Gem #160: Разработка блочных тестов с помощью GNATtest

Автор: Vasiliy Fofanov - AdaCore

Краткое содержание: Никому не хочется тратить время на тестирование. Это скучно и утомительно: нужно писать средство тестирования, сами тесты, а также подстраивать их под разработку. К счастью, для этого есть `GNATtest` - небольшой инструмент, входящий в состав набора инструментов GNAT, с помощью которого можно автоматически создавать и поддерживать средства тестирования, что значительно упрощает этот процесс.

Давайте начнём…

Когда вы вызываете `GNATtest`, он анализирует вашу программу и идентифицирует все подпрограммы уровня библиотеки (то есть подпрограммы, объявленные в спецификациях пакета); Для каждой такой подпрограммы `GNATtest` создает скелет модульного теста. Он также заботится о создании полностью компилируемой тестовой системы, которая будет инкапсулировать эти скелеты модульного теста, а также ваш собственный код. Когда вы компилируете и запускаете сгенерированную совокупность - жгут, он вызывает каждый из модульных тестов один за другим, затем анализирует результаты и сообщает о них. Все, что вам нужно сделать, это заменить скелеты модульного теста на действительный код модульного теста.

Дистрибутив GNAT предоставляет несколько примеров использования `GNATtest`; их можно найти в **`[install root]/share/examples/gnattest`**. Давайте рассмотрим простой пример "`simple`", чтобы увидеть, каким может быть типичное использование `GNATtest`.

Если вы откроете проект `Simple`, вы увидите, что он содержит один пакет, который в настоящее время объявляет одну подпрограмму. Давайте разработаем набор тестов для него.

Правильная идея – это разобраться с приложением `GNATtest` более подробно, потому что оно интегрировано с GPS. Поэтому вам не нужно вызывать инструмент из командной строки (подробности, как обычно, можно найти в Руководстве пользователя GNAT) – даже если Вы привыкли к этому способу работы. Вместо этого вы можете начать работу с набором тестов буквально одним щелчком мыши.

Для этого просто выберите команду GPS "**`Tools -> GNATtest -> Generate unit test setup`**" («**`Инструменты -> GNATtest -> Создать настройку модульного теста`**»). В проводнике проекта вы увидите, что ваш проект заменен автоматически сгенерированной иерархией проектов, в которой ваш собственный проект стал лишь одной из зависимостей.

Теперь вы можете быстро перейти к коду тестового примера, щелкнув правой кнопкой мыши на подпрограмме `Inc` и выбрав «**`GNATtest -> Go to test case`**». Вы увидите, что в настоящее время тест содержит только заглушку:

```
AUnit.Assertions.Assert
  (Gnattest_Generated.Default_Assert_Value, "Test not implemented.");
```

Вы уже можете собрать и запустить свой жгут – совокупность сгенерированных проектов, но все, что сделает приведенный выше код, - это провал теста:

```
$ test_runner

simple.ads:7:4: corresponding test FAILED: Test not implemented.
(simple-test_data-tests.adb:25)
```

```
1 tests run: 0 passed; 1 failed; 0 crashed.
```

Давайте реализуем тест, заменив `Gnattest_Generated.Default_Assert_Value` логическим выражением, которое фактически вызывает нашу подпрограмму, проверяет результат и предоставляет информативное сообщение:

```
AUnit.Assertions.Assert (Inc (1) = 2, "Incrementation failed.");
```

После перекомпиляции тестового драйвера вы можете увидеть, что тест теперь проходит.

Поддержание набора тестов в актуальном состоянии

Вы можете вызывать `GNATtest` любое количество раз на уже сгенерированном жгуте - он никогда не будет перезаписывать тела тестовых программ ранее сгенерированных, поэтому вы не потеряете свою работу. В то же время, это может добавить больше скелетов модульного тестирования, если вы добавите больше подпрограмм в свой код.

Посмотрим, как это работает. Раскомментируйте вторую подпрограмму и ее тело в исходных файлах проекта `Simple`, затем снова запустите `GNATtest`, скомпилируйте и запустите тестовый драйвер.

Вы увидите, что старый тест все еще проходит, но теперь есть новый нереализованный тест:

```
$ test_runner

simple.ads:7:4: corresponding test PASSED

simple.ads:9:4: corresponding test FAILED: Test not implemented.
(simple-test_data-tests.adb:46)

2 tests run: 1 passed; 1 failed; 0 crashed.
```

`GNATtest` также предупредит вас о том, что некоторые тесты стали зависать, если вы измените профиль параметра или удалите подпрограммы, для которых уже написаны модульные тесты.

Заключение

Как видите, `GNATtest` предоставляет легкое, и доступное решение для начала разработки модульных тестов. Если у вас ещё нет инфраструктуры для юнит-тестирования, которая работает на вас - мы призываем вас попробуйте `GNATtest`! (и, кстати, знаете ли вы, что вы также можете объединить существующие модульные тесты в жгут, сгенерированный `GNATtest`?).

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Vasiliy Fofanov – AdaCore

Василий Фофанов

Ведущий программист

Местоположение

Проживание: Франция, Париж

Возраст и стаж

Стаж: 18 лет и 9 месяцев

Контактная информация

Мой круг: https://moikrug.ru/vfofanov

Опыт работы

Август 2000 - По наст. время(18 лет и 9 месяцев)

AdaCoreведущий программист

Франция, Париж

Высшее образование

Январь 1991 - Январь 2000

Московский государственный университет имени М.В.Ломоносова

Вычислительной математики и кибернетики, ВМиК

Россия, Москва

О себе

Специализация и профессиональные навыки: Язык программирования Ада, ОС OpenVMS

«Мой Круг» - вакансии для IT-специалистов

Last Updated: 10/30/2017
Posted on: 6/23/2014

Обсуждение…