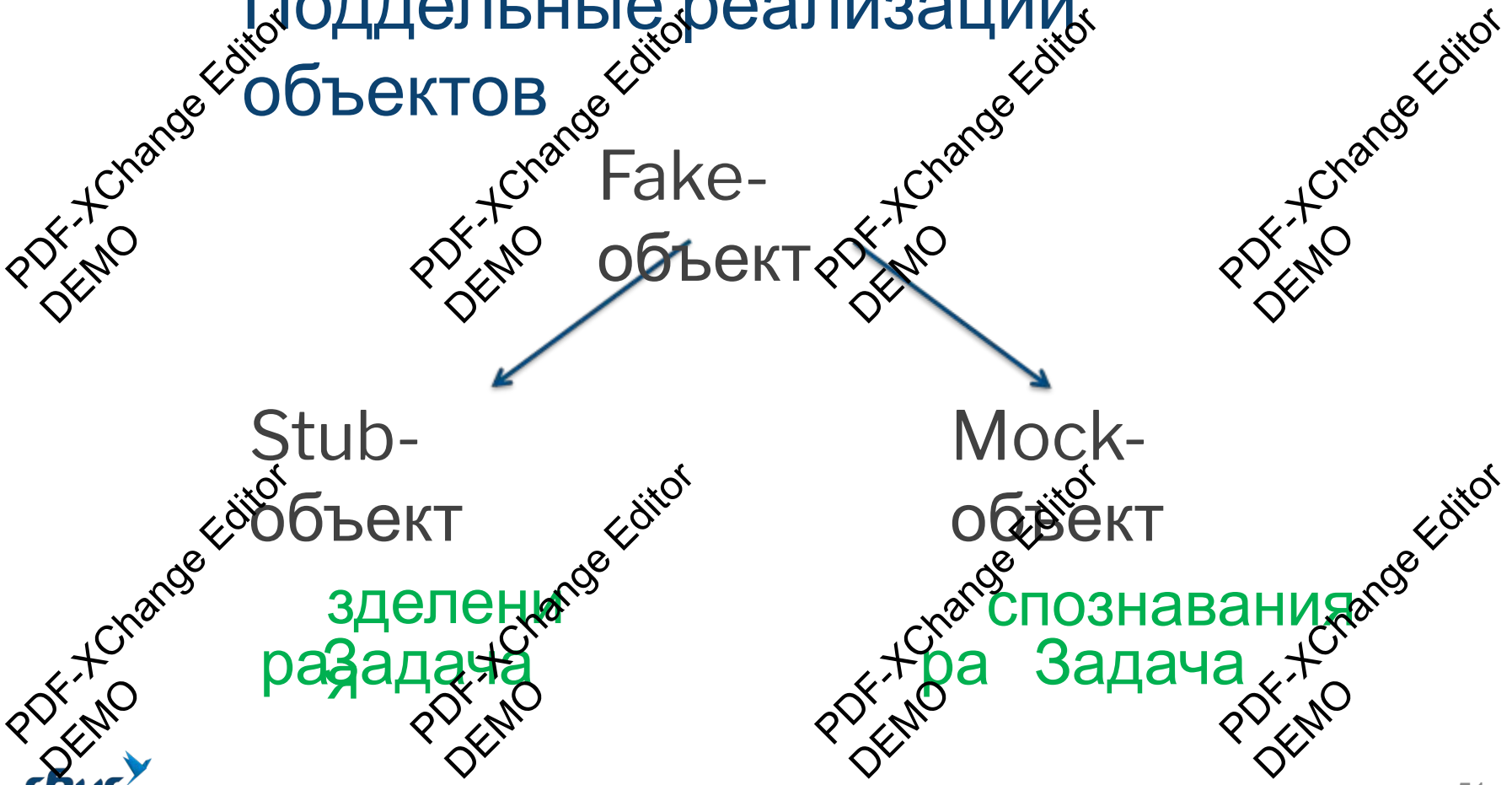
Структура юнит-теста

```
TEST_F( CalculatorTest,
    Sum_ByDefault_ReturnsZero )
{
  Calculator calc;

  int last_sum = calc.Sum();

  ASSERT_EQ( 0, last_sum );
}
```

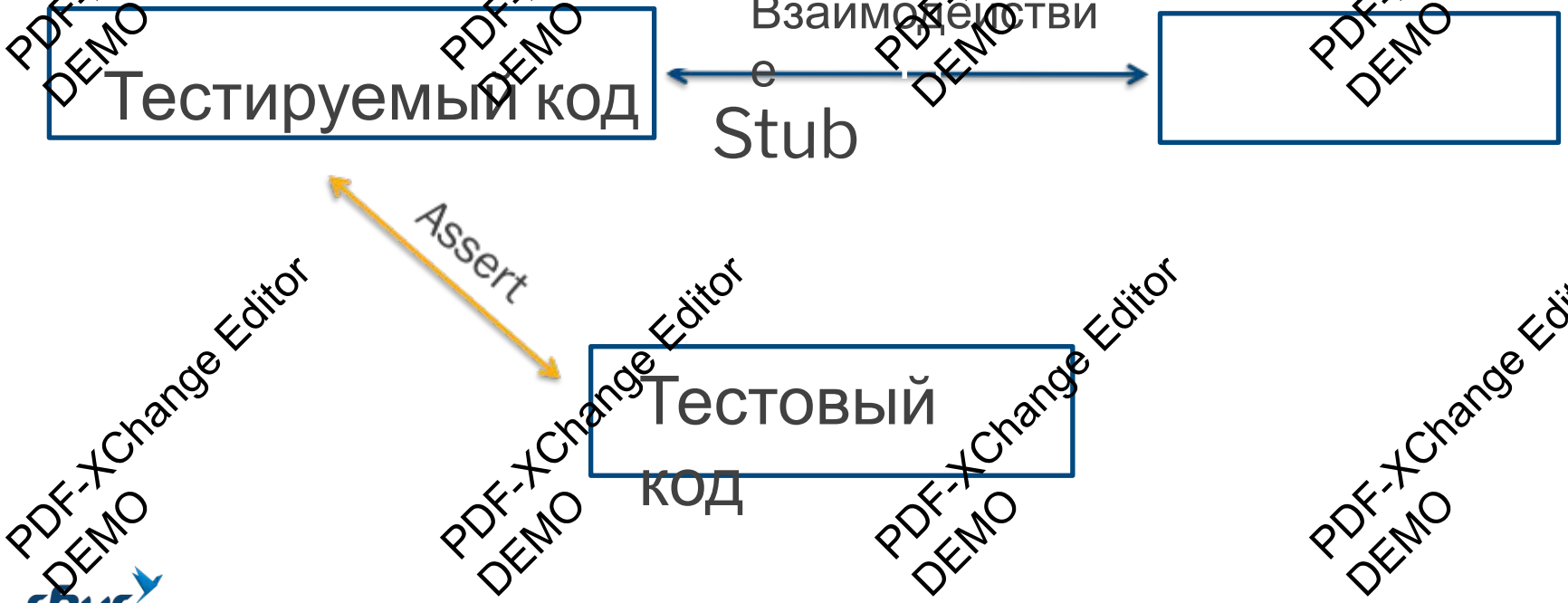
1. Arrange
2. Act
3. Assert

Понятие поддельных объектов

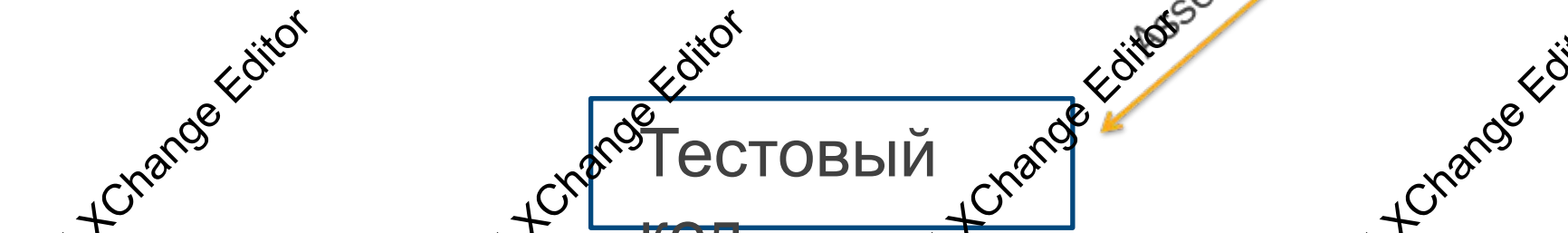
Поддельные реализации объектов



Stub-объект



Mock-объект



Слои тестирования

Модифицированный исходный код

Шов предварительной обработки

Машинный код

Шов этапа компиляции

Исполняемый файл

Компоновочный шов

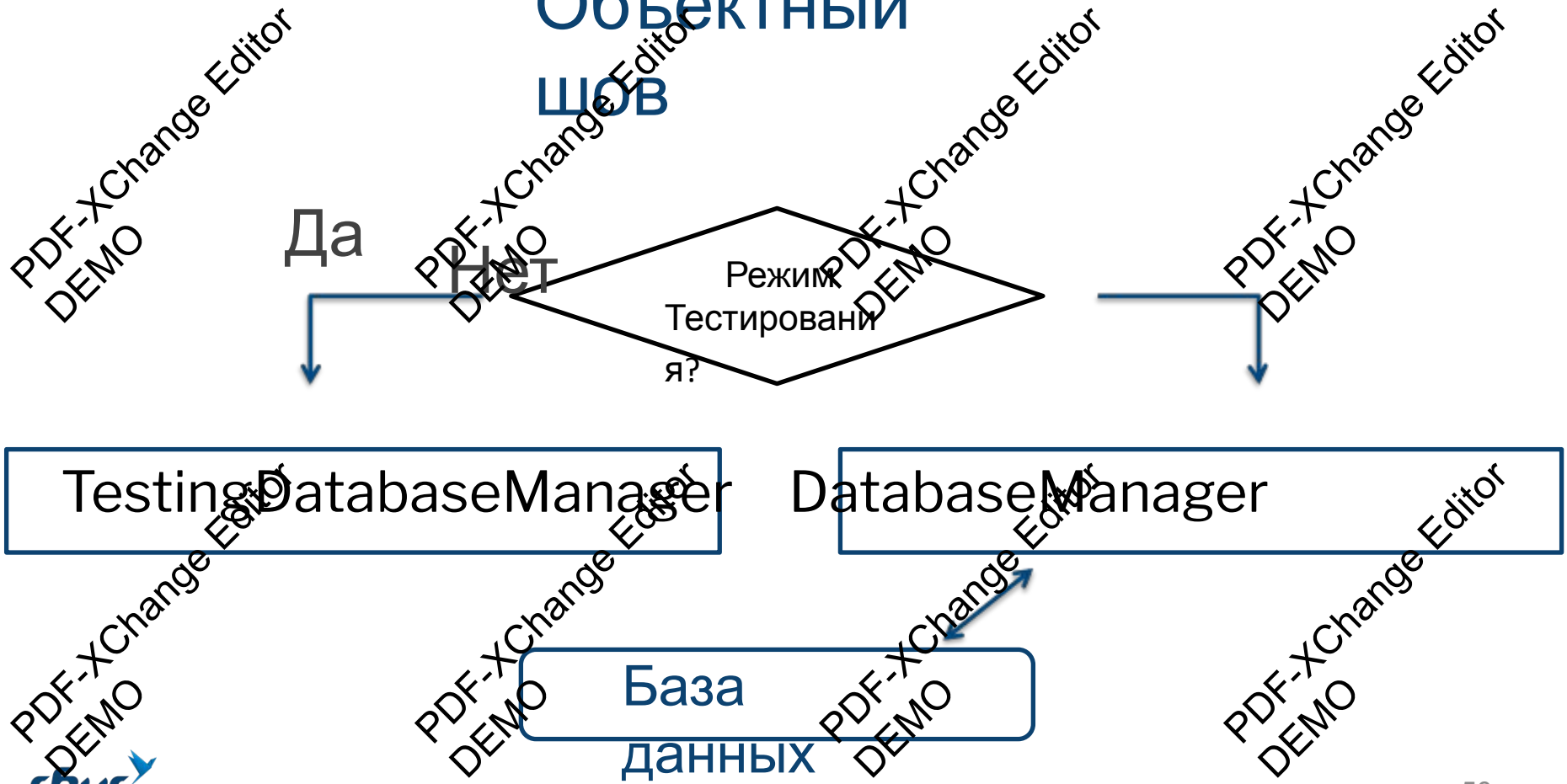
Выполнение приложения

Объектный шов

Объектный шов

- Разрешающая точка находится на этапе создания объектов
- Выбор стратегии выполнения действия в зависимости от используемого объекта

Объектный шов



Использование для разрыва зависимости

“Выделить и переопределить”

Выделение зависимости

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
         WebService     webService;
};
```

Выделение зависимости

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
         WebService     webService;
};
```

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
            ...

         return IsValid( ename );
      }

   protected:
      bool IsValid( std::string ename ) {
         return dbManager.IsValid( ename );
      }

   private:
      DatabaseManager dbManager;
};
```

Переопределение зависимости

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         ...

         return IsValid( ename );
      }

   protected:
      virtual bool IsValid( std::string ename ) {
         return dbManager.IsValid( ename );
      }

   private:
      DatabaseManager dbManager;
};
```

Переопределение зависимости

наследование

```
class TestingEntryAnalyzer :
   public EntryAnalyzer {

   public:
      bool willBeValid;

   private:
      bool IsValid( std::string ename ) override {
         return willBeValid;
      }
};
```

```
class EntryAnalyzer
{
public:
   bool Analyze( std::string ename ) {
      ...

      return IsValid( ename );
   }

protected:
   virtual bool IsValid( std::string ename ) {
      return dbManager.IsValid( ename );
   }

private:
   DatabaseManager dbManager;
};
```

Переопределение зависимости наследование

тестируемый класс

```
class TestingEntryAnalyzer :
   public EntryAnalyzer {

   public:
      bool willBeValid;

   private:
      bool IsValid( std::string ename ) override {
         return willBeValid;
      }
};
```

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
            ...

         return IsValid( ename );
      }

   protected:
      virtual bool IsValid( std::string ename ) {
         return dbManager.IsValid( ename );
      }

   private:
      DatabaseManager dbManager;
};
```

Переопределение зависимости

тестируемый класс

наследование

внедрение зависимости

```
class TestingEntryAnalyzer :
   public EntryAnalyzer {

   public:
      bool willBeValid;

   private:
      bool IsValid( std::string ename ) override {
         return willBeValid;
      }
};
```

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         ...

         return IsValid( ename );
      }

   protected:
      virtual bool IsValid( std::string ename ) {
         return dbManager.IsValid( ename );
      }

   private:
      DatabaseManager dbManager;
};
```

Тестирование возвращаемого значения

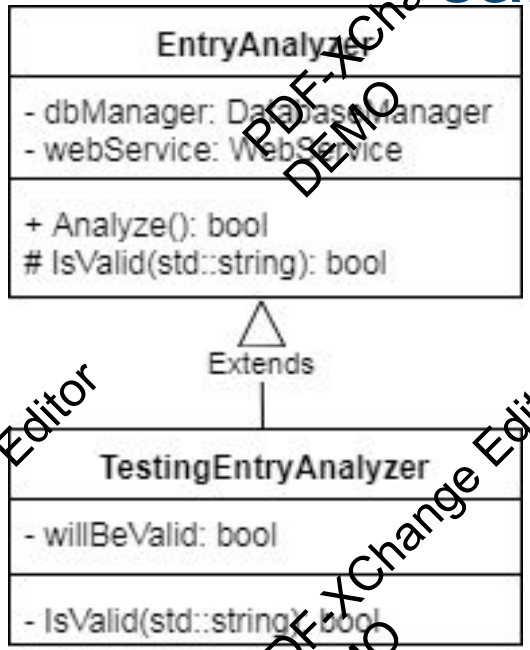
```
TEST_F( EntryAnalyzerTest,
   Analyze_ValidEntryName_ReturnsTrue)
{
   TestingEntryAnalyzer ea;
   ea.willBeValid = true;

   bool result = ea.Analyze( "valid_entry_name" );

   ASSERT_EQ( result, true );
}
```

Тестирование возвращаемого значения

```
TEST_F( EntryAnalyzerTest,
   Analyze_ValidEntryName_ReturnsTrue)
{
   TestingEntryAnalyzer ea;
   ea.willBeValid = true;

   bool result = ea.Analyze( "valid_entry_name" );

   ASSERT_EQ( result, true );
}
```

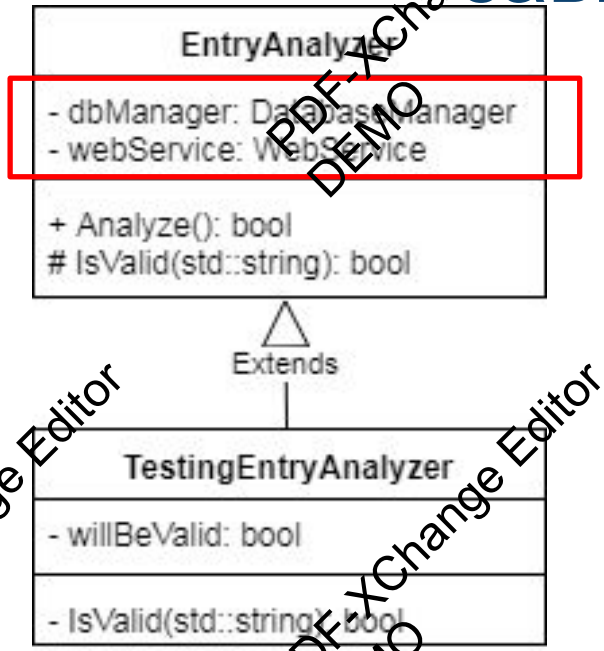
```
class TestingEntryAnalyzer :
   public EntryAnalyzer {

   public:
      bool  willBeValid;

   private:
      bool IsValid( std::string ename ) override {
         return willBeValid;
      }
};
```

Возможные проблемы при переопределении зависимости




```
class EntryAnalyzer {
   public:
      EntryAnalyzer() {
      }

      ...

   private:
      DatabaseManager dbManager;
      WebService webService;
};

class TestingEntryAnalyzer :
   public EntryAnalyzer {

   public:
      TestingEntryAnalyzer() {
      }
      bool willBeValid;
};
```

Возможные проблемы при переопределении зависимости

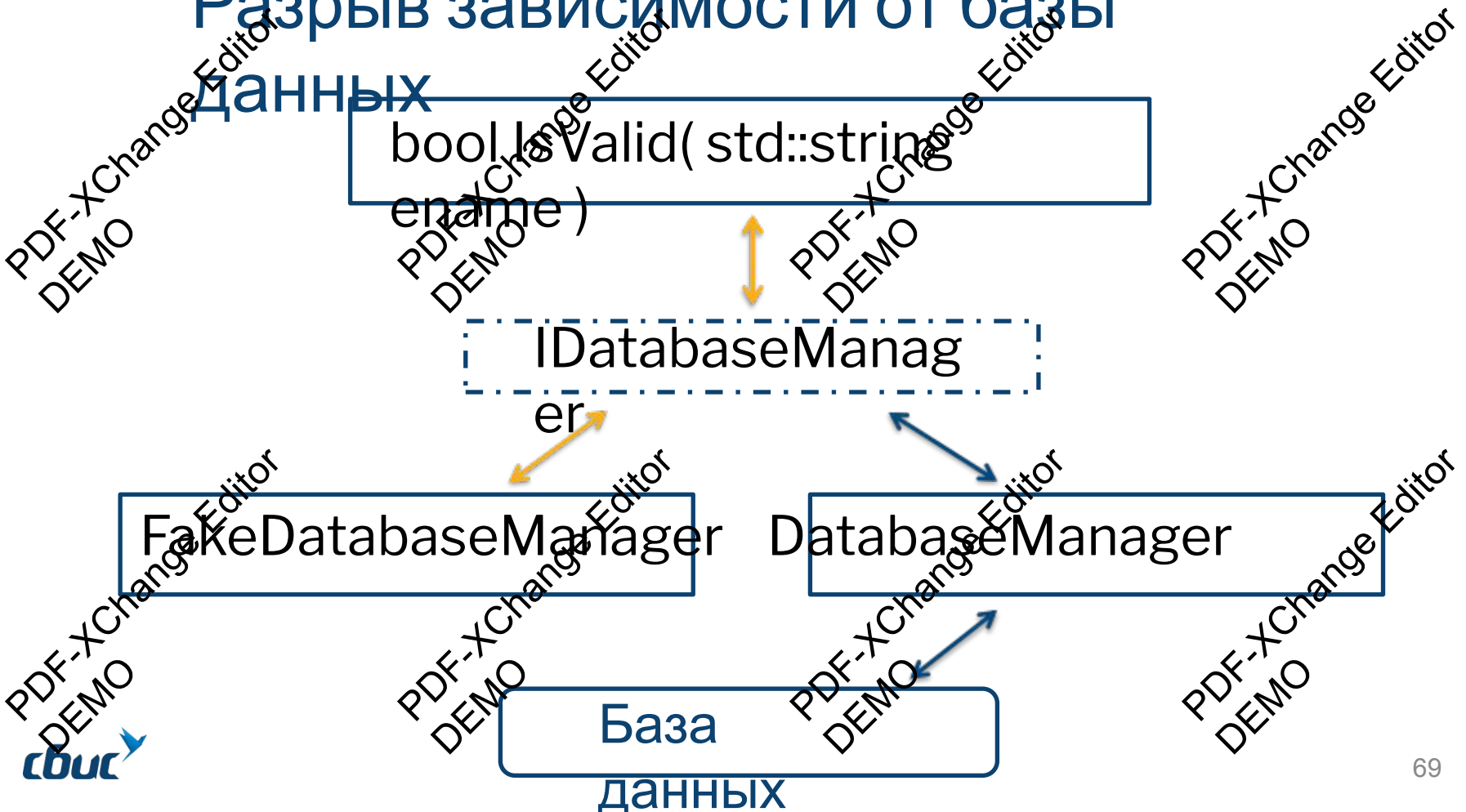



```
class EntryAnalyzer {
  public:
    EntryAnalyzer() {
    }

    ...

  private:
    DatabaseManager dbManager;
    WebService webService;
};

class TestingEntryAnalyzer :
  public EntryAnalyzer {

  public:
    TestingEntryAnalyzer() {
    }
    bool willBeValid;
};
```

Использование setter для разрыва зависимости

Параметризация конструктора.

Разрыв зависимости от базы данных



Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string code ) = 0;
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string ename ) = 0;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

   public:
      bool WillBeValid;

      FakeDatabaseManager( bool will_be_valid )
         WillBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) {
         return WillBeValid;
      }
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string ename ) = 0;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

   public:
      bool WillBeValid;

      FakeDatabaseManager( bool will_be_valid )
         WillBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) {

         return WillBeValid;
      }
};
```

```
class DatabaseManager :
   public IDatabaseManager {

   public:
      bool IsValid( std::string ename ) {

         /* сложная логика, включающая
         операции чтения из базы данных*/
      }
};
```

Вместо конкретной реализации – интерфейс

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error" + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
         WebService     webService;
};
```

Вместо конкретной реализации – интерфейс

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
      WebService        webService;
};
```

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() :
         pDbManager( std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager>   pDbManager;
      WebService                          webService;
};
```

Вместо конкретной реализации – интерфейс

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
      WebService        webService;
};
```

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() :
         pDbManager( std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager>   pDbManager;
                        WebService        webService;
};
```

Где появился баг?

Вместо конкретной реализации – интерфейс

```
class EntryAnalyzer {
   public:
      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return dbManager.IsValid( ename );
      }
   private:
      DatabaseManager   dbManager;
      WebService        webService;
};
```

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() :
         pDbManager( std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager>   pDbManager;
                                 WebService   webService;
};
```

Утечка памяти!

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string email ) = 0;
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
    public:
        virtual bool IsValid( std::string email ) = 0;
        virtual ~IDatabaseManager() = default;
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string ename ) = 0;
      virtual ~IDatabaseManager() = default;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

   public:
      bool WillBeValid;

      FakeDatabaseManager( bool will_be_valid )
         WillBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) override {

         return WillBeValid;
      }
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string ename ) = 0;
      virtual ~IDatabaseManager() = default;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

   public:
      bool WillBeValid;

      FakeDatabaseManager( bool will_be_valid )
         WillBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) override {

         return WillBeValid;
      }
};
```

```
class DatabaseManager :
   public IDatabaseManager {

   public:
      bool IsValid( std::string ename ) override {

         /* сложная логика, включающая
         операции чтения из базы данных*/
      }
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
public:
   virtual bool IsValid( std::string ename ) = 0;
   virtual ~IDatabaseManager() = default;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

public:
   bool WillBeValid;

   FakeDatabaseManager( bool will_be_valid )
      WillBeValid( will_be_valid ) {
   }

   bool IsValid( std::string ename ) override {

      return WillBeValid;
   }
};
```

```
class DatabaseManager :
   public IDatabaseManager {

public:
   bool IsValid( std::string ename ) override {

      /* сложная логика, включающая
      операции чтения из базы данных*/
   }
};
```

Разрыв зависимости от базы данных

```
class IDatabaseManager {
   public:
      virtual bool IsValid( std::string ename ) = 0;
      virtual ~IDatabaseManager() = default;
};
```

```
class FakeDatabaseManager :
   public IDatabaseManager {

   public:
      bool WillBeValid;

      FakeDatabaseManager( bool will_be_valid )
         WillBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) override {

         return WillBeValid;
      }
};
```

```
class DatabaseManager :
   public IDatabaseManager {

   public:
      bool IsValid( std::string ename ) override {

         /* сложная логика, включающая
         операции чтения из базы данных*/
      }
};
```

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_db_mng ) :
         pDbManager( std::move( p_db_mng ) ) {
      }

      bool Analyze( std::string ename ) {
      ...


         return pDbManager->IsValid( ename ) ;
      }

   private:
      std::unique_ptr<DatabaseManager> pDbManager;
          ...
};
```

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_db_mng ) :
         pDbManager( std::move( p_db_mng ) ) {
      }

      bool Analyze( std::string ename ) {
      ...


         return pDbManager->IsValid( ename ) ;
      }

   private:
      std::unique_ptr<DatabaseManager> pDbManager;
         ...
};
```

Внедрение зависимости

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_db_mng ) :
         pDbManager( std::move( p_db_mng ) ) {
      }

      bool Analyze( std::string ename ) {
      ...


         return pDbManager->IsValid( ename ) ;
      }

   private:
      std::unique_ptr<DatabaseManager> pDbManager;
         ...
};
```

Внедрение зависимости

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() : pDbManager(
      std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() : pDbManager(
      std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

```
class EntryAnalyzer {
public:
   EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_mng ) :
       pDbManager( std::move( p_mng ) )
   {
   }

   bool Analyze( std::string ename ) {

      ...

   }
private:
   std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() : pDbManager(
      std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

```
class EntryAnalyzer {
public:
   EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_mng =
   std::make_unique<DatabaseManager>() ) :
       pDbManager( std::move( p_mng ) )
   {
   }

   bool Analyze( std::string ename ) {

      ...

   }
private:
   std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

Внедрение зависимости

```
class EntryAnalyzer {
   public:
      EntryAnalyzer() : pDbManager(
      std::make_unique<DatabaseManager>() ) {
      }

      bool Analyze( std::string ename ) {
         if( ename.size() < 2 ) {
            webService.LogError( "Error: " + ename );
            return false;
         }

         return pDbManager->IsValid( ename );
      }
   private:
      std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
};
```

```
class EntryAnalyzer {
public:
   EntryAnalyzer( std::unique_ptr<IDatabaseManager> &&p_mng =
   std::make_unique<DatabaseManager>() ) :
         pDbManager( std::move( p_mng ) )
   {
   }

   bool Analyze( std::string ename ) {

      ...

   }
private:
   std::unique_ptr<IDatabaseManager> pDbManager;
                              WebService webService;
                                                   };
```

Тестирование возвращаемого значения

```
TEST_F( EntryAnalyzerTest, Analyze_ValidEntryName_ReturnsTrue )
{
    EntryAnalyzer ea( std::make_unique<FakeDatabaseManager>( true ) );

    bool result = ea.Analyze( "valid_entry_name" );

    ASSERT_EQ( result, true );
}
```

Тестирование возвращаемого значения

```
TEST_F( EntryAnalyzerTest, Analyze_ValidEntryName_ReturnsTrue )

   EntryAnalyzer ea( std::make_unique<FakeDatabaseManager>( true ) );

   bool result = ea.Analyze( "valid_entry_name" );

   ASSERT_EQ( result, true );
}

class FakeDatabaseManager : public IDatabaseManager {
   public:
      bool willBeValid;

      FakeDatabaseManager( bool will_be_valid ) : willBeValid( will_be_valid ) {
      }

      bool IsValid( std::string ename ) override {
         return willBeValid;
      }
};
```

Разбор на примере

Исходный код

```
class Analyzer {
public:
    int Analyze() {
        if( false == WinAPIFind() ) return 1;
        if( false == WinAPIProc() ) return 2;


        return 0;
    }

};
```

Исходный код

```
class Analyzer {
public:
  int Analyze() {
    if( false == WinAPIFind() ) return 1;
    if( false == WinAPIProc() ) return 2;

    return 0;
  }
};
```

Локализация внешних зависимостей

```
class Analyzer {
public:
   int Analyze() {
      if( false == WinAPIFind() ) return 1;
      if( false == WinAPIProc() ) return 2;

      return 0;
   }
};
```

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

Подготовка к разрыву зависимости

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

Подготовка к разрыву зависимости

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
   virtual ~Analyzer() = default;
protected:
   virtual bool Find() {
      return WinAPIFind();
   }
   virtual bool Proc() {
      return WinAPIProc();
   }
};
```

Вводим класс-наследник

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
   virtual ~Analyzer() = default;
protected:
   virtual bool Find() {
      return WinAPIFind();
   }
   virtual bool Proc() {
      return WinAPIProc();
   }
};
```

```
class TestingAnalyzer : public Analyzer
{
public:
   bool findApiCallWillBeReturned;
   bool procApiCallWillBeReturned;

private:
   virtual bool Find() override
   {
      return findApiCallWillBeReturned;
   }

   virtual bool Proc() override
   {
      return procApiCallWillBeReturned;
   }
};
```

Вводим класс-наследник

```
class Analyzer {
public:
    int Analyze() {
        if( false == Find() ) return 1;
        if( false == Proc() ) return 2;

        return 0;
    }
    virtual ~Analyzer() = default;
protected:
    virtual bool Find() {
        return WinAPIFind();
    }
    virtual bool Proc() {
        return WinAPIProc();
    }
};
```

```
class TestingAnalyzer : public Analyzer
{
public:
    bool findApiCallWillBeReturned;
    bool procApiCallWillBeReturned;

private:
    virtual bool Find() override
    {
        return findApiCallWillBeReturned;
    }

    virtual bool Proc() override
    {
        return procApiCallWillBeReturned;
    }
};
```

Вводим класс-наследник

```
class Analyzer {
public:
    int Analyze() {
        if( false == Find() ) return 1;
        if( false == Proc() ) return 2;

        return 0;
    }
    virtual ~Analyzer() = default;
protected:
    virtual bool Find() {
        return WinAPIFind();
    }
    virtual bool Proc() {
        return WinAPIProc();
    }
};
```

```
class TestingAnalyzer : public Analyzer
{
public:
    bool findApiCallWillBeReturned;
    bool procApiCallWillBeReturned;

private:
    virtual bool Find() override
    {
        return findApiCallWillBeReturned;
    }

    virtual bool Proc() override
    {
        return procApiCallWillBeReturned;
    }
};
```

Покрываем существующую логику работы класса юнит-тестами

```
Analyze_WhenFindFails_Returns1 (
{
    TestingAnalyzer tf ;
    tf.findApiCallWillBeReturned  =
false;
    int test_ret =
tf.Analyze();
    int etalon_ret =
1; ASSERT_EQ( test_ret,
etalon_ret);
}
```


101

Покрываем существующую логику работы класса юнит-тестами

```
Analyze_WhenFindFails_Returns1 (
{
    TestingAnalyzer tf ;
    tf.findApiCallWillBeReturned  =
false;
    int test_ret  =
tf.Analyze();
    int etalon_ret  =
1; ASSERT_EQ( test_ret,
etalon_ret);
}
```

```
Analyze_WhenProcFails_Returns2 (
{
    TestingAnalyzer tf ;
    tf.findApiCallWillBeReturned  =
true;
    tf.procApiCallWillBeReturned  =
false;
    int test_ret  =
tf.Analyze();
    int etalon_ret  =
2; ASSERT_EQ( test_ret,
etalon_ret);
}
```

Покрываем существующую логику работы класса юнит-тестами

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret =
tf.Analyze();
   int etalon_ret =
1; ASSERT_EQ( test_ret,
etalon_ret);
}
```

```
Analyze_WhenProcFails_Returns2 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
true;
   tf.procApiCallWillBeReturned  =
   int test_ret =
false;
tf.Analyze();
   int etalon_ret =
2; ASSERT_EQ( test_ret,
etalon_ret);
}
```

```
Analyze_WhenSucceed_Returns0 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
true;
   tf.procApiCallWillBeReturned  =
   int test_ret =
true;
tf.Analyze();
   int etalon_ret =
0; ASSERT_EQ( test_ret,
etalon_ret);
}
```

Вносим изменения в боевой класс

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

Вносим изменения в боевой класс

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

```
class Analyzer
{
public:
   int Analyze()
   {
      WindowsProcessWrapper wr;
      return wr.Process();
   }
   virtual ~Analyzer() = default;
};
```

Вносим изменения в боевой класс

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

```
class Analyzer
{
public:
   int Analyze()
   {
      WindowsProcessor wr;
      return wr.Process();
   }
   virtual ~Analyzer() = default;
};
```

```
class WindowsProcessor()
{
public:
   int Process()
   {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
protected:
   virtual bool Find()
   {
      return WinAPIFind();
   }

   virtual bool Proc()
   {
      return WinAPIProc();
   }
};
```

Поиск внешних зависимостей

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

Внешние зависимости

```
class Analyzer
{
public:
   int Analyze()
   {
      WindowsProcessor wr;
      return wr.Process();
   }
   virtual ~Analyzer() = default;
};
```

```
class WindowsProcessor()
{
public:
   int Process()
   {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
protected:
   virtual bool Find()
   {
      return WinAPIFind();
   }

   virtual bool Proc()
   {
      return WinAPIProc();
   }
};
```

Поиск внешних зависимостей

```
class Analyzer {
public:
   int Analyze() {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
private:
   bool Find() {
      return WinAPIFind();
   }
   bool Proc() {
      return WinAPIProc();
   }
};
```

```
class Analyzer
{
public:
   int Analyze()
   {
      WindowsProcessor wr;
      return wr.Process();
   }
   virtual ~Analyzer() = default;
};
```

Внешняя зависимость

```
class WindowsProcessor()
{
public:
   int Process()
   {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;

      return 0;
   }
protected:
   virtual bool Find()
   {
      return WinAPIFind();
   }

   virtual bool Proc()
   {
      return WinAPIProc();
   }
};
```

Подготовка к разрыву зависимости

```
class IWindowsProcessor
{
   virtual int Process() = 0; virtual ~IWindowsProcessor() = default;
};
```


PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO
PDF-XChange Editor DEMO

Подготовка к разрыву зависимости

```
class
IWindowsProcessor
{
   virtual int Process() =
0; virtual ~IWindowsProcessor() =
default;
};
```

```
class WindowsProcessor :
   public
IWindowsProcessor
{
public:
   int Process() override
   {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;
      return 0;
   }

   virtual ~WindowsProcessor() =
default;
protected:
   virtual bool Find()
   {
      return WinAPIFind();
   }

   virtual bool Proc()
   {
      return WinAPIProc();
   }
};
```

Подготовка к разрыву зависимости

```
class
IWindowsProcessor
{
   virtual int Process() =
0; virtual ~IWindowsProcessor() =
default;
};
```

```
class WindowsProcessor :
   public
IWindowsProcessor
{
public:
   int Process() override
   {
      if( false == Find() ) return 1;
      if( false == Proc() ) return 2;
      return 0;
   }

   virtual ~WindowsProcessor() =
default;
protected:
   virtual bool Find()
   {
      return WinAPIFind();
   }

   virtual bool Proc()
   {
      return WinAPIProc();
   }
};
```

```
class TestingWindowsProcessor :
   public WindowsProcessor
{
public:
   bool findApiCallWillBeReturned;
   bool procApiCallWillBeReturned;
private:
   virtual bool Find()
override
   {
      return
findApiCallWillBeReturned;
   }
   virtual bool Proc()
override
   {
      return
procApiCallWillBeReturned;
};
```

Подготовка к разрыву зависимости

```
class Analyzer
{
public:
    int Analyze()
    {
        WindowsProcessor wp;
        return wr.Process();
    }
   virtual ~Analyzer() = default;
};
```

Подготовка к разрыву зависимости

```
class Analyzer
{
public:
    int Analyze()
    {
        WindowsProcessor wp;
        return wr.Process();
    }
   virtual ~Analyzer() = default;
};
```

```
class
Analyzer
{
public:
    Analyzer()
:       pWindowsProcessor ( nullptr
)   {
        pWindowsProcessor  =
std::make_unique<WindowsProcessor>();
    }
    int
Analyze()
    {
        return
pWindowsProcessor->Process();
    }
   virtual ~Analyzer() =
default;
private:
    std::unique_ptr<IWindowsProcessor>
pWindowsProcessor;
};
```

Внедрение зависимости

```
class
Analyzer
{
public:
   Analyzer()
:      pWindowsProcessor ( nullptr
){
   pWindowsProcessor  =
std::make_unique<WindowsProcessor>();
   }

   int
Analyze()
   {
      return
pWindowsProcessor->Process();
   }
   virtual ~Analyzer() =
default;
private:
   std::unique_ptr<IWindowsProcessor>
pWindowsProcessor_;
};
```

Внедрение зависимости

```
class
Analyzer
{
public:
   Analyzer()
:      pWindowsProcessor ( nullptr
)  {
      pWindowsProcessor  =
std::make_unique<WindowsProcessor>();
   }

   int
Analyze()
   {
      return
pWindowsProcessor->Process();
   }
   virtual  ~Analyzer() =
default;
private:
   std::unique_ptr<IWindowsProcessor>
pWindowsProcessor;
};
```

```
class
Analyzer
{
public:
   Analyzer()
:      pWindowsProcessor ( nullptr
)  {
      pWindowsProcessor  =
std::make_unique<WindowsProcessor>();
   }

   void SetWindowsProcessor (
      std::unique_ptr<IWindowsProcessor  > &&wp
)
   {
      pWindows Processor  = std::move( wp
);  }

   int
Analyze()
   {
      int ret =
pWindowsProcessor->Process();
      return
ret;
   }
   virtual  ~Analyzer() =
default;
private:
   std::unique_ptr<IWindowsProcessor>
pWindowsProcessor;
};
```

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
}etalon_ret );
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f ;
   std::unique_ptr<TestingWindowsProcessor > p_twp  =
      std::make_unique<TestingWindowsProcessor >();
   p_twp->findApiCallWillBeReturned  = false;
   SetWindowsProcessor ( std::move( p_twr )  );

   int test_ret  =
tf.Analyze();
   int etalon_ret  = 1;
   ASSERT_EQ ( test_ret ,
}etalon_ret );
```

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor > p_twp  =
      std::make_unique<TestingWindowsProcessor >();
   p_twp->findApiCallWillBeReturned  = false;
   SetWindowsProcessor ( std::move( p_twr )  );

   int test_ret  =
tf.Analyze();
   int etalon_ret  = 1;
   ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

Создаем боевой объект

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
etalon_ret );
}
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor> p_twp =
      std::make_unique<TestingWindowsProcessor>();
   p_twp->findApiCallWillBeReturned  = false;
   SetWindowsProcessor ( std::move( p_twr )  );

   int test_ret =
tf.Analyze();
   int etalon_ret = 1;
   ASSERT_EQ ( test_ret ,
etalon_ret );
}
```

Создаем stub-объект

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor> p_twp
=
      std::make_unique<TestingWindowsProcessor>();
   p_twp->findApiCallWillBeReturned = false;
   SetWindowsProcessor ( std::move( p_twr )  );
   int test_ret =
tf.Analyze();
   int etalon_ret  = 1;
   ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

Настраиваем stub-объект

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor > p_twp =
      std::make_unique<TestingWindowsProcessor >();
   p_twp->findApiCallWillBeReturned  = false;
   f.SetWindowsProcessor( std::move( p_twr ) );

   int test_ret =
tf.Analyze();
   int etalon_ret = 1;
   ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

Внедряем зависимость

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor > p_twp  =
      std::make_unique<TestingWindowsProcessor >();
   p_twp->findApiCallWillBeReturned  = false;
   f.SetWindowsProcessor( std::move( p_twr ) );

   int test_ret = tf.Analyze();

   int etalon_ret  = 1;
   ASSERT_EQ ( test_ret ,
etalon_ret);
}
```

Проводим тестируемое действие

Обновление тестов

```
Analyze_WhenFindFails_Returns1 (
{
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
false;
   int test_ret  =
tf.Analyze();
   int etalon_ret  =
1; ASSERT_EQ( test_ret,
}etalon_ret);
```

```
Analyze_WhenFindFails_Returns1 (
{
   Analyzer f;
   std::unique_ptr<TestingWindowsProcessor > p_twp  =
      std::make_unique<TestingWindowsProcessor >();
   p_twp->findApiCallWillBeReturned  = false;
   f.SetWindowsProcessor( std::move( p_twr ) );

   int test_ret =
tf.Analyze();
   int etalon_ret = 1;
   ASSERT_EQ( test_ret, etalon_ret);
}
```

Оцениваем результат

Обновление тестов

```
Analyze_WhenProcFails_Returns2 (
{
    TestingAnalyzer tf ;
    tf.findApiCallWillBeReturned  =
true;
    tf.procApiCallWillBeReturned  =
    int test_ret =
false;
tf.Analyze();
    int etalon_ret =
2; ASSERT_EQ( test_ret,
etalon_ret);
}
```

```
Analyze_WhenProcFails_Returns2 (
{
    Analyzer f ;
    std::unique_ptr<TestingWindowsProcessor > p_twp
=
        std::make_unique<TestingWindowsProcessor >();
    p_twr->findApiCallWillBeReturned  = true;
    p_twr->procApiCallWillBeReturned  = false;
    f.SetWindowsProcessor ( std::move( p_twp ) );
    int test_ret =
t.Analyze();
    int etalon_ret =
2; ASSERT_EQ( test_ret,
etalon_ret);
}
```

Обновление тестов

```
Analyze_WhenSucceed_Returns0 (
)
   TestingAnalyzer tf ;
   tf.findApiCallWillBeReturned  =
true;
   tf.procApiCallWillBeReturned  =
true;
   int test_ret =
tf.Analyze();
   int etalon_ret = 0;
   ASSERT_EQ( test_ret,
etalon_ret);
}
```

```
Analyze_WhenSucceeded_Returns0 (
)
   Analyzer f ;
   std::unique_ptr<TestingWindowsProcessor > p_twp =
      std::make_unique<TestingWindowsProcessor >();
   p_twr->findApiCallWillBeReturned  = true;
   p_twr->procApiCallWillBeReturned  = true;
   p_twr->thirdFunclWillBeReturned  = false;
   f.SetWindowsProcessor ( std::move( p_twp ) );
   int test_ret =
t.Analyze();
   int etalon_ret = 0;
   ASSERT_EQ( test_ret,
etalon_ret);
}
```

Общие рекомендации и практические приемы

Рекомендации

- Общий принцип: рефакторинг кода с целью
  - локализация мест обращения к внешним

Рекомендации

- Общий принцип: рефакторинг кода с целью зависимостей мест обращения к внешним
- Сохраняем сигнатуры методов при внесении изменений

Практические приемы

Практические приемы

- Один тест — один результат работы

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_TooShortEntryName_ReturnsFalseLoggsErrorToWebServer )
{
   TestingEntryAnalyzer ea;

   bool is_valid = ea.Analyze( "e" );
   std::string s_err_msg = ea.lastErrMsg;

   bool b_test_result = false == is_valid && "Error: e" == s_err_msg );
   ASSERT_EQ( test_result, true );
}
```

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_TooShortEntryName_ReturnsFalseLoggsErrorToWebServer )
{
   TestingEntryAnalyzer ea;

   bool is_valid = ea.Analyze( "e" );
   std::string s_err_msg = ea.lastErrMsg;

   bool b_test_result = false == is_valid && "Error: e" == s_err_msg );
   ASSERT_EQ( test_result, true );
}
```

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_TooShortEntryName_ReturnsFalseLoggsErrorToWebServer )
{
   TestingEntryAnalyzer ea;

   bool is_valid = ea.Analyze( "e" );
   std::string s_err_msg = ea.lastErrMsg;

   bool b_test_result = false == is_valid && "Error: e" == s_err_msg );
   ASSERT_EQ( test_result, true );
}
```

1

2

Практические приемы


```
TEST_F( EntryAnalyzerTest,
    Analyze_TooShortEntryName_ReturnsFalseLoggsErrorToWebServer )
{
    TestingEntryAnalyzer ea;

    bool is_valid = ea.Analyze( "e" );
    std::string s_err_msg = ea.lastErrMsg;

    bool b_test_result = false == is_valid && "Error: e" == s_err_msg );
    ASSERT_EQ( test_result, true );
}
```

1

?

2

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_TooShortEntryName_ReturnsFalseAndSetsErrorMsg )
{
   TestingEntryAnalyzer ea;

   bool is_valid = ea.Analyze( "e" );
   std::string s_err_msg = ea.lastErrMsg;

   ASSERT_EQ( is_valid, false );
   ASSERT_EQ( s_err_msg, "Error: e" );
}
```

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_TooShortEntryName_ReturnsFalseAndLoggsErrorToWebServer )
{
   TestingEntryAnalyzer ea;

   bool is_valid = ea.Analyze( "e" );
   std::string s_err_msg = ea.lastErrMsg;

   ASSERT_EQ( is_valid, false );
   ASSERT_EQ( s_err_msg, "Error: e" );
}
```

1

2

?

Практические приемы

- Один тест — один результат работы

Практические приемы

- Один тест — один результат работы
- Тестируем только публичные методы

Практические приемы

- Один тест — один результат работы
- Тестируем только публичные методы
  - Нет ветвления
    - операторы: `switch`, `if`, `else`
    - циклы: `for`, `while`, `std::for_each`

Практические приемы

- Один тест — один результат работы
- Тестируем только публичные методы
  - Нет ветвления
    - операторы: `switch`, `if`, `else`
    - циклы: `for`, `while`, `std::for_each`
- Юнит тест — последовательность вызовов методов + assert

Практические приемы

- Один тест — один результат работы
- Тестируем только публичные методы
- Нет ветвления
  - операторы: `switch`, `if`, `else`
  - циклы: `for`, `while`, `std::for_each`
- Юнит тест — последовательность вызовов методов + assert
- Используем фабрики

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );

         ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
   ...
}
```

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );

         ...
}
```

```
EntryAnalyzer ea(
   std::string s_db_name );
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
   ...
}
```

Практические приемы


```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );
   ...
}
```

```
EntryAnalyzer ea(
   std::string s_db_name );
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
   ...
}
```

Практические приемы

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );
      ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
      ...
}
```

```
EntryAnalyzer ea(
   std::string s_db_name );
```

```
std::unique_ptr<EntryAnalyzer> CreateDefaultEntryAnalyzer()
{
   std::string s_db_name( "DummmyDB" );
   return std::make_unique<EntryAnalyzer>( s_db_name );
}
```

Практические приемы

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );
         ...
}
```

```
EntryAnalyzer ea(
   std::string s_db_name );
```

```
std::unique_ptr<EntryAnalyzer> CreateDefaultEntryAnalyzer()
{
   std::string s_db_name( "DummmyDB" );
   return std::make_unique<EntryAnalyzer>( s_db_name );
}

TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
 std::unique_ptr<EntryAnalyzer> p_ea =
    CreateDefaultEntryAnalyzer;
 p_ea->Analyze( "name1" );
         ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
         ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
 std::unique_ptr<EntryAnalyzer> p_ea =
    CreateDefaultEntryAnalyzer;
 p_ea->Analyze( "name2" );
         ...
}
```

Практические приемы

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name1" );
      ...
}
```

```
EntryAnalyzer ea(
   std::string s_db_name );
```

```
std::unique_ptr<EntryAnalyzer> CreateDefaultEntryAnalyzer()
{
   std::string s_db_name( "DummmyDB" );
   return std::make_unique<EntryAnalyzer>( s_db_name );
}

TEST_F( EntryAnalyzerTest,
   Analyze_Condition1_ReturnsTrue )
{
   std::unique_ptr<EntryAnalyzer> p_ea =
      CreateDefaultEntryAnalyzer;
   p_ea->Analyze( "name1" );
      ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   EntryAnalyzer ea;
   ea.Analyze( "name2" );
      ...
}
```

```
TEST_F( EntryAnalyzerTest,
   Analyze_Condition2_ReturnsTrue )
{
   std::unique_ptr<EntryAnalyzer> p_ea =
      CreateDefaultEntryAnalyzer;
   p_ea->Analyze( "name2" );
      ...
}
```

Практические приемы

- Один тест — один результат работы
- Тестируем только публичные методы
- Нет ветвления
  - операторы: `switch`, `if`, `else`
  - циклы: `for`, `while`, `std::for_each`
- Юнит тест — последовательность вызовов методов, assert
- Используем фабрики

Где почитать подробнее

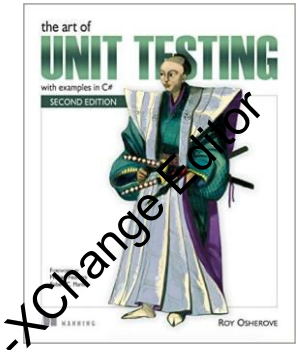
- Майкл Физерс

“Эффективная работа с унаследованным кодом”

Где почитать подробнее

- Майкл Физерс
  “Эффективная работа с унаследованным кодом”

- Roy Osherove
  “The art of unit testing”. 2nd edition



#UFADEVCONF

Спасибо за внимание!

Ястребов Виктор

к.т.н., ведущий разработчик компании “Тензор”

victor.yastrebov1@yandex.ru
http://dc.ufacoder.com - 2018