

Практика 1: Вступление, задача про CLI

15.01.2020г

1. Формальности

Занятия у нас раз в неделю, курс длится два модуля. В конце надо будет получить оценку за этот курс (которая станет частью итоговой оценки вместе с оценкой по соответствующему теоретическому курсу, который читает Тимофей Брыксин). На практике мы будем пытаться применять знания, полученные на теоретическом курсе, поэтому будет некоторое (довольно большое) количество домашних, которые обсуждать и начинать делать мы будем в аудитории, а доделывать по всей видимости, придётся дома. Домашние задачи будут оцениваться по десятибалльной шкале, с текущим и максимальным возможным баллом, как это было в курсе по Java. Будет некоторое количество дедлайнов, за нарушение которых будут сниматься баллы. Задачи сразу предупреждаю, что довольно объёмные (по крайней мере, некоторые из них, будут и простые), большинство будет требовать программной реализации, на любом языке программирования. Итоговый балл будет вычисляться по смещённой линейной шкале с разными весами для разных задач, система будет устроена так, чтобы промотивировать сделать вообще все домашки, да ещё и работать на парах.

Сдавать решения надо будет, выкладывая их на GitHub и делая пуллреквест в свой же собственный репозиторий, я буду стараться этот пуллреквест комментировать и выставять текущий балл в отдельную таблицу на Google Docs, ссылка на которую будет в HwProj. Когда пуллреквест принят (accepted, мерджить я сам не буду), задача считается зачтённой и дальше её править нет смысла.

Задачи надо будет сдавать, как обычно, через HwProj. Надо записаться на курс <http://hwproj.me/courses/51> и выкладывать ссылки на пуллреквесты туда. Если что, пишите на почту. Если потребуется более оперативная связь, есть телеграм-чат курса по SE, мы его будем переиспользовать, можно писать и в личку. В качестве языка и среды программирования можно использовать то, что вам больше нравится, курс всё-таки больше про архитектуру, а не про реализацию, только уж совсем эзотерических языков не надо, я должен иметь возможность хотя бы собрать и запустить то, что у вас получилось. Если процесс сборки нетривиален или используются какие-нибудь редкие технологии (включая Haskell), следует задокументировать (а ещё лучше — полностью автоматизировать) процесс сборки. Лучше писать на Java/C#/F#/C++, больше шансов получить содержательные комментарии, относящиеся к технике кодирования.

Ещё обратите внимание, что просто сделать домашку может быть недостаточно. У нас тут курс по проектированию, поэтому решение, удовлетворяющее всем функциональным

требованиям, вообще, строго говоря, никому не интересно. Важна архитектурная красота, обоснованность принятых решений, чистая и аккуратная реализация, наличие необходимых сопровождающих артефактов. Поскольку архитектура — это в каком-то смысле больше искусство, чем наука, чётких критериев оценки не будет, и придирайтесь я буду в основном к форме — качеству кода, наличию документации, соответствию формальным правилам.

Студентами от практики по архитектуре обычно ожидается много критики архитектуры, но тут проблема в том, что если вы достаточно владеете программированием и имеете хоть какой-то опыт, вы окажетесь вполне способны породить адекватное решение к учебным задачкам, так что, скорее всего, критика архитектуры решения будет в духе “ок, мне нравится, но можно было ещё вот так” или её не будет вообще. Это плохо, но делать нечего, реальные проблемы с архитектурой могут возникнуть только на реальных проектах. Кстати, если интересно и у кого-нибудь будут интересные НИР, можно устроить их архитектурный разбор. В целом же, тренировать мы будем прежде всего владение инструментами, которые позволяют создать и описать архитектуру.

2. Что будет в курсе

Собственно, практические занятия по проектированию ПО — затея, изначально обречённая на провал, невозможно научиться проектированию ПО, посещая практические занятия в университете и даже делая все домашки. Лучший способ научиться проектировать ПО — проектировать реальные проекты, причём достаточно большие. С этим может не сложиться, по крайней мере, поначалу, и уж тем более это нельзя впихнуть в формат практических занятий. Поэтому мы будем практиковаться на игрушечных примерах, но требования к ним будут предъявляться такие, будто это огромные промышленные проекты в сотни человеколет трудозатрат. Так что аргументы в духе “да я это на питоне в двести строчек нафигачу без всякой архитектуры-шмархитектуры” заранее признаются невалидными.

На занятиях примерно в половине случаев я буду что-то рассказывать (больше про практическую сторону дела, какие-то примеры из реальной практики, про инструменты и методы, а иногда и просто продолжать лекции), в половине случаев прямо на паре надо будет сделать что-нибудь небольшое и, если не успели, доделать дома. Будет несколько задач на кодирование (всего штук три крупные задачи) и много мелких задач в духе “нарисовать диаграмму”, “написать документ”, “проанализировать”. Рассказывать я буду про архитектурную документацию, будет довольно много и теории и практики про UML и другие визуальные языки, которые используются при разработке архитектуры, про реализационные аспекты паттернов проектирования (что такое, как устроено и зачем надо, расскажут на лекциях, а как это писать и где можно прострелить себе ногу — на практике, такие дела), про антипаттерны (на лекциях будут рассказывать, как писать надо, на практиках — как не надо), примеры различных архитектур, разные подходы к созданию архитектуры и вещи, которые не так важны, чтобы включать их в курс лекций и программу экзамена, но тем не менее достойные упоминания.

Однако даже к задачам на кодирование надо относиться как к архитектурным, у нас тут всё-таки не практика по Java — надо сначала продумать и описать архитектуру разрабатываемой системы, уже затем кидаться её кодить, и главное, при кодировании не забывать

про архитектуру и её связь с кодом. Овердизайн и массивованное применение знаний, полученных на лекциях, приветствуется (правда, до разумных пределов). При проверке от решений задач будет ожидаться:

1. работоспособность и соответствие требованиям (явным или неявным);
 - (а) кстати, negotiation является важной частью работы архитектора, так что требования можно (и иногда нужно) уточнять, договариваться, возможно, убеждать препода, что тот или иной пункт условия ему на самом деле не нужен и т.д., но, опять-таки, до разумных пределов;
2. наличие архитектурной документации;
 - (а) её форма и количество будут меняться по ходу курса, от README и комментариев к коду в начале до UML-диаграмм и формального design document-а в конце;
 - (б) комментарии к каждому классу, интерфейсу и public-методу, тем не менее, всегда будут обязательны;
 - (с) краткое описание деталей реализации в README — тоже;
3. следование стайлгайдам и общепринятым правилам здравого смысла в программировании;
4. наличие юнит-тестов, покрывающих все требования условия (там, где это возможно);
5. максимально возможной кроссплатформенности и переносимости кода (писать ради этого на ANSI C не надо, но писать на Java программы, на ровном месте не работающие под виндой просто потому, что вы забыли об её существовании — тоже);
 - (а) проверяться задача может как под Linux, так и под Windows, и если у неё нет веских причин не работать под одной из этих операционных систем, а она не работает, могут попросить исправить;
6. применение промышленных практик, общепринятых при разработке production-кода: логирование, Continuous Integration, разумные стратегии обработки исключений.

Обратите внимание, что многое из того, что я буду рассказывать, и главное, требовать от решений, может показаться оверкиллом — например, комментарии к **каждому** public-методу, о боже, что? Или рисование пачки диаграмм для программы в 500 строк на Питоне. Это как раз связано с тем, про что я уже говорил — мы считаем, что речь идёт о больших реальных проектах, где это всё не выпендрёж, а единственный способ не развалиться от собственной сложности. Часто возражают, что в реальных проектах так тоже не делают — я слышал от кого-то из магистров, что в Яндексе сколько-то гигабайт кода и ни одной UML-диаграммы — охотно верю, поэтому у нас всё так плохо но на этих парах мы учимся техникам, принципам и инструментам, так что некоторые требования действительно искусственны. Исключительно с учебными целями. Можно этого всего не делать и просто писать код, но тогда всю жизнь будете junior-ами, которым говорят, какой код писать.

3. Задача про CLI

Начнём мы с задачи на проектирование и покодить. Задача такая: реализовать простой интерпретатор командной строки, поддерживающий следующие команды:

- `cat [FILE]` — вывести на экран содержимое файла;
- `echo` — вывести на экран свой аргумент (или аргументы);
- `wc [FILE]` — вывести количество строк, слов и байт в файле;
- `pwd` — распечатать текущую директорию;
- `exit` — выйти из интерпретатора.

Кроме того, должны поддерживаться одинарные и двойные кавычки (full and weak quoting, то есть одинарные кавычки передают текст как есть, двойные выполняют подстановки переменных окружения с оператором `$`), собственно окружение (команды вида “имя=значение”), оператор `$`, вызов внешней программы через `Process` (или его аналоги) для любой команды, которую интерпретатор не знает. Должны ещё поддерживаться пайплайны (оператор `|`), то есть перенаправление ввода и вывода. Примеры:

```
> echo "Hello, world!"
Hello, world!
> FILE=example.txt
> cat $FILE
Some example text
> cat example.txt | wc
1 3 18
> echo 123 | wc
1 1 3
> x=exit
> $x
```

Тимофей ещё не рассказывал, наверное, про Architectural drivers (честно как-то не встречал адекватного русского перевода, так что будем обходиться английским термином), но ещё расскажет. Пока что — это основные факторы, определяющие архитектуру системы. Поскольку мы пишем объектно-ориентированный код (ну, потому что), мы будем исходить из обычных для объектно-ориентированных программ качеств — сопровождаемость, расширяемость, переиспользуемость и т.д. В нашем случае это значит, в частности, что проектировать систему надо так, чтобы новые команды было добавлять легко (логично, что шелл будет постепенно расширяться новыми встроенными командами), но желательно поддерживать архитектуру достаточно простой и слабо связанной, чтобы можно было реализовать и другие требования, которые могут возникать по ходу. Может потребоваться внезапно реализовать ещё что-нибудь из того, что умеют обычные шеллы, как и в реальной жизни, желания заказчика непредсказуемы (поэтому, кстати, не надо пытаться их предугадать и заложить в архитектуру — то, о чём вы подумали, никогда не случится, случится то, о чём вы не подумали). В итоге от задачи хочется видеть код, который делает то, что написано в условии (то есть удовлетворяет функциональным требованиям),

адекватно покрыт юнит-тестами (то есть удовлетворяет требованиям качества и сопровождаемости), имеет комментарии к каждому классу и хотя бы к каждому публичному методу (ещё один необходимый компонент сопровождаемости), имеет небольшой текстовый документ (можно Javadoc-комментарий в `package-info.java`), описывающий кратко архитектуру (пока design document-ы не проходили, не надо писать подробно, полстраницы ок), имеет единый стиль кодирования и написан без явных косяков кодирования на языке, который вам больше нравится. Результаты надо закоммитить к себе в репозиторий на гитхаб (можно создать один репозиторий для всех задач этого курса и раскладывать задачи по отдельным папкам) в отдельную ветку и сделать пуллреквест в мастер, про что через HwProj сообщить мне. В идеале в конце курса хочется получить репозиторий со всеми решениями в master, так что ветки с зачтёнными задачами не стесняйтесь мерджить.

Прямо сейчас задача — выполнить анализ и определить подходы к решению задачи, выявить как можно больше потенциальных подводных камней и способов их преодоления. Дается некоторое время подумать, после чего ожидается живое обсуждение, в ходе которого мы обсуждаем детали того, как бы вы стали писать такую задачу. И можно прямо на паре начинать писать.

Цель сейчас двоякая — во-первых, вы, пока вам почти ничего про архитектуру не рассказали, должны продемонстрировать те навыки, что у вас уже есть (умеете UML — рисуйте, шарите в паттернах — используйте, чем больше, тем лучше). Во-вторых, у всех должно к концу пары сложиться понимание, как сделать домашку, потому как даже среди магистров шестого курса были люди, которые не знали даже, как подступиться к решению. Это вполне ок, чем больше окажется людей, которые сейчас без идей, как эту задачу делать, тем лучше — в этом основной смысл архитектуры, превращать непонятные задачи в набор простых задач на реализацию методов, с которыми справится даже новичок.