

# Практика 7: Примеры архитектур

Юрий Литвинов  
[yurii.litvinov@gmail.com](mailto:yurii.litvinov@gmail.com)

28.03.2022

# Enterprise Fizz-Buzz

Задача:

Для чисел от 1 до 100:

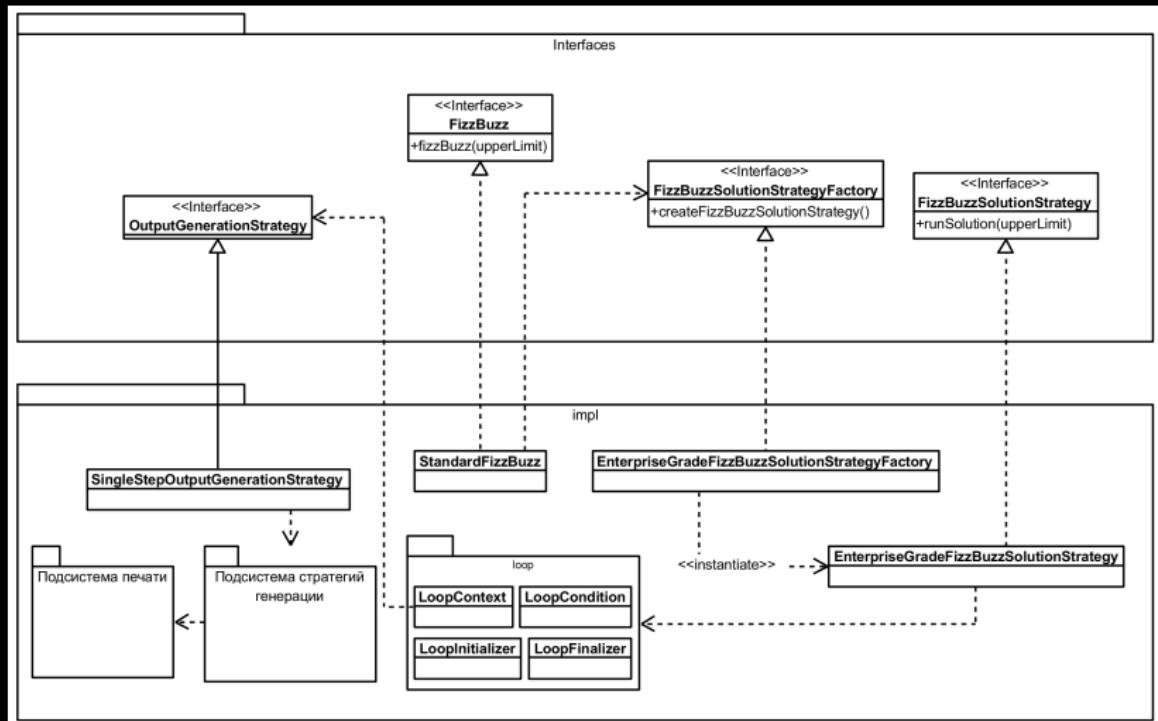
- ▶ если число делится на 3, вывести “Fizz”
- ▶ если число делится на 5, вывести “Buzz”
- ▶ если число делится и на 3, и на 5, вывести “FizzBuzz”
- ▶ во всех остальных случаях вывести само число

Решение:

[https:](https://github.com/EnterpriseQualityCoding/FizzBuzzEnterpriseEdition)

//github.com/EnterpriseQualityCoding/FizzBuzzEnterpriseEdition

# Структура системы



# Хорошие идеи

- ▶ Separation of Concerns
- ▶ Dependency Inversion
- ▶ Dependency Injection
  - ▶ Spring Framework
- ▶ Паттерны “Фабрика”, “Стратегия”, “Посетитель”,  
“Адаптер”, что-то вроде паттернов “Спецификация” и  
“Цепочка ответственности”

# Плохие идеи

- ▶ Не выполняется принцип Keep It Simple Stupid
  - ▶ Неправильно говорить “строк кода написано”, правильно — “строк кода израсходовано”
- ▶ “Синтаксическое” разделение на пакеты, а не “семантическое”
  - ▶ Отсутствие модульности, антипаттерн “Big Ball of Mud”
- ▶ Хардкод основных параметров вычисления
- ▶ Нет юнит-тестов, только интеграционные; нет логирования
- ▶ 1662 строки кода, очень мало комментариев (несмотря на принятый пуллреквест “Serious Documentation”)
  - ▶ Отсутствие архитектурного описания

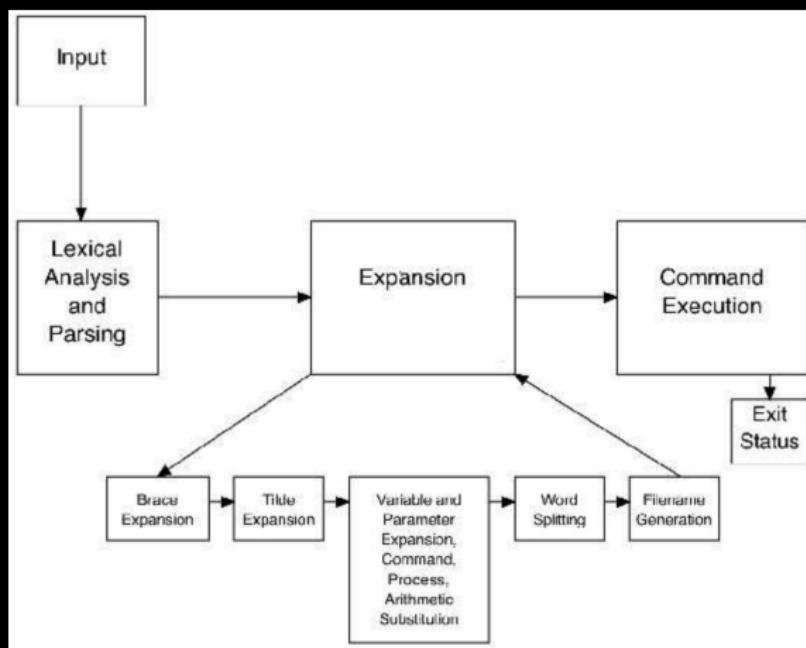
# Bash<sup>1</sup>

- ▶ Примерно 70К строк кода
- ▶ Исходный автор — Brian Fox, maintainer — Chet Ramey
- ▶ Первый релиз — 1989
- ▶ Написан на С
- ▶ Архитектурное описание — глава в *The Architecture of Open Source Applications*, написанная Chet Ramey

---

<sup>1</sup>По <http://aosabook.org>

# Архитектура Bash



# Основные структуры данных

```
typedef struct word_desc {  
    char *word; /* Zero terminated string. */  
    int flags; /* Flags associated with this word. */  
} WORD_DESC;  
  
typedef struct word_list {  
    struct word_list *next;  
    WORD_DESC *word;  
} WORD_LIST;
```

## Ввод с консоли

- ▶ Библиотека Readline
  - ▶ независимая библиотека, но пишется в основном для Bash
- ▶ Цикл read/dispatch/execute/redisplay
- ▶ Dispatch table (или Keymap)
- ▶ Буфер редактирования, хитрый механизм расчёта действий для отображения
- ▶ Хранит все данные как 8-битные символы, но знает про Unicode

# Синтаксический разбор

- ▶ Зависимый от контекста лексический анализ  
`for for in for; do for=for; done; echo $for`
- ▶ Использует lex + bison
- ▶ Подстановка alias-ов выполняется лексером
- ▶ Сохранение и восстановление состояния парсера

## Подстановки

`${parameter:-word}`

раскрывается в *parameter*, если он установлен, и в *word*, если нет

`pre{one,two,three}post`

раскрывается в

`preonepost pretwopost prethreepost`

Ещё бывает подстановка тильды и арифметическая подстановка, сопоставление шаблона

## Исполнение команд

- ▶ Встроенные и внешние команды, обрабатываются единообразно
- ▶ Перенаправление ввода-вывода, отмена перенаправления
- ▶ Принимают набор слов
  - ▶ Иногда обрабатывают по-особому, например, присваивание в *export*
- ▶ Присваивание — тоже команда, но особая
- ▶ Перед запуском внешней команды — поиск в PATH, кеширование результатов
- ▶ Job control, foreground и background

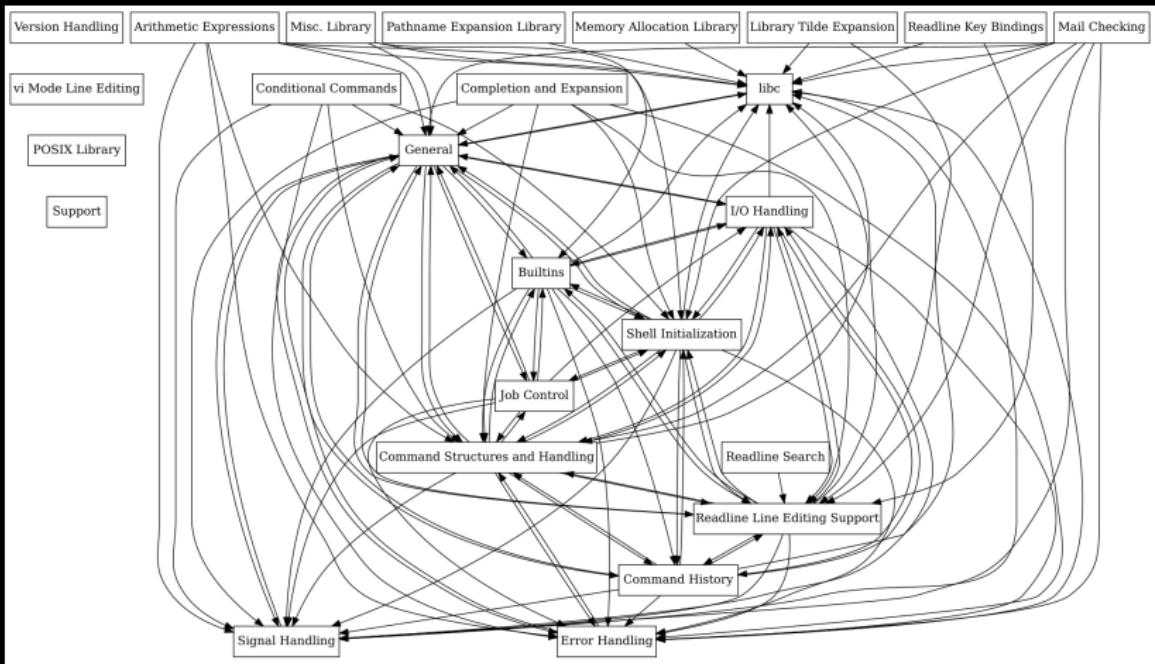
## Lessons Learned

- ▶ Комментарии к коммитам со ссылками на багрепорты с шагами воспроизведения
- ▶ Хороший набор тестов, в Bash их тысячи
- ▶ Стандарты, как внешние на функциональность шелла, так и на код
- ▶ Пользовательская документация
- ▶ Переиспользование

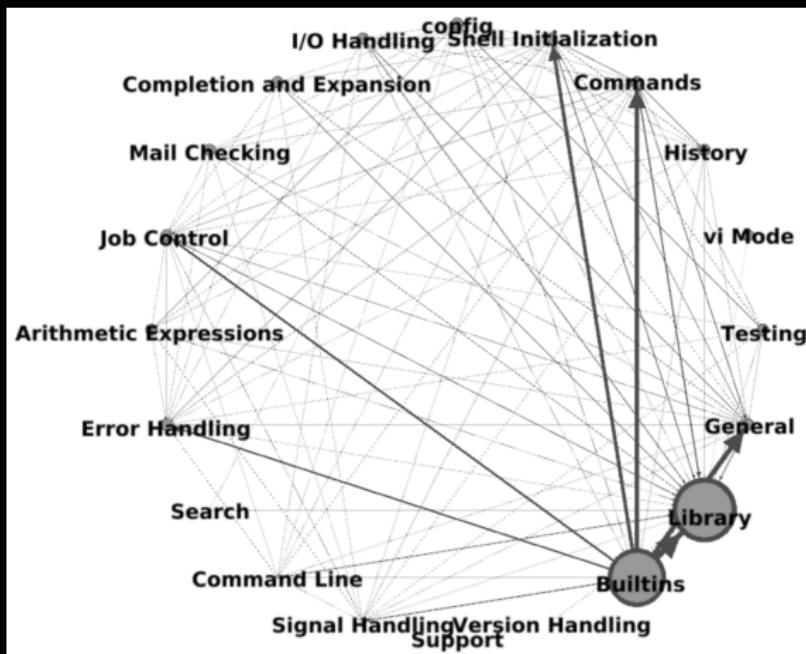
# Архитектура Bash, на самом деле

- ▶ J. Garcia et al., *Obtaining Ground-Truth Software Architectures*
- ▶ 1 аспирант, 80 часов работы
- ▶ Верификация от Chet Ramey
- ▶ 70K строк кода, 200 файлов, 25 компонент
  - ▶ 16 — ядро, 9 — утилиты
- ▶ Структура папок почти не соответствует выделенным компонентам

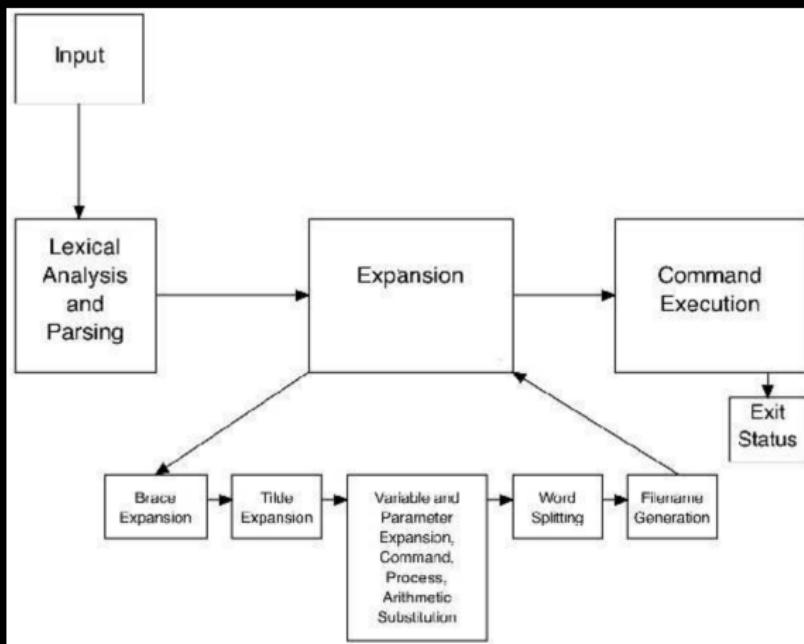
## Архитектура Bash, на самом деле



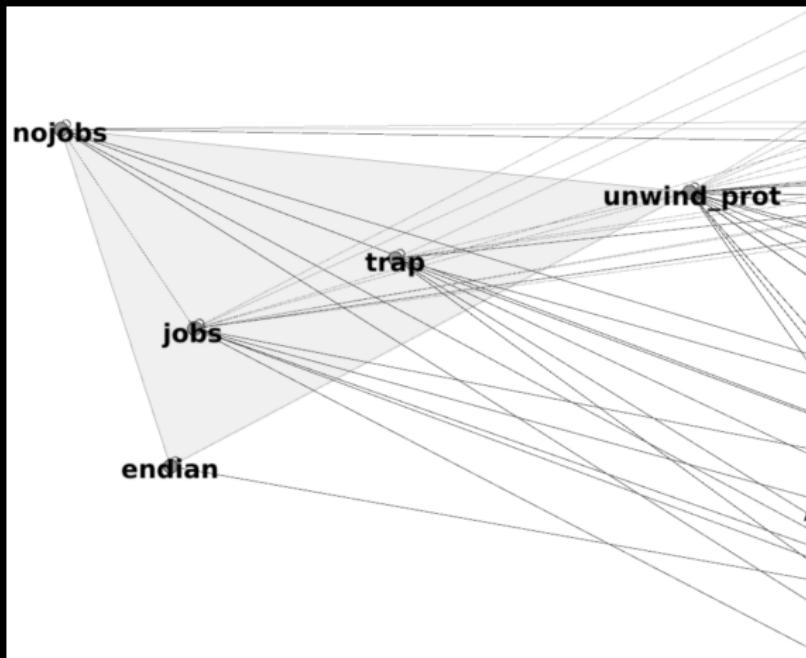
# Результаты анализа кода



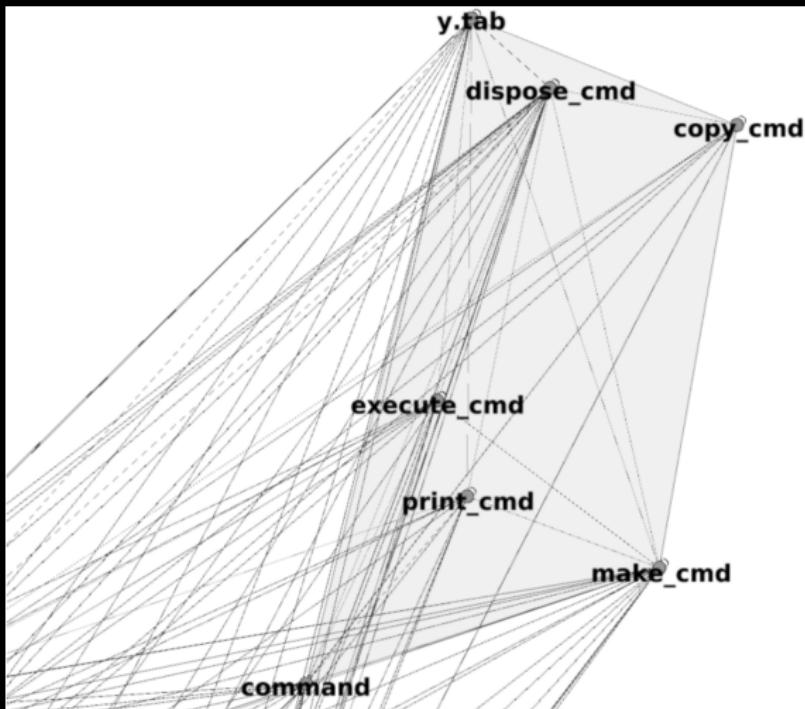
# Сравним с исходной



# Job Control



# Commands



# Battle for Wesnoth<sup>2</sup>

- ▶ Пошаговая стратегия
- ▶ Порядка 200000 строк кода на C++
- ▶ 4 миллиона скачиваний
- ▶ 9/10 на Steam
- ▶ 2003 год



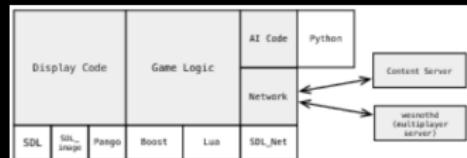
© <https://www.wesnoth.org/>

## Architectural Drivers

- ▶ Доступность для новых разработчиков и авторов контента
- ▶ В ущерб технической красоте
- ▶ Не nice to have, а условие выживания проекта в контексте широкого open-source сообщества из людей без каких-либо обязательств и разного технического уровня

# Высокоуровневая архитектура

- ▶ Wesnoth Markup Language (WML)
- ▶ Минимизация зависимостей от сторонних библиотек
  - ▶ SDL (Simple Directmedia Layer) для видео и ввода/вывода
    - ▶ Простота использования и кроссплатформенность
  - ▶ Boost, Pango, zlib, Python, Lua, GNU gettext



# Основные компоненты

- ▶ Парсер и препроцессор WML
- ▶ Базовый ввод-вывод — видео, звук, сеть
- ▶ GUI — виджеты
- ▶ Display module — игровая доска, юниты, анимация и т.д.
- ▶ ИИ
- ▶ Поиск пути (плюс утилиты для работы с гексагональной доской)
- ▶ Генератор карт
- ▶ Специализированные модули
  - ▶ Титульный экран
  - ▶ Storyline module — для проигрывания катсцен
  - ▶ Лобби — для мультиплеера
  - ▶ “Play game” module — управление основным игровым процессом
- ▶ Отдельно — wesnothd и content server

# Wesnoth Markup Language

```
[unit_type]
id=Elvish Fighter
name=_ "Elvish Fighter"
image="units/elves-wood/fighter.png"
hitpoints=33
advances_to=Elvish Captain,Elvish Hero
{LESS_NIMBLE_ELF}
[attack]
name=sword
icon=attacks/sword-elven.png
range=melee
damage=5
[/attack]
[/unit_type]
```

# Макросы

```
#define GOLD EASY_AMOUNT NORMAL_AMOUNT HARD_AMOUNT  
#ifdef EASY  
    gold={EASY_AMOUNT}  
#endif  
#ifdef NORMAL  
    gold={NORMAL_AMOUNT}  
#endif  
#ifdef HARD  
    gold={HARD_AMOUNT}  
#endif  
#enddef  
...  
{GOLD 50 100 200}
```

## Модель данных

- ▶ Всё сливается в один гигантский WML-документ
- ▶ Перезагружается при смене опций
- ▶ Всякие хаки на уровне препроцессора, чтобы не грузить вообще всё
- ▶ Классы unit и unit\_type (архитектурный стиль Knowledge Layer)
- ▶ Фиксированный набор поддерживаемых движком атрибутов, задаваемых для каждого типа через WML
  - ▶ Нельзя описывать произвольное поведение через WML, хотели сохранить декларативность
- ▶ Класс attack\_type
- ▶ Трейты, инвентарь

## Мультиплеер

- ▶ Начальное состояние и команды
- ▶ Сервер просто пересыпает команды между клиентами
  - ▶ TCP/IP
- ▶ Replay
- ▶ Никакой защиты от читов
- ▶ Версии клиентов

## Lessons Learned

- ▶ 250 тысяч строк на WML
- ▶ Сотни созданных пользователями кампаний
- ▶ 74 тысячи коммитов, 196 контрибуторов
- ▶ Сами разработчики смеются над WML
- ▶ В целом задача обеспечить доступность для модификации очень сложна