Отчет за период 06.04-17.04
Запланировано задач к текущему дню: 10
Выполнено: 12
Всего выполнено: 31/36.
Общий процент выполнения плана: 86%
Остается недель до окончания: 1,2

В RxJava стало легче и проще понимать операторы, документацию и исходный код фреймворка. Нашел варианты для улучшений в рабочем проекте, в особенности что касается используемых потоков. Приступил к чтению книги. Прочитал к текущему моменту большую часть. Как ранее отмечал, книга про построение реактивных систем и не связана именно с RxJava.

День 08.04
Приступил к чтению Reactive Design Patterns. Прочел Part 1.
Затронули концепты и принципы, необходимые для построения реактивных приложений. Цену построения традиционных, не реактивных, приложений. Затронули функциональное программирование, как оно соотносится с реактивным.

Вводятся принцип Reactive Manifesto:
- Responsive
- Resilient
- Elastic
- Message-driven.

Почитал о подходах к асинхронному вычислению в системах:
- модель Green threads
- Future/Promise (Java 8, Scala)
- Reactive extensions
- Actors

Описывается доступность систем. ACID (Associative, Commutative, Idempotent, Distributed).

День 09.04
Приступил к Part 2.
В главе будут рассмотрены следующие положения:
- Включение инкапсуляции и изоляции с асинхронной передачей сообщений
- Улучшение композиции и добавление горизонтального масштабирования с Location transparency
- Структурирование системы в иерархию модулей по принципу “Разделяй и властвуй”
- Как такая структура позволяет управлять ошибками
- Достижение согласованной программной семантики в распределенных системах

- Избежание непредсказуемости там, где возможно. И добавление ее там, где необходимо
- Руководство разработкой приложения на основе топологии его потоков сообщений

Event-based vs. message-based

В общем существуют 2 модели подключения потребителей данных к производителям данных: основанная на событиях и основанная на сообщениях.

Системы, основанные на событиях, обычно организованы вокруг цикла событий. Каждый раз, когда что-то происходит, соответствующее событие добавляется в очередь. Цикл событий постоянно извлекает события из очереди и выполняет их callback, который были к ним присоединены. Почти все GUI основаны на такой модели.

Напротив, система, основаная на сообщениях, предоставляют способ отправки сообщений конкретным получателям. Callback заменяется активным получателем, который ответственен за их обработку.

Backpressure как ситуация, когда обработка сообщений превышает по времени доставку сообщений.

Принципы гарантии доставки сообщений:
- At-most once delivery. Каждый запрос отправляется однажды и не повторяется
- At-least once delivery. Получателем должно отправить ответное сообщение, что первое обработано. Отправителем удерживается запрос, чтобы в случае не получения информации о доставки отправить запрос повторно
- Exactly once delivery. Самый дорогой способ, и источник, и получатель должны удерживать соединение

Реализация на Actors обычно предоставляет at-most once delivery из коробки, но и поддерживает остальные.

Рассмотрели явную передачу сообщений между распределенными сервисами

День 10.04. Principled failure handling

Про показ Внутренней ошибки сервиса пользователю приведена аналогия с сотрудником магазина и покупателем. Если сотрудник начинает покупателю говорить о своих внутренних проблемах и как ему не нравится работа, то это не ответственность покупателя. Юморной пример, однако некоторые сервисы говорят пользователю, что что-то внутри сервиса сломалось.

День 11.04

Прочитал об идее создания более предсказуемых (determenistic) программ с pure functional programming. Также узнал о разумном компромиссе между предсказуемыми и непредсказуемыми программами в угоду производительности.

Завершил чтение логической части о принципах, лежащих в проектировании реактивных приложений. Ключевая вещь, - декомпозиция всей бизнес-логики по принципу “разделяй и властвуй”, - полная изолированность модулей, взаимодействие между которыми в соответствии посредством message flow, которые:

- Асинхронны
- Неблокируемы
- Location-transparent

Процесс модуляризации управляется следующими правилами:

- Один модуль делает ровно одну работу
- Ответственность модуля ограничена ответственностью его родителя
- Модули инкапсулируют в себе сбои, а их иерархия определяется паттерном supervisor
- Жизненный цикл модуля лежит в границах ЖЦ родителя

День 14-15.05

Приступил к чтению третьей части, описывающей паттерны. Первый паттерн был о тестировании приложений. В целом, пока не особо полезна тема, но некоторые мысли были интересны.

День 16.05

Рассмотрели проектирование и реализацию надежных систем:

- Описали простые компоненты, которые следуют принципу single responsibility
- Рассмотрели реализацию паттерна Error Kernel
- Также рассмотрели последствия перезапуска компонентов в шаблоне Let-it-crash
- Также изучили разделение компонентов между собой в шаблоне circuit breaker

Также рассмотрел паттерну репликации:

Active-Passive replication pattern.

Храните множество копий работающего сервиса в разных местах, но применяйте изменения только в одном. Паттерн также известен как Master-Slave replication.

Multiple-Master replication pattern.

Храните множество копий работающего сервиса в разных местах, применяйте изменения везде и распространяйте все изменения