

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ**

К. А. Кулаков, В. М. Димитров

Инструменты тестирования программного обеспечения

Методические указания для обучающихся
Института математики и информационных технологий

Петрозаводск
Издательство ПетрГУ
2018

УДК 004
ББК 32.973.2
К90

Р е ц е н з е н т:

Р. В. Воронов, доктор технических наук, профессор ПетрГУ

*Издается по решению редакционно-издательского совета
Петрозаводского государственного университета*

Кулаков, Кирилл Александрович.

К90 Инструменты тестирования программного обеспечения [Электронный ресурс] : методические указания для обучающихся Института математики и информационных технологий / К. А. Кулаков, В. М. Димитров ; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. — Электрон. дан. — Петрозаводск : Издательство ПетрГУ, 2018. — 1 электрон. опт. диск (CD-R) ; 12 см. — Систем. требования : PC, MAC с процессором Intel 1.3 ГГц и выше ; Windows, MAC OSX ; 256 Мб ; видеосистема : разрешение экрана 800x600 и выше ; графический ускоритель (опционально) ; мышь или другое аналогичное устройство. — Загл. с этикетки диска.

ISBN 978-5-8021-3411-5

В методических указаниях содержатся практические сведения по использованию популярных инструментов и технологий для планирования, организации и проведения одного из основополагающих этапов разработки программного обеспечения — тестирования.

Пособие предназначено для обучающихся Института математики и информационных технологий направлений.

УДК 004
ББК 32.973.2

ISBN 978-5-8021-3411-5

© Кулаков К. А., Димитров В. М., 2018
© Петрозаводский государственный
университет, 2018

Содержание

Введение	4
§ 1. Программная платформа Qt	5
§ 2. Система контроля версий git	7
§ 3. Платформа GitHub	11
§ 4. Система непрерывной сборки Travis	14
§ 5. Сервис оценки покрытия кода тестами Coveralls	19
§ 6. Сервис непрерывного статического анализа кода SonarCloud	21
§ 7. Лабораторные работы	23
Список литературы	25

Введение

Этап обеспечения качества (в частности, тестирование) в проектах по разработке программного обеспечения (ПО) является одной из важных составляющих успеха проекта в современной индустрии. В связи с этим на практике используются различные программные средства, позволяющие упростить и автоматизировать процесс тестирования.

Практическая часть курсов «Введение в тестирование», «Технология производства программного обеспечения» и «Верификация программного обеспечения», читаемых в Петрозаводском государственном университете для студентов Института математики и информационных технологий, направлена на ознакомление обучающихся с современными инструментами автоматизации, повсеместно применяемыми в индустрии разработки ПО.

Тестирование включает в себя различные этапы с применением различных инструментов:

- 1) планирование работ (Test Management);
- 2) проектирование тестов путем ручной разработки или автоматической генерации (Test Design);
- 3) выполнение тестирования с получением результатов (Test Execution);
- 4) анализ полученных результатов выполнения с целью оценки качества ПО (Test Analysis).

Методические указания предназначены для обучающихся по направлениям «Прикладная математика и информатика», «Информационные системы и технологии» и «Программная инженерия». Освоение материала, представленного в методических указаниях, позволит приобрести компетенции использования некоторых современных инструментов в тестировании на практике.

§ 1. Программная платформа Qt

Для сборки проекта в бинарный код платформа Qt использует систему *qmake*. Конфигурация сборки задается в файле с расширением *pro*. Пример *pro* файла представлен в листинге 1.1.

Листинг 1.1: Пример конфигурационного *pro* файла

```
TEMPLATE = app
CONFIG += console
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += main.c \
          myfunc.c

HEADERS += \
          myfunc.h

QMAKE_CFLAGS += -Wall -Wextra -Werror

# gcov
QMAKE_CFLAGS += -fprofile-arcs -ftest-coverage
LIBS += -lgcov
```

Конфигурационная опция *TEMPLATE* задает тип проекта, который должен собрать *qmake* (*app* — приложение, *lib* — библиотека, *subdirs* — проект состоит из нескольких подпроектов, расположенных в поддиректориях проекта, в каждой поддиректории имеется свой *pro* файл).

Опция *CONFIG* задает основные параметры сборки. Использование «+=» добавляет в конфигурацию соответствующую запись, а «-» исключает из использования. В примере листинга 1.1 в *CONFIG += console*, нет необходимости создавать образ для системы MAC OS (строка *CONFIG -= app_bundle*) и нет необходимости использовать *qt* библиотеку (строка *CONFIG -= qt*).

В опции *SOURCES* перечисляются файлы с исходным кодом

проекта, в опции *HEADERS* — заголовочные файлы (.h). В опции *QMAKE_CFLAGS* задаются опции для компилятора C, в *LIBS* — подключаются дополнительные библиотеки.

Также могут быть указаны следующие опции:

- *DESTDIR* — директория для исполняемого или бинарного файлов;
- *FORMS* — список UI файлов для модуля uic;
- *QT* — специфичные опции для QT;
- *RESOURCES* — список файлов с ресурсами (.rc).

§ 2. Система контроля версий *git*

Система контроля версий кода в современной технологии разработки программного обеспечения является обязательной частью проекта. Ее использование облегчает возврат к предыдущим правкам кода (понятие *commit*), позволяет переключаться между различными версиями кода (понятие *branch*), обмениваться изменениями в коде с другими участниками команды (понятие *push* и *pull*).

Рассмотрим одну из распространенных систем контроля версия — *git*. Общий список файлов, историю их изменений в системе *git* называют репозиторием. Система *git* является распределенной системой контроля версий, что означает, что каждый пользователь имеет полную копию репозитория.

Любое логически законченное изменение кода рекомендуется сопровождать созданием новой отметки об этом в репозитории (выполнять команду *commit*). Данная отметка остается в локальном репозитории, поэтому, для того чтобы загрузить данную отметку или набор отметок в другой репозиторий, необходимо выполнить команду *push*. Чтобы загрузить в локальный репозиторий отметки с удаленного репозитория, выполняется команда *pull*.

Важной составляющей системы *git* является понятие ветвления. Обычно новая ветка (*branch*) создается для того, чтобы реализовать новый функционал или проверить какую-либо гипотезу, связанную, например, с работой алгоритмов. Каждая ветка имеет название. Исходная ветка, которую создает система *git*, называется *master*. Чтобы создать ветку, необходимо выполнить команду *branch*.

После того как были внесены изменения в новую ветку, возникает необходимость перенести эти изменения в предыдущую ветку. Для этого необходимо воспользоваться командой *merge*. Если в предыдущей ветке изменялся тот же файл, что и в текущей, то может возникнуть конфликт версий, с которыми система *git* не может справиться самостоятельно. В этом случае она помечает в файле те части кода, которые она не может соединить в одну, и пользователю необходимо самостоятельно разрешить данные конфликты.

Приведем типичный пример работы с проектом и системой *git* (примеры представлены для консольной утилиты *git*):

- Перейдите в директорию проекта.

- Инициализируйте проект *git*:

git init

- Создайте файл *.gitignore*, в котором указывается список файлов, не включаемых в репозиторий. Обычно в нем указываются генерируемые файлы, временные файлы (создаваемые редакторами кода или другими программами) и др. Примеры *.gitignore* файлов можно найти по ссылке <https://github.com/github/gitignore>.
- Проверьте статус репозитория (должен вывестись список всех неослеживаемых файлов):

git status

- Добавьте все ваши файлы в репозиторий (данной командой добавятся все ваши файлы, кроме тех, что указаны в файле *.gitignore*):

git add -A

- Проверьте статус репозитория (должен вывестись список файлов, доступных для включения в коммит):

git status

- Сделайте коммит в репозиторий:

git commit -m 'Первая версия проекта' -a

- Проверьте статус репозитория (должно появиться сообщение, что нет изменений в репозитории):

git status

- Создайте новый файл с названием *refs.txt*, внесите в него правки.
- Добавьте файл в индекс репозитория:

git add refs.txt

- Сделайте новый коммит:


```
git commit -m 'Добавлен список источников' refs.txt
```

- Список коммитов можно посмотреть, выполнив команду:

```
git log
```

- Создайте новую ветку с названием develop:

```
git branch develop
```

- Список веток можно посмотреть, выполнив команду:

```
git branch
```

- Переключитесь в новую ветку:

```
git checkout develop
```

- Внесите правки в файл refs.txt.
- Сделайте коммит (он должен оказаться в develop):

```
git commit -m 'Добавлен новый источник' refs.txt
```

- Переключитесь в начальную ветку master:

```
git checkout master
```

- Убедитесь, что в данной ветке нет последнего коммита:

```
git log
```

- Выполните слияние с веткой develop:

```
git merge develop
```

- Убедитесь, что в ветке master появился последний коммит, а файл имеет содержание из ветки develop1.
- Внесите правки в файл refs.txt и сделайте коммит в ветке master.
- Переключитесь в ветку develop, измените refs.txt, сделайте коммит в ветке develop, переключитесь в ветку master.

- Выполните слияние с веткой develop:

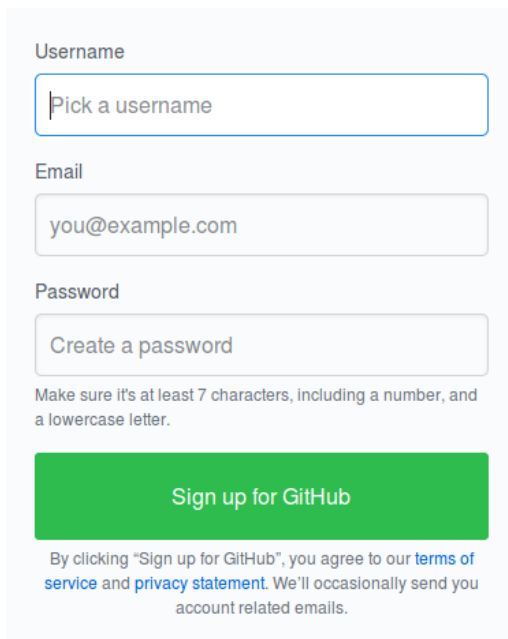
git merge develop

- Система git увидит две разные версии файла и попытается их соединить. В некоторых случаях она не сможет сделать это самостоятельно и предложит пользователю решить какие изменения оставлять, а какие нет.

§ 3. Платформа GitHub

Для ведения проекта (в это понятие включается размещение репозитория с кодом, возможности для ведения ошибок, ведение документации, интеграция с системами тестирования и др.) существует множество онлайн-сервисов. Один из таких сервисов GitHub (<https://github.com>).

Для получения доступа ко всем возможностям необходимо зарегистрироваться в системе. Для этого придумайте входное имя, содержащее только латинские буквы, цифры и тире, пароль и введите адрес электронной почты. На рисунке 1 представлен экран регистрации в системе GitHub. Далее для примеров будем использовать входное имя *demo*.



The image shows a registration form for GitHub. It consists of three input fields: 'Username' with the placeholder 'Pick a username', 'Email' with the placeholder 'you@example.com', and 'Password' with the placeholder 'Create a password'. Below the password field is a note: 'Make sure it's at least 7 characters, including a number, and a lowercase letter.' At the bottom of the form is a large green button labeled 'Sign up for GitHub'. Below the button is a disclaimer: 'By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails.'

Рис. 1. Экран регистрации в системе GitHub

После регистрации в системе необходимо подтвердить введенный адрес электронной почты. Для этого система высылает специальное письмо на данный адрес, в котором указывается ссылка, по которой необходимо пройти. Если этого не сделать, то доступ к функциям системы не будет предоставлен.

Для создания репозитория в меню добавления необходимо выбрать “New repository” (новый репозиторий). На рисунке 2 представлен экран добавления нового репозитория.

The screenshot shows the 'Create a new repository' page on GitHub. At the top, it says 'Create a new repository' and 'A repository contains all the files for your project, including the revision history.' Below this, there are two main input fields: 'Owner' and 'Repository name'. The 'Owner' field is a dropdown menu showing 'vyacheslav-dimitrov'. The 'Repository name' field is an empty text box. Below these fields, there is a note: 'Great repository names are short and memorable. Need inspiration? How about [jubilant-telegram](#).' Underneath is a 'Description (optional)' text area. Further down, there are two radio button options for repository visibility: 'Public' (selected) and 'Private'. Below these are two checkboxes: 'Initialize this repository with a README' and 'Add a license'. At the bottom, there is a green 'Create repository' button.

Рис. 2. Экран добавления нового репозитория в системе GitHub

Обязательным является имя репозитория, которое затем используется для ссылки. Далее для примеров будем использовать имя репозитория *r_demo*. Можно задать два типа репозитория: “public” – открытый для просмотра всеми (в том числе незарегистрированными) пользователями репозиторий и “private” — владелец репозитория самостоятельно выбирает кому доступен данный репозиторий.

Если планируется разместить существующий репозиторий в си-

стеме, то создание дополнительных, на практике рекомендуемых к добавлению в каждый репозиторий, файлов (README.md, .gitignore, LICENSE.md) не требуется.

После нажатия на кнопку “Create repository”, репозиторий будет создан, и пользователь будет перенаправлен на страницу данного репозитория.

Чтобы разместить имеющийся проект в системе GitHub, необходимо создать локальный репозиторий с использованием системы контроля версий git (см. раздел § 2.). Для размещения проекта укажите в системе GitHub ссылку с помощью команды:

```
git remote add origin https://github.com/demo/r_demo.git
```

Затем разместите проект:

```
git push -u origin master
```

После ввода данной команды система потребует ввести входное имя и пароль. Если введенные данные были верны, то страница с репозиторием должны выглядеть примерно так, как представлена на рисунке 3.

Система GitHub позволяет редактировать репозиторий (редактирование/добавление файлов, выполнение коммитов, создание веток и др.) прямо через Web-интерфейс. Для того чтобы отредактировать файл, необходимо из списка файлов пройти на страницу файла по ссылке с именем файла, нажать иконку «редактировать» (рис. 4), отредактировать файл, написать описание изменений и нажать кнопку “Commit changes” (выполнить коммит, рис. 5).

Для того чтобы каждый раз не вводить входное имя, можно использовать доступ к системе GitHub по ключу.

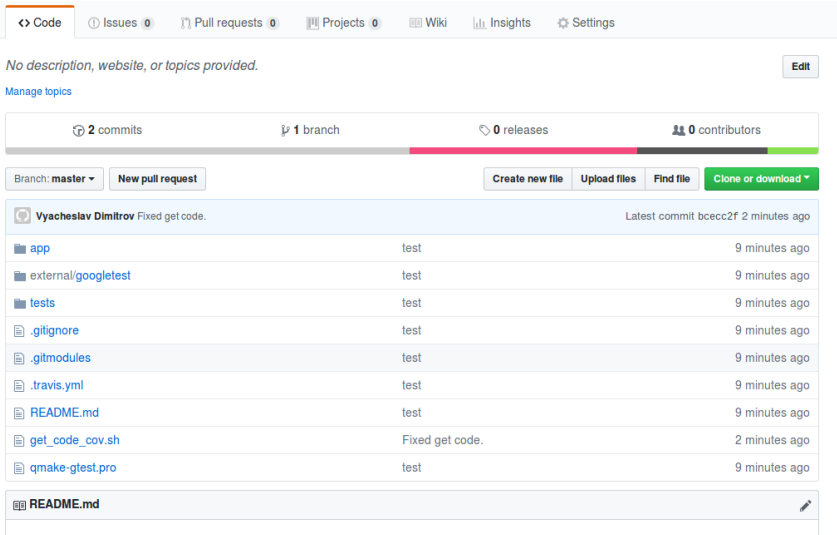


Рис. 3. Страница репозитория с добавленными файлами в системе GitHub

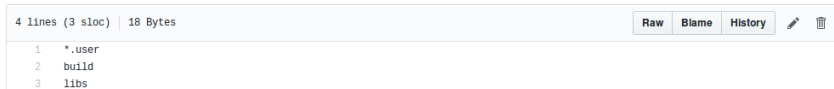


Рис. 4. Страница репозитория с содержимым файла и иконкой редактирования в системе GitHub

§ 4. Система непрерывной сборки Travis

Система непрерывной сборки Travis CI (<https://travis-ci.org/>) предназначена для автоматической компиляции, тестирования и развертывания приложений при изменении исходного кода. В зависимости от настроек, сборка может прерываться при возникновении ошибок компиляции и/или тестирования и уведомлять заинтересованные

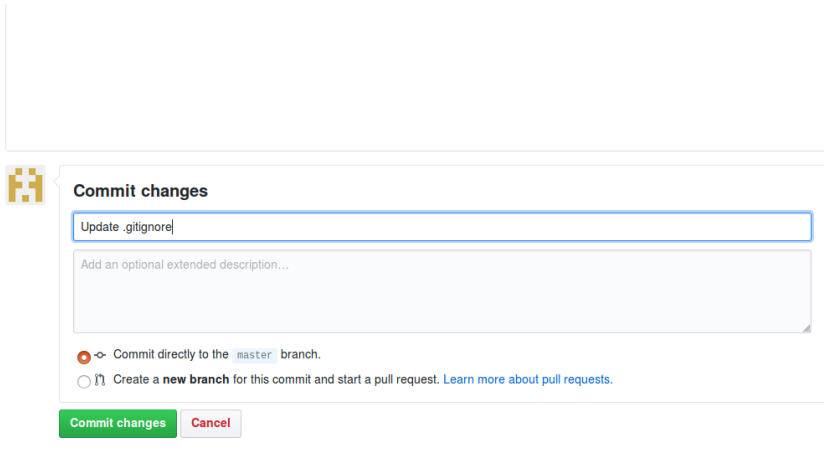


Рис. 5. Страница репозитория выполнения коммита в системе GitHub

лица о статусе выполнения задачи по электронной почте. Таким образом, разработчик освобождается от рутинной операции по развертыванию и проверке проекта в «чистой среде».

Для использования системы непрерывной сборки необходимо зарегистрироваться в системе Travis CI через вход в GitHub и создать скрипт сборки. Скрипт сборки должен иметь predetermined имя “.travis.yml”. Инструкция по созданию скрипта сборки доступна на сайте Travis CI (<https://docs.travis-ci.com/user/customizing-the-build/>). В шаблоне проекта представлен пример скрипта, который можно использовать для организации непрерывной сборки (см. листинг 4.1).

Листинг 4.1: Пример скрипта сборки Travis CI

```
language:
- cpp

before_install:
- pip install --user cpp-coveralls
```

```
addons:
sonarcloud:
organization: "seekerk-github"
token:
secure: $SONAR_TOKEN

script:
- qmake
- make
- ./tests/tests

after_success:
- coveralls --root . -E ".*external.*" \
  -E ".*tests.*"
- build-wrapper-linux-x86-64 --out-dir \
  bw-output make clean all
- sonar-scanner

cache:
directories:
- '$HOME/.sonar/cache'

notifications:
email: false
```

Скрипт состоит из набора секций, каждая из которых описывает часть процесса сборки.

- “language” — используемый в проекте язык программирования. В зависимости от языка выбирается среда сборки с предустановленными инструментами, компиляторами, библиотеками и т. д.
- “before_install” — перечень команд для подготовки среды сборки к работе. В примере выполняется установка инструмента анализа покрытия кода тестами.
- “addons” — настройка дополнительных модулей Travis CI. В примере представлена настройка Sonar Cloud (см. раздел § 6).

- “script” — основной раздел, служит для описания команд сборки. Travis CI умеет выполнять сборку типовых конфигураций, однако в ряде случаев требуется ручная конфигурация.
- “after_success” — перечень команд, выполняемых после успешной сборки. В примере выполняется экспорт результатов анализа покрытия кода тестами (см. раздел § 5) и запуск сканера кода для Sonar Cloud (см. раздел § 6).
- “cache” — настройки кеширования данных между сборками. В связи с тем, что каждый запуск сборки выполняется на отдельной чистой машине, в ряде случаев необходим перенос данных между сборками.
- “notifications” — настройка уведомлений о результатах сборки. В примере уведомления по почте выключены.

При отправке коммитов в репозиторий GitHub выполняется запуск сборки Travis CI. Сборку также можно запустить вручную, нажав кнопку “Restart build”. Пользователю доступен лог сборки, включающий вывод всех запускаемых приложений. Пример лога представлен в соответствии с рисунком 6.

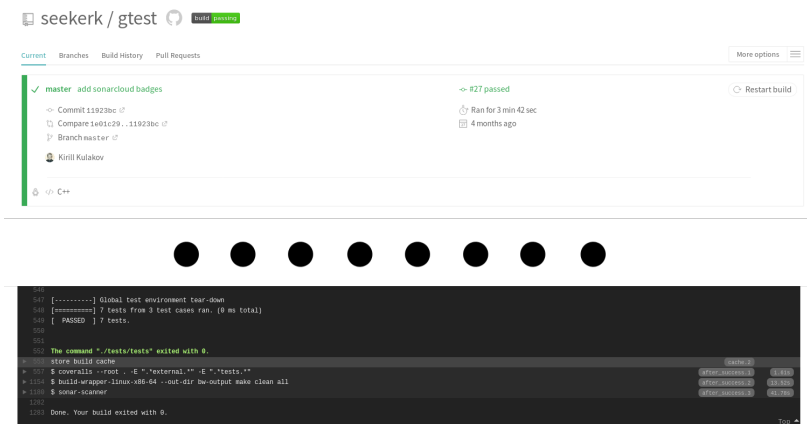


Рис. 6. Фрагменты экрана успешной сборки

В ходе сборки могут возникнуть ошибки двух типов. Самая распространенная — ошибка сборки, когда во время выполнения скрипта программа вернула код, отличный от нуля, или требуемая команда не может быть выполнена. Пример ошибки представлен в соответствии с рисунком 7. Вторая категория ошибок — ошибка разбора скрипта сборки. В этом случае запуск сборки не выполняется, а выводится сообщение об ошибке.

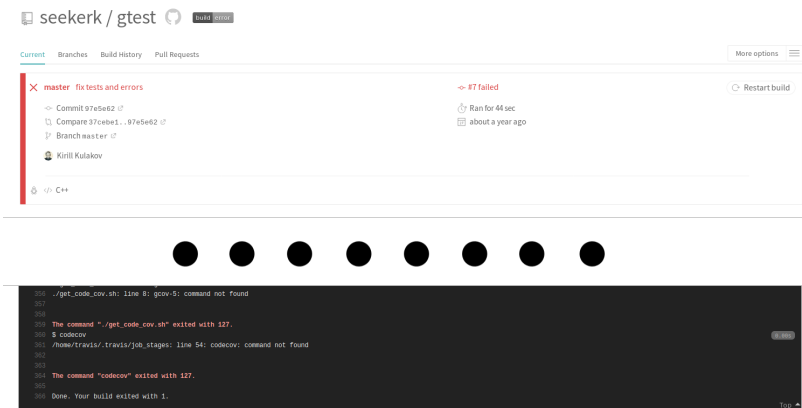


Рис. 7. Фрагменты экрана сборки с ошибкой

Одним из преимуществ использования Travis CI является возможность создания бейджей с результатами сборки. Бейдж отображает последний результат сборки проекта и позволяет пользователям оценить возможность использования кода, опубликованного на GitHub (см. пример ссылки на бейдж проекта в формате Markdown 4.2).

Листинг 4.2: Пример ссылки на бейдж проекта Travis CI

```
[![ Build Status ]( https://travis-ci.org/seekerk/↵
  ↳ gtest.svg?branch=master )]( https://travis-ci.↵
  ↳ org/seekerk/gtest )
```

§ 5. Сервис оценки покрытия кода тестами Coveralls

Одним из традиционных способов оценки качества выполняемого тестирования является оценка покрытия кода тестами. Существуют различные методики оценки [8], однако самой простой и эффективной является оценка числа задействованных строк кода при выполнении теста.

Веб-сервис оценки покрытия кода тестами Coveralls (<https://coveralls.io/>) предназначен для автоматического подсчета числа задействованных строк кода при выполнении тестирования и представления результатов в сети Интернет. Веб-сервис поддерживает работу с различными языками программирования, репозиториями и системами сборки.

Для запуска оценки покрытия необходимо:

- 1) выполнить вход в веб-сервис с помощью GitHub;
- 2) зарегистрировать проект на сервисе Coveralls (кнопка “Add repo” в левой панели);
- 3) добавить вызов анализатора кода и отправку результатов в скрипт Travis CI (см. пример скрипта в § 4.);
- 4) запустить сборку проекта на Travis CI.

Оценка покрытия кода тестами выполняется только после компиляции и запуска тестов в системе сборки Travis CI. Результаты оценки представлены на сайте Coveralls как в графическом формате (см. рис. 8), так и с детализацией по модулям/файлам (см. рис. 9).

Одним из преимуществ использования Coveralls является возможность создания бейджей с результатами сборки. Бейдж отображает уровень покрытия кода тестами и позволяет пользователям оценить возможность использования кода, опубликованного на GitHub (см. пример ссылки на бейдж проекта в формате Markdown 5.1).

Листинг 5.1: Пример ссылки на бейдж проекта Coveralls

```
[![ Coverage Status ]( https://coveralls.io/repos/✓  
  ↪ seekerk/gtest/badge.svg?branch=master )]( https://coveralls.io/github/seekerk/gtest?branch=✓  
  ↪ master )
```

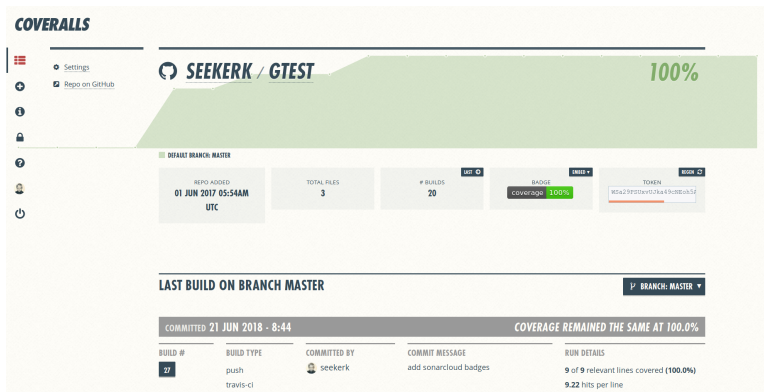


Рис. 8. Отчет о покрытии кода тестами

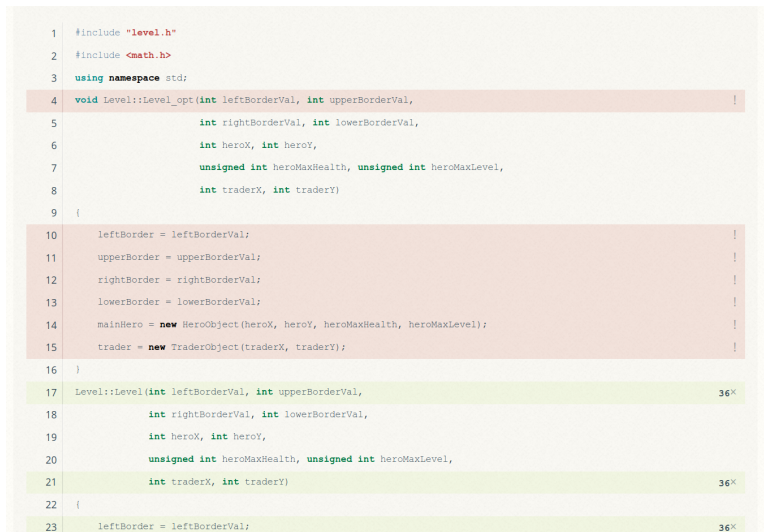


Рис. 9. Детализированный отчет о покрытии для фрагмента кода тестами

§ 6. Сервис непрерывного статического анализа кода SonarCloud

Одним из ключевых аспектов гарантии качества программного обеспечения является статический анализ кода. Современные среды разработки имеют встроенные средства статического анализа, однако такие средства проверяют лишь часть аспектов правильного написания кода.

Сервис непрерывного статического анализа кода SonarCloud (<https://sonarcloud.io>) построен на базе популярного инструмента статического анализа SonarQube и позволяет оценивать качество публикуемого в репозитории GitHub-кода. Сервис поддерживает работу с различными системами непрерывной сборки, выполняет проверку кода на 17 языках программирования.

Для запуска оценки качества необходимо:

- 1) выполнить вход в систему через учетную запись GitHub;
- 2) сгенерировать токен (My Accounts → Security);
- 3) добавить токен в перечень переменных проекта Travis CI (Settings → Environment Variables);
- 4) добавить отправку кода на анализ в скрипте сборки (см. листинг 4.1 в § 4.);
- 5) запустить процедуру сборки.

В результате анализа на сервисе SonarCloud формируется отчет об обнаруженных ошибках (см. пример отчета в соответствии с рисунком 10). Каждая обнаруженная ошибка содержит местоположение в коде, время возникновения, подробное описание и возможные варианты исправления.

Одним из преимуществ использования SonarCloud является возможность создания бейджей с результатами статического анализа кода. Бейджи отображают различную информацию, например, количество ошибок или оценка улучшения качества кода со временем и позволяет пользователям оценить возможность использования кода, опубликованного на GitHub (см. пример ссылки на бейдж проекта в формате Markdown 6.1).

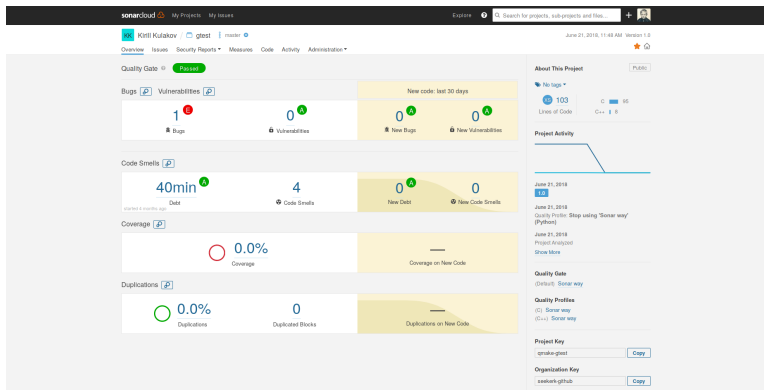


Рис. 10. Отчет о результатах статического анализа кода

Листинг 6.1: Пример ссылок на бейджи проекта SonarCloud

```
[![ Quality Gate ]](https://sonarcloud.io/api/v2
  ↳ project_badges/measure?project=qmake-gtest&
  ↳ metric=alert_status)](https://sonarcloud.io/
  ↳ dashboard?id=qmake-gtest)
[![ Code smells ]](https://sonarcloud.io/api/v2
  ↳ project_badges/measure?project=qmake-gtest&
  ↳ metric=code_smells)](https://sonarcloud.io/
  ↳ dashboard?id=qmake-gtest)
```

§ 7. Лабораторные работы

Лабораторная работа 1. Изучение структуры проекта Qt. Сборка и запуск

- Скачать архив проекта-шаблона (<https://github.com/seekerk/gtest>) с исходным кодом тестового окружения, распаковать в домашнем каталоге ("unzip master.zip").
- Открыть тестовое окружение в Qt creator, запустить тесты (меню "инструменты" → "тесты" → "Запустить все"), проверить результат запуска тестов.
- Создать проект на <https://github.com>.
- Опубликовать проект на <https://github.com>.
- Проверить запуск тестов и оценки покрытия кода тестами (<https://travis-ci.org>, <https://coveralls.io>, <https://sonarcloud.io>).

Лабораторная работа 2. Интеграция модуля в проект Qt

- Взять проект из лабораторной работы № 1.
- Добавить в проект код разработанного модуля.
- Добавить необходимые для компиляции заголовочные файлы и зависимости.
- Добавить вызов функции из модуля в основную программу.
- Выполнить компиляцию и запуск.
- Опубликовать код на <https://github.com>, проверить результат сборки и оценки покрытия кода тестами.

Лабораторная работа 3. Тестирование модуля и оценка качества

- Взять проект из лабораторной работы № 2.

- Составить перечень тестов, проверяющих правильность работы модуля (позитивные тесты) и его реакцию на возможные отклонения от стандартного поведения (негативные тесты).
- Реализовать запланированные тесты. Выполнить тестирование. В случае обнаружения ошибки сформировать протокол ошибки, выполнить отладку и повторное тестирование.
- Опубликовать код тестов, перечень тестов и протоколы ошибок на <https://github.com>, проверить результат сборки и оценки покрытия кода тестами.

Лабораторная работа 4. Реализация многомодульного приложения

- Выбрать тематику проекта.
- Сделать описание проекта и интерфейса пользователя.
- Создать проект на <https://github.com>.
- Реализовать многомодульное приложение.
- Проверить запуск тестов и оценки покрытия кода тестами (<https://travis-ci.org>, <https://coveralls.io>, <https://sonarcloud.io>).

Лабораторная работа 5. Тестирование многомодульного приложения, оценка качества

- Взять проект из лабораторной работы № 4.
- Составить перечень тестов, проверяющих правильность работы приложения (позитивные тесты) и его реакцию на возможные отклонения от стандартного поведения (негативные тесты).
- Реализовать запланированные тесты. Выполнить тестирование. В случае обнаружения ошибки сформировать протокол ошибки, выполнить отладку и повторное тестирование.
- Опубликовать код тестов, перечень тестов и протоколы ошибок на <https://github.com>, проверить результат сборки и оценки покрытия кода тестами.

Список литературы

- [1] IEEE Standard for Software Unit Testing, in ANSI/IEEE Std 1008-1987, 1986.
- [2] IEEE Standard for Software Test Documentation, in IEEE Std 829-1998, 1998. — с. 1-64.
- [3] Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / пер. с англ. Сэм Канер, Джек Фолк, Енг Кек Нгуен. — Киев : ДиаСофт, 2001. — 544 с.
- [4] Верификация программного обеспечения : курс лекций / С.В. Синецын, Н. Ю. Налютин. — Москва : Интуит НОУ, 2016. — 446 с.
- [5] Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл ; пер. с англ. — Москва : Издательско-торговый дом «Русская редакция»; Санкт-Петербург : Питер, 2005. — 896 с.
- [6] McCabe T. “A complexity measure” / T. McCabe // IEEE Transactions on Software Engineering. 1976. — Vol. SE-2, № 4. — Pp. 308—320.
- [7] ISTQB Exam Certification <http://istqbexamcertification.com/>
- [8] Кулаков К. А. Основы тестирования программного обеспечения / К. А. Кулаков, В. М. Димитров. — Петрозаводск : Изд-во ПетрГУ, 2017. — 68 с.

Учебное электронное издание

Кулаков Кирилл Александрович
Димитров Вячеслав Михайлович

Инструменты тестирования программного обеспечения

Методические указания для обучающихся
Института математики и информационных технологий

Редактор *А. Б. Соболева*

Электронная версия и оформление обложки *В. М. Димитрова*

Подписано к изготовлению 25.11.18. Тираж 100 экз.
1 CD-R. 1,5 Мб. Изд. № 201

Федеральное государственное бюджетное образовательное
учреждение высшего образования

**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ**

185910, г. Петрозаводск, пр. Ленина, 33

<https://petsu.ru>

Тел. (8142) 71-10-01

Изготовлено в Издательстве ПетрГУ

185910, г. Петрозаводск, пр. Ленина, 33

URL: press.petsu.ru/UNIPRESS/UNIPRESS.html

Тел./факс (8142) 78-15-40

nvrahomova@yandex.ru