Структурный дизайн

Древний секрет простого и быстрого кода

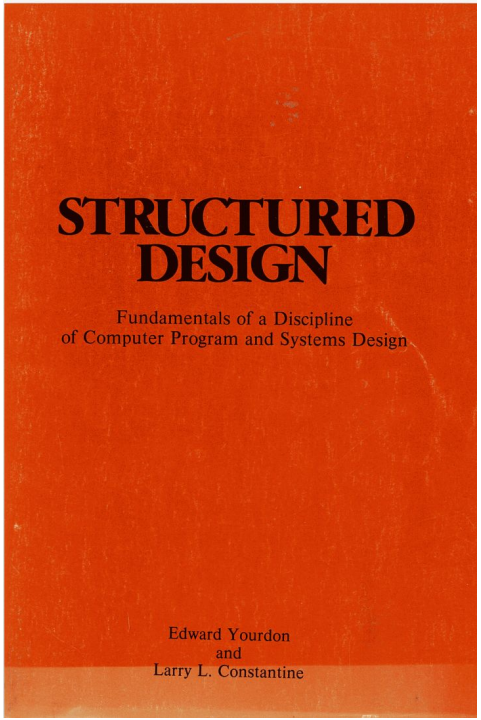
БИО

- Пишу код в две смены с 2003 года
- Ищу способ систематически писать хороший код с 2014 года

О чём поговорим

1. Экспресс-курс структурного дизайна
2. Кейс рефакторинга 50 строк кода
    a. Kotlin и диаграммы
3. Кейс реинжиниринга 2К строк кода
    a. Диаграммы и Java 8
4. Кейс реинжиниринга 40К строк кода
    a. Сводная таблица
5. Что почитать

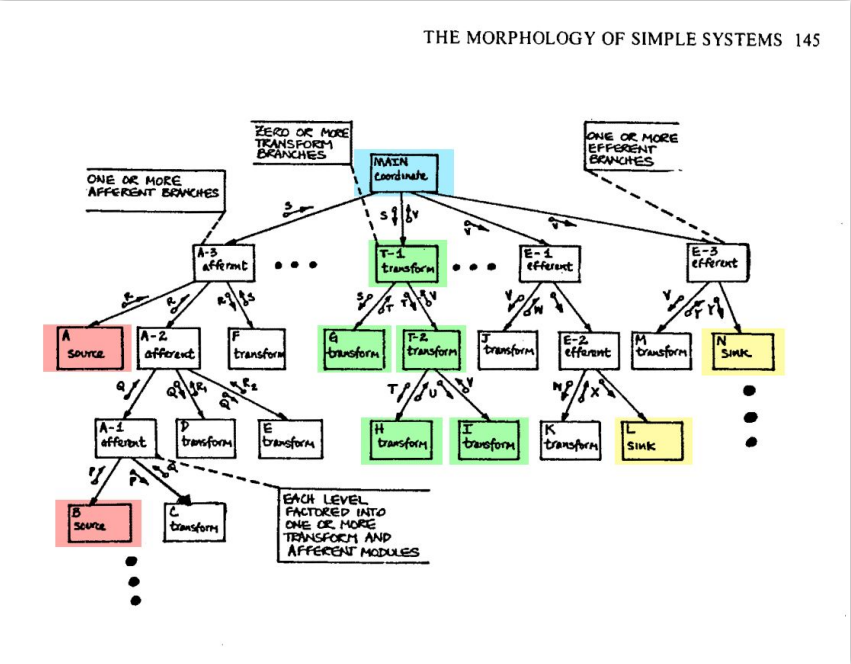
Экспресс-курс Структурного дизайна



https://tinyurl.com/stctd - книга доступна бесплатно

Сбалансированная форма системы*

Структурная схема (structured chart) во многом напоминает code property graph (CPG)



На моём сленге “управление”, “координация”, “оркестрация” - синонимы

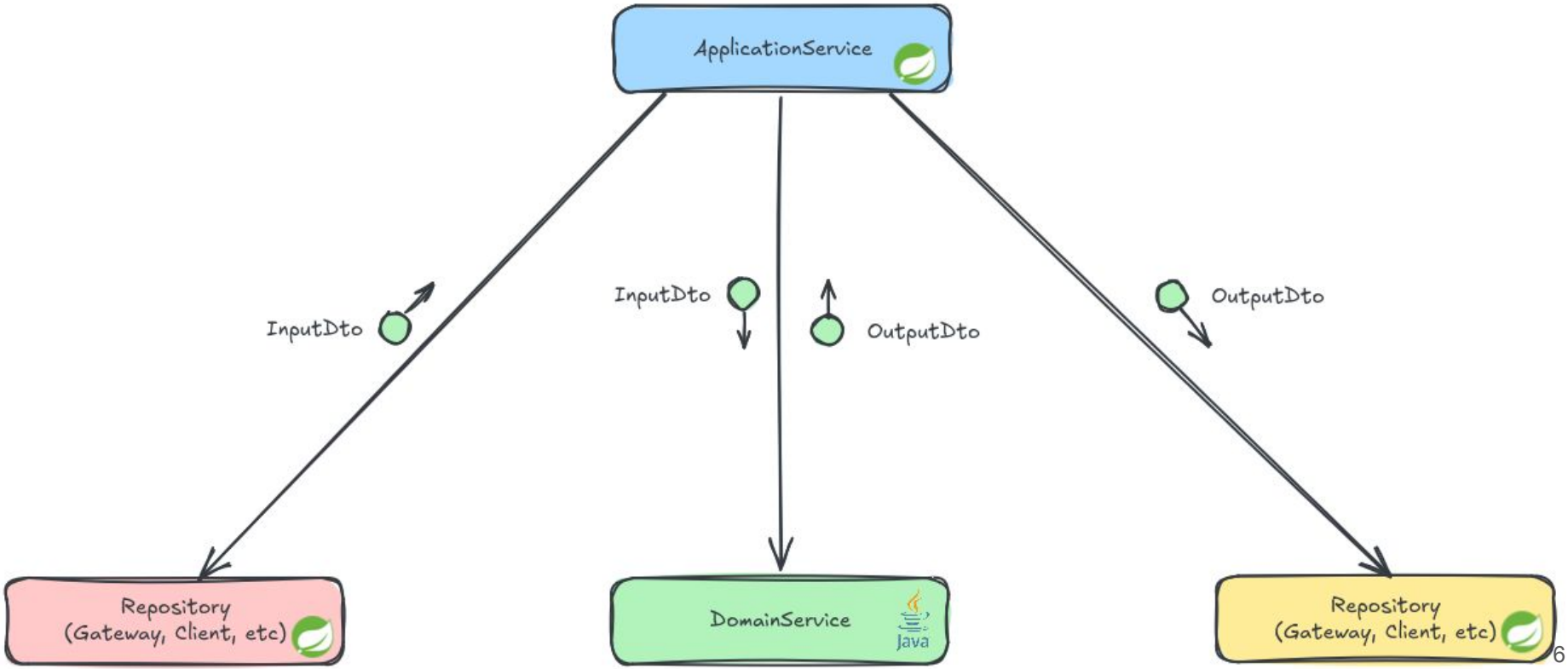
A module is a lexically contiguous sequence of program statements, bounded by boundary elements, having an aggregate identifier.

— Structured Design

На моём сленге “логика”, “бизнес-логика”, “трансформация” - синонимы

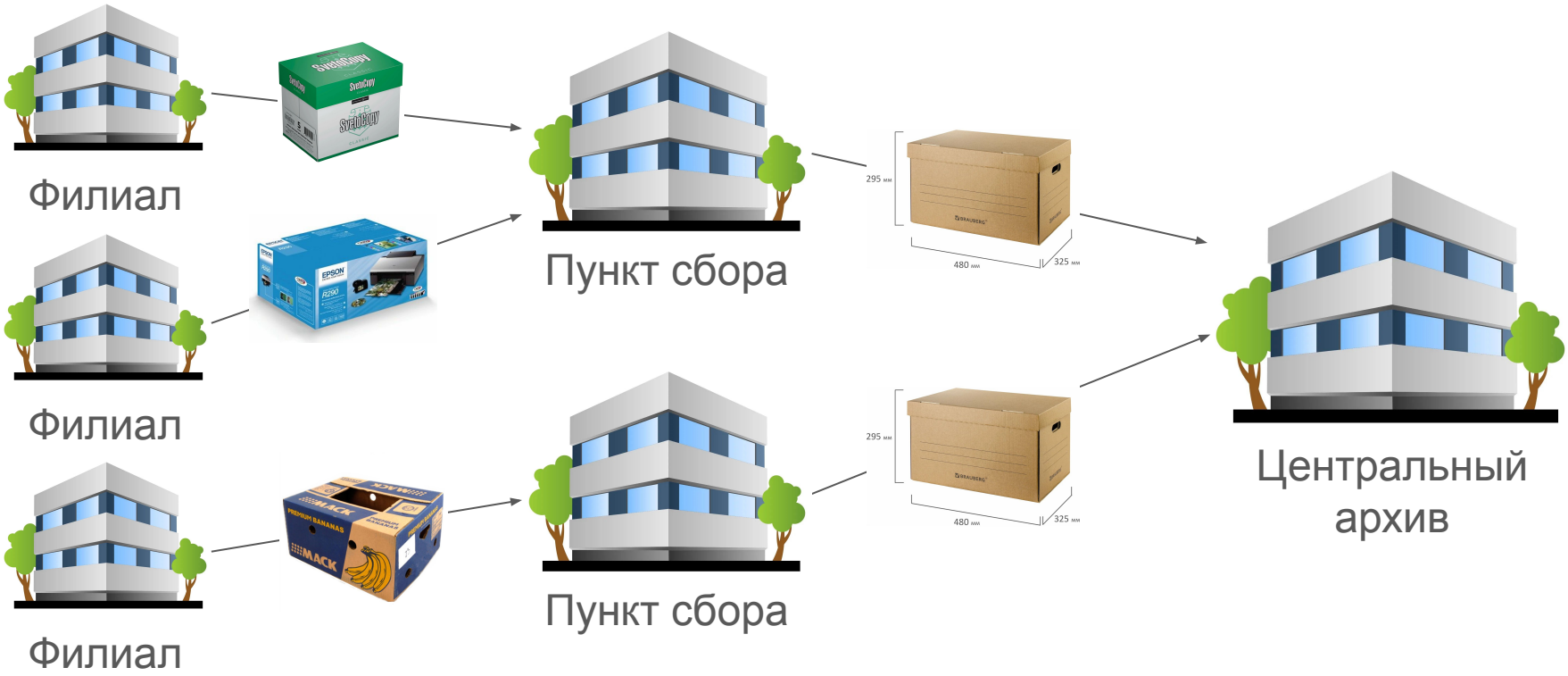
* в оригинальном тексте используется термин “transform-centered systems”, но в более поздней книге The practical guide to structured systems design используется уже более благозвучный “balanced system”

Сбалансированная форма системы 2024



Кейс №1: Project Barcoder

Project Barcoder



Ревью - точка входа

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val casesIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, casesIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}
```

Ревью - кейс переупаковки одной коробки

```
private fun repackageSingleBox(boxId: Long, cases: List<CaseDto>) {
   val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
   // привязываем отчётные интервалы коробки к разных коробам
   box.reportingIntervals = toReportingIntervals(cases)
   boxRepo.save(box)
   caseService.insertLinks(cases.map { it.id }, listOf(boxId))
}

private fun toReportingIntervals(cases: List<CaseDto>) =
   cases.flatMap {
       it.periods.map { rp ->
           ReportingInterval(
               rp.convertFromDateToDate(),
               rp.convertToDateToDate(),
               it.id
           )
       }
   }.toSet()
```

Ревью - кейс переупаковки нескольких коробок

```
private fun repackageMultipleBoxes(boxIds: List<Long>, case: CaseDto) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    // привязываем все отчётные интервалы коробки к одному коробу
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```

Чистый аккуратный код

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}

private fun repackageSingleBox(
    boxId: Long,
    cases: List<CaseDto>,
) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals = toReportingIntervals(cases)
    boxRepo.save(box)
    caseService.insertLinks(cases.map { it.id }, listOf(boxId))
}

private fun toReportingIntervals(cases: List<CaseDto>) =
    cases.flatMap {
        it.periods.map { rp ->
            ReportingInterval(
                rp.convertFromDateToDate(),
                rp.convertToDateToDate(),
                it.id
            )
        }
    }.toSet()
```

```
private fun repackageMultipleBoxes(
    boxIds: List<Long>,
    case: CaseDto,
) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```

Но есть нюанс

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}

private fun repackageSingleBox(
    boxId: Long,
    cases: List<CaseDto>,
) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals = toReportingIntervals(cases)
    boxRepo.save(box)
    caseService.insertLinks(cases.map { it.id }, listOf(boxId))
}

private fun toReportingIntervals(cases: List<CaseDto>) =
    cases.flatMap {
        it.periods.map { rp ->
            ReportingInterval(
                rp.convertFromDateToDate(),
                rp.convertToDateToDate(),
                it.id
            )
        }
    }.toSet()
```

```
private fun repackageMultipleBoxes(
    boxIds: List<Long>,
    case: CaseDto,
) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId)
              ?: throw BoxNotFound("")
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```

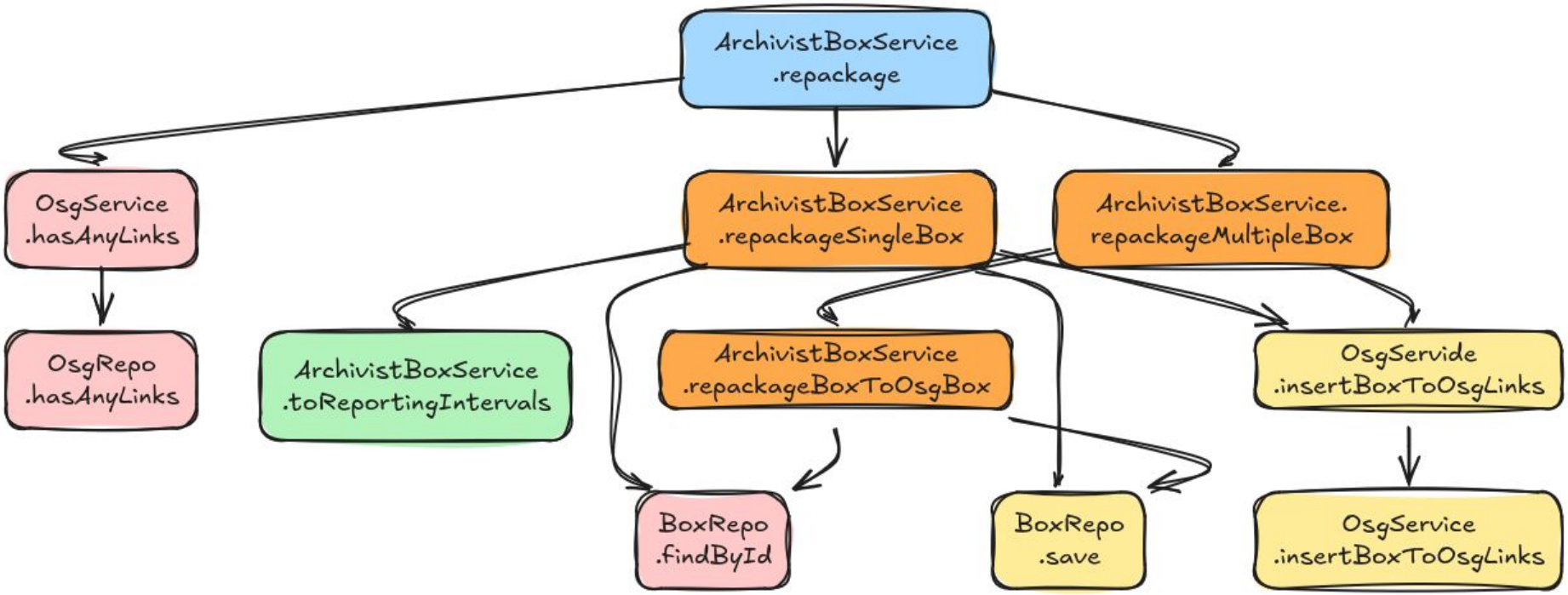
Но есть нюанс

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}

private fun repackageSingleBox(
    boxId: Long,
    cases: List<CaseDto>,
) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals = toReportingIntervals(cases)
    boxRepo.save(box)
    caseService.insertLinks(cases.map { it.id },listOf(boxId))
}

private fun toReportingIntervals(cases: List<CaseDto>) =
    cases.flatMap {
        it.periods.map { rp ->
            ReportingInterval(
                rp.convertFromDateToDate(),
                rp.convertToDateToDate(),
                it.id
            )
        }
    }.toSet()
```

Логика

IO

```
private fun repackageMultipleBoxes(
    boxIds: List<Long>,
    case: CaseDto,
) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId)
        ?: throw BoxNotFound("")
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```


ArchivistBoxService
.repackage
OsgService
.hasAnyLinks
ArchivistBoxService
.repackageSingleBox
ArchivistBoxService.
repackageMultipleBox
OsgRepo
.hasAnyLinks
ArchivistBoxService
.toReportingIntervals
ArchivistBoxService
.repackageBoxToOsgBox
OsgServide
.insertBoxToOsgLinks
BoxRepo
.findById
BoxRepo
.save
OsgService
.insertBoxToOsgLinks

```kotlin
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}

private fun repackageMultipleBoxes(
    boxIds: List<Long>,
    case: CaseDto,
) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```

```kotlin
private fun repackageSingleBox(
    boxId: Long,
    cases: List<CaseDto>,
) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals = toReportingIntervals(cases)
    boxRepo.save(box)
    caseService.insertLinks(cases.map { it.id }, listOf(boxId))
}

private fun toReportingIntervals(cases: List<CaseDto>) =
    cases.flatMap {
        it.periods.map { rp ->
            ReportingInterval(
                rp.convertFromDateToDate(),
                rp.convertToDateToDate(),
                it.id
            )
        }
    }.toSet()
```

Шаг 1: Инлайн приватных методов

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")

    }
    if (boxIds.size == 1) {
        val boxId = boxIds.single()
        val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
        box.reportingIntervals = toReportingIntervals(cases)
        boxRepo.save(box)
        caseService.insertLinks(cases.map { it.id }, listOf(boxId))
    } else {
        val case = cases.single()
        boxIds.forEach { boxId ->
            val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
            boxRepo.save(box)
        }
        caseService.insertLinks(listOf(case.id), boxIds)
    }
}
```

Шаг 2: Выделение ввода

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
   val caseIds = cases.map { it.id }
   if (caseService.hasAnyLinks(boxIds, caseIds)) {
       throw CaseAlreadyLinkedException("")

   }
   if (boxIds.size == 1) {
       val boxId = boxIds.single()
       val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
       box.reportingIntervals = toReportingIntervals(cases)
       boxRepo.save(box)
       caseService.insertLinks(cases.map { it.id }, listOf(boxId))
   } else {
       val case = cases.single()
       boxIds.forEach { boxId ->
           val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
           box.reportingIntervals.forEach {
               it.caseId = case.id
           }
           boxRepo.save(box)
       }
       caseService.insertLinks(listOf(case.id), boxIds)
   }
}
```

Шаг 2: Выделение ввода

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    val boxes = boxRepo.findByIdIn(boxIds)
    if (boxes.size == 1) {
        val box = boxes.single()
        box.reportingIntervals = toReportingIntervals(cases)
        boxRepo.save(box)
        caseService.insertLinks(cases.map { it.id }, listOf(box.id))
    } else {
        val case = cases.single()
        boxes.forEach { box ->
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
            boxRepo.save(box)
        }
        caseService.insertLinks(listOf(case.id), boxes.map { it.id })
    }
```

Шаг 3: Выделение вывода №1

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    val boxes = boxRepo.findByIdIn(boxIds)
    if (boxes.size == 1) {
        val box = boxes.single()
        box.reportingIntervals = toReportingIntervals(cases)
        boxRepo.save(box)
        caseService.insertLinks(cases.map { it.id }, listOf(box.id))
    } else {
        val case = cases.single()
        boxes.forEach { box ->
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
            boxRepo.save(box)
        }
        caseService.insertLinks(listOf(case.id), boxes.map { it.id })
    }
```

Шаг 3: Выделение вывода №1

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    val boxes = boxRepo.findByIdIn(boxIds)

    repackBoxes(boxes, cases)
}
```

```
private fun repackBoxes(
    boxes: List<Box>,
    cases: List<CaseDto>
) {
    if (boxes.size == 1) {
        val box = boxes.single()
        box.reportingIntervals = toReportingIntervals(cases)
        boxRepo.save(box)
        caseService.insertLinks(cases.map { it.id }, listOf(box.id))

    } else {
        val case = cases.single()
        boxes.forEach { box ->
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
            boxRepo.save(box)
        }
        caseService.insertLinks(cases.map { it.id }, listOf(box.id))
    }
}
```

Шаг 3: Выделение вывода №1

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    val boxes = boxRepo.findByIdIn(boxIds)

    val repackageResult = repackBoxes(boxes, cases)

    caseService.insertLinks(repackageResult.caseIds,
repackageResult.boxIds)
}
```

```
private fun repackBoxes(
    boxes: List<Box>,
    cases: List<CaseDto>
): RepackageResult {
    if (boxes.size == 1) {
        val box = boxes.single()
        box.reportingIntervals = toReportingIntervals(cases)
        boxRepo.save(box)
        return RepackageResult(cases.map { it.id }, listOf(box.id))
    } else {
        val case = cases.single()
        boxes.forEach { box ->
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
            boxRepo.save(box)
        }
        return RepackageResult(listOf(case.id), boxes.map { it.id })
    }
}

private data class RepackageResult(
    val boxIds: List<Long>,
    val caseIds: List<Long>
)
```

Шаг 4: Выделение вывода №2

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
   val caseIds = cases.map { it.id }
   if (caseService.hasAnyLinks(boxIds, caseIds)) {
       throw CaseAlreadyLinkedException("")
   }
   val boxes = boxRepo.findByIdIn(boxIds)

   val repackageResult = repackBoxes(boxes, cases)

   caseService.insertLinks(repackageResult.caseIds,
repackageResult.boxIds)
}
```

```
private fun repackBoxes(
   boxes: List<Box>,
   cases: List<CaseDto>
): RepackageResult {
   if (boxes.size == 1) {
       val box = boxes.single()
       box.reportingIntervals = toReportingIntervals(cases)
       boxRepo.save(box)
       return RepackageResult(cases.map { it.id }, listOf(box.id))
   } else {
       val case = cases.single()
       boxes.forEach { box ->
           box.reportingIntervals.forEach {
               it.caseId = case.id
           }
           boxRepo.save(box)
       }
       return RepackageResult(listOf(case.id), boxes.map { it.id })
   }
}

private data class RepackageResult(
   val boxIds: List<Long>,
   val caseIds: List<Long>
)
```

Шаг 4: Выделение вывода №2

```
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    val boxes = boxRepo.findByIdIn(boxIds)

    val repackageResult = repackBoxes(boxes, cases)

    boxRepo.saveAll(boxes)
    caseService.insertLinks(repackageResult.caseIds,
                            repackageResult.boxIds)
}
```

```
private fun repackBoxes(
    boxes: List<Box>,
    cases: List<CaseDto>
): RepackageResult {
    if (boxes.size == 1) {
        val box = boxes.single()
        box.reportingIntervals = toReportingIntervals(cases)
        return RepackageResult(cases.map { it.id }, listOf(box.id))
    } else {
        val case = cases.single()
        boxes.forEach { box ->
            box.reportingIntervals.forEach {
                it.caseId = case.id
            }
        }
        return RepackageResult(listOf(case.id), boxes.map { it.id })
    }
}

private data class RepackageResult(
    val boxIds: List<Long>,
    val caseIds: List<Long>
)
```

Шаг 5: вынос бизнес-логики из Spring-бина

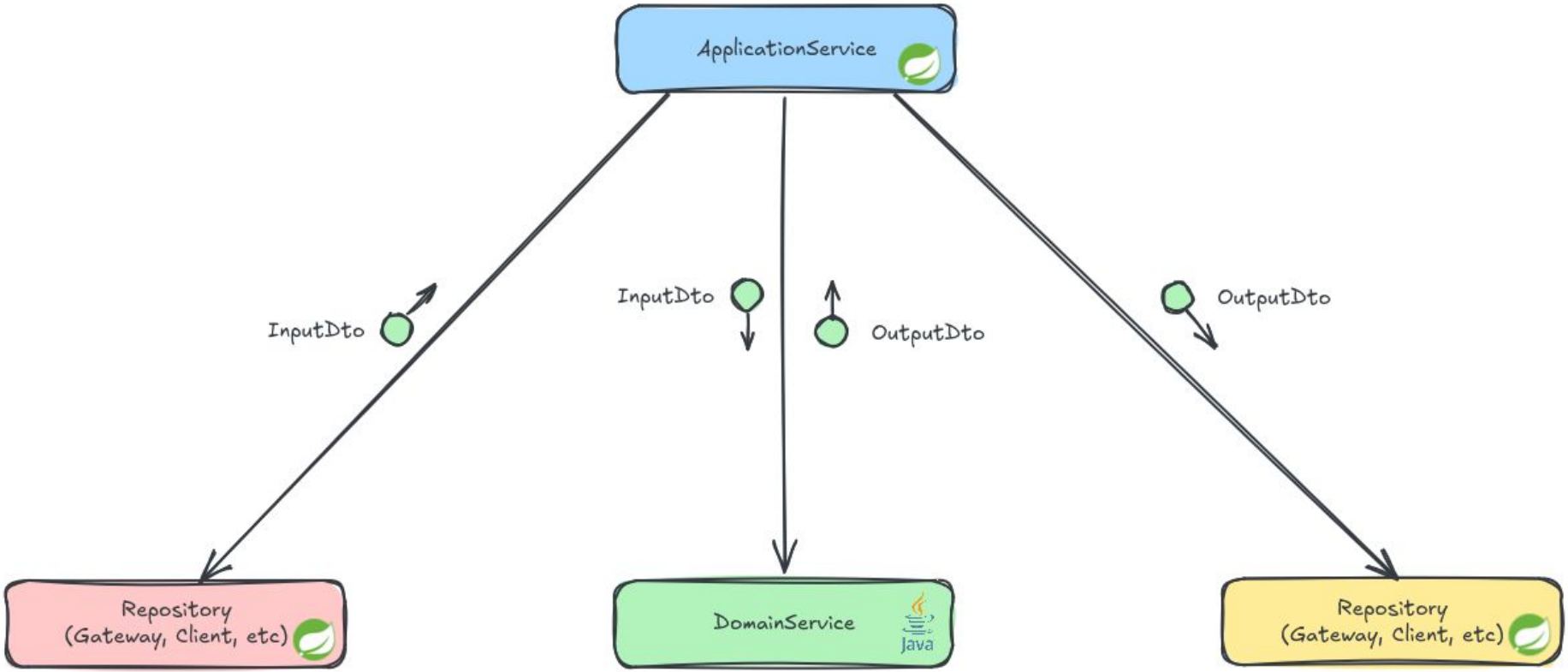
1. В статический метод
2. В метод одного из доменных объектов (сущностей, агрегатов)
3. В отдельный класс, с созданием объекта руками на каждый запрос

Шаг 5: вынос бизнес-логики из Spring-бина

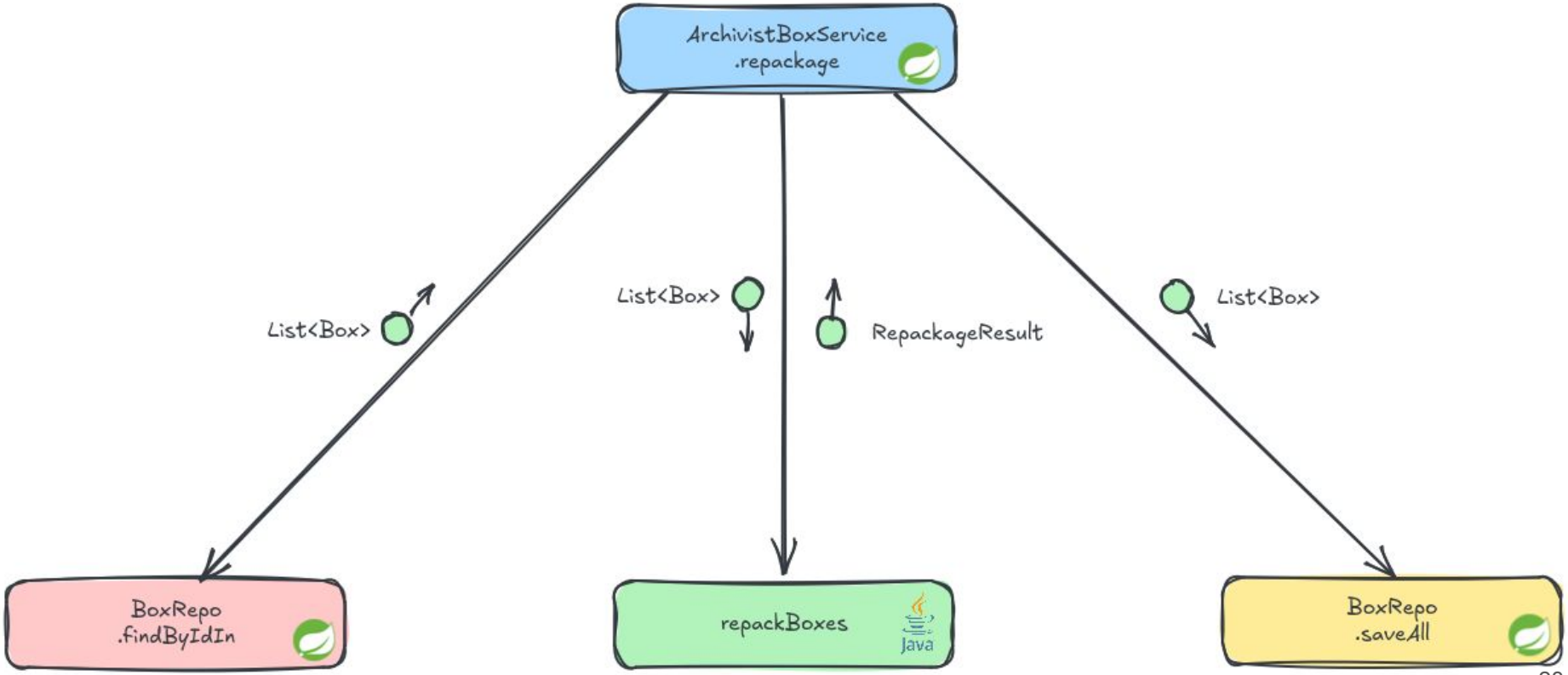
```
class ArchivistBoxService(
   private val boxRepo: BoxRepo,
   private val caseService: CaseService
) {

    fun repackage(
        boxIds: List<Long>,
        cases: List<CaseDto>
     ) {
        // ...
        val repackageResult = repackBoxes(
                                    boxes,
                                    cases)
        // ...

    }

    private fun repackBoxes(
        boxes: List<Box>,
        cases: List<CaseDto>
    ): RepackageResult {
        // ...
    }
}
```

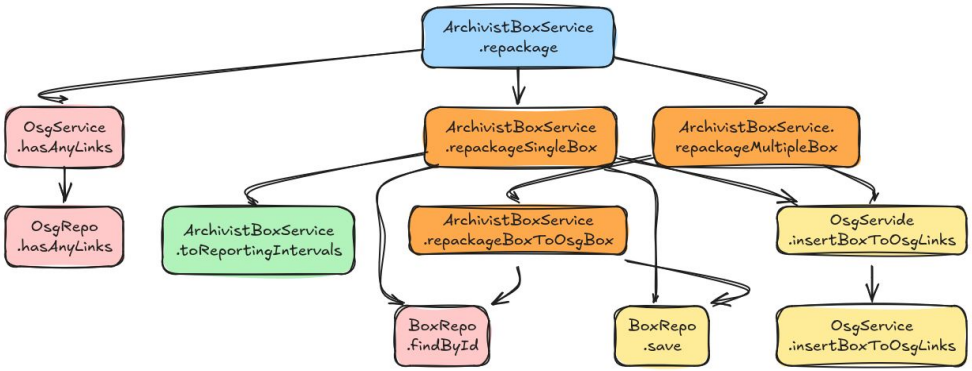
```
class ArchivistBoxService(
   private val boxRepo: BoxRepo,
   private val caseService: CaseService
) {

    fun repackage(
        boxIds: List<Long>,
        cases: List<CaseDto>
     ) {
        // ...
        val repackageResult = repackBoxes(
                                    boxes,
                                    cases)
        // ...

    }

}


private fun repackBoxes(
    boxes: List<Box>,
    cases: List<CaseDto>
): RepackageResult {
    // ...
}
```

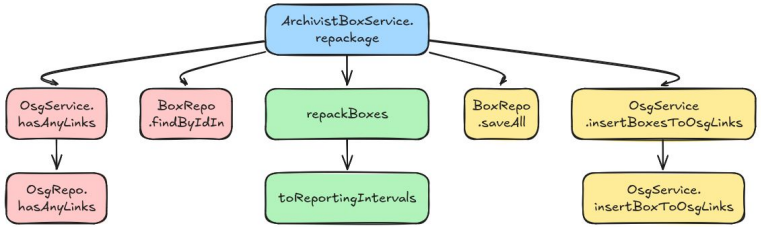
Сбалансированная форма системы 2024



Сбалансированная форма перепаковки

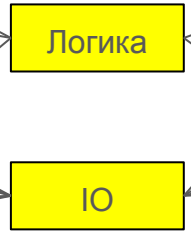



ArchivistBoxService
.repackage
OsgService
.hasAnyLinks
ArchivistBoxService
.repackageSingleBox
ArchivistBoxService.
repackageMultipleBox
OsgRepo
.hasAnyLinks
ArchivistBoxService
.toReportingIntervals
ArchivistBoxService
.repackageBoxToOsgBox
OsgServide
.insertBoxToOsgLinks
BoxRepo
.findById
BoxRepo
.save
OsgService
.insertBoxToOsgLinks


ArchivistBoxService.
repackage
OsgService.
hasAnyLinks
BoxRepo
.findByIdIn
repackBoxes
BoxRepo
.saveAll
OsgService
.insertBoxesToOsgLinks
OsgRepo.
hasAnyLinks
toReportingIntervals
OsgService.
insertBoxToOsgLinks

Было - IO и логика перемешаны

```kotlin
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {
    val caseIds = cases.map { it.id }
    if (caseService.hasAnyLinks(boxIds, caseIds)) {
        throw CaseAlreadyLinkedException("")
    }
    if (boxIds.size == 1) {
        repackageSingleBox(boxIds.single(), cases)
    } else {
        repackageMultipleBoxes(boxIds, cases.single())
    }
}

private fun repackageSingleBox(
    boxId: Long,
    cases: List<CaseDto>,
) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals = toReportingIntervals(cases)
    boxRepo.save(box)
    caseService.insertLinks(
         cases.map { it.id },
         listOf(boxId)
    )
}

private fun toReportingIntervals(cases: List<CaseDto>) =
    cases.flatMap {
        it.periods.map { rp ->
            ReportingInterval(
                rp.convertFromDateToDate(),
                rp.convertToDateToDate(),
                it.id
            )
        }
    }.toSet()
```




```kotlin
private fun repackageMultipleBoxes(
    boxIds: List<Long>,
    case: CaseDto,
) {
    boxIds.forEach { boxId ->
        repackageBoxToCase(boxId, case)
    }
    caseService.insertLinks(listOf(case.id), boxIds)
}

private fun repackageBoxToCase(boxId: Long, case: CaseDto) {
    val box = boxRepo.findByIdOrNull(boxId) ?: throw BoxNotFound("")
    box.reportingIntervals.forEach {
        it.caseId = case.id
    }
    boxRepo.save(box)
}
```

Стало - IO и логика разделены

```
// Оркестрация
fun repackage(boxIds: List<Long>, cases: List<CaseDto>) {

   // Ввод
   val caseIds = cases.map { it.id }
   if (caseService.hasAnyLinks(boxIds, caseIds)) {
       throw CaseAlreadyLinkedException("")
   }
   val boxes = boxRepo.findByIdIn(boxIds)

   // Трансформация
   val repackageResult = repackBoxes(boxes, cases)

   // Вывод
   boxRepo.saveAll(boxes)
   caseService.insertLinks(repackageResult.caseIds,
repackageResult.boxIds)

}
```

Логика

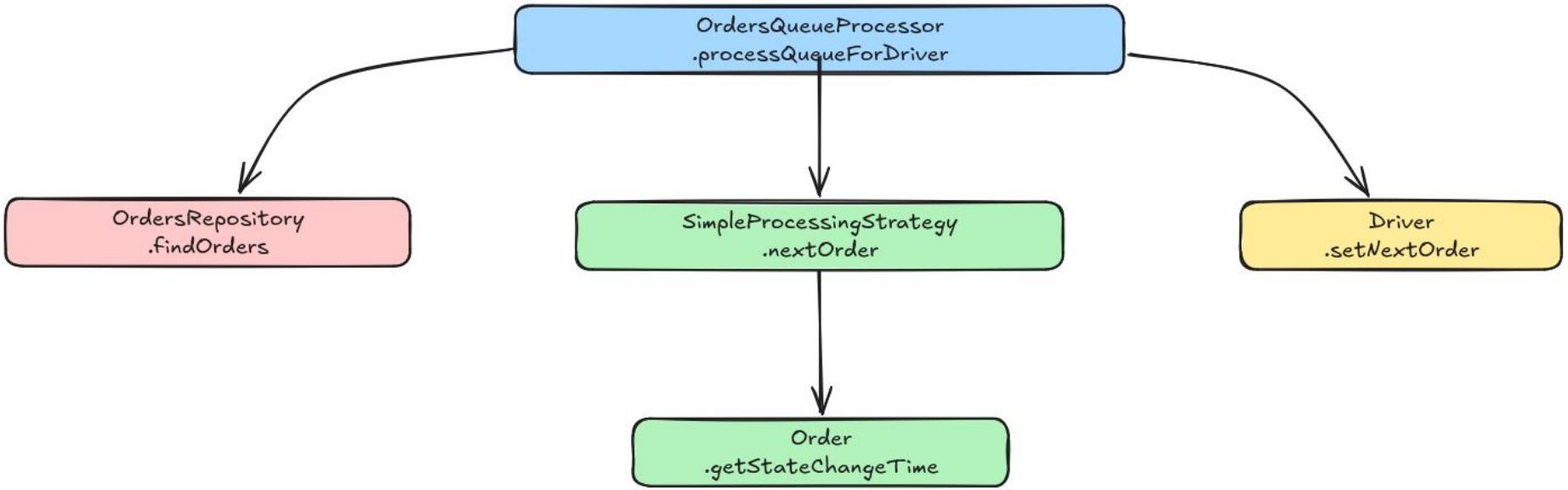
IO

```
private fun repackBoxes(
   boxes: List<Box>,
   cases: List<CaseDto>
): RepackageResult {
   if (boxes.size == 1) {
       val box = boxes.single()
       box.reportingIntervals = toReportingIntervals(cases)
       return RepResult(cases.map { it.id }, listOf(box.id))
   } else {
       val case = cases.single()
       boxes.forEach { box ->
           box.reportingIntervals.forEach {
               it.caseId = case.id
           }
       }
       return RepResult(listOf(case.id), boxes.map { it.id })
   }
}

private data class RepackageResult(
   val boxIds: List<Long>,
   val caseIds: List<Long>
)

private fun toReportingIntervals(cases: List<CaseDto>) =
   cases.flatMap {
       it.periods.map { rp ->
           ReportingInterval(
               rp.convertFromDateToDate(),
               rp.convertToDateToDate(),
               it.caseId
           )
       }
   }.toSet()
```

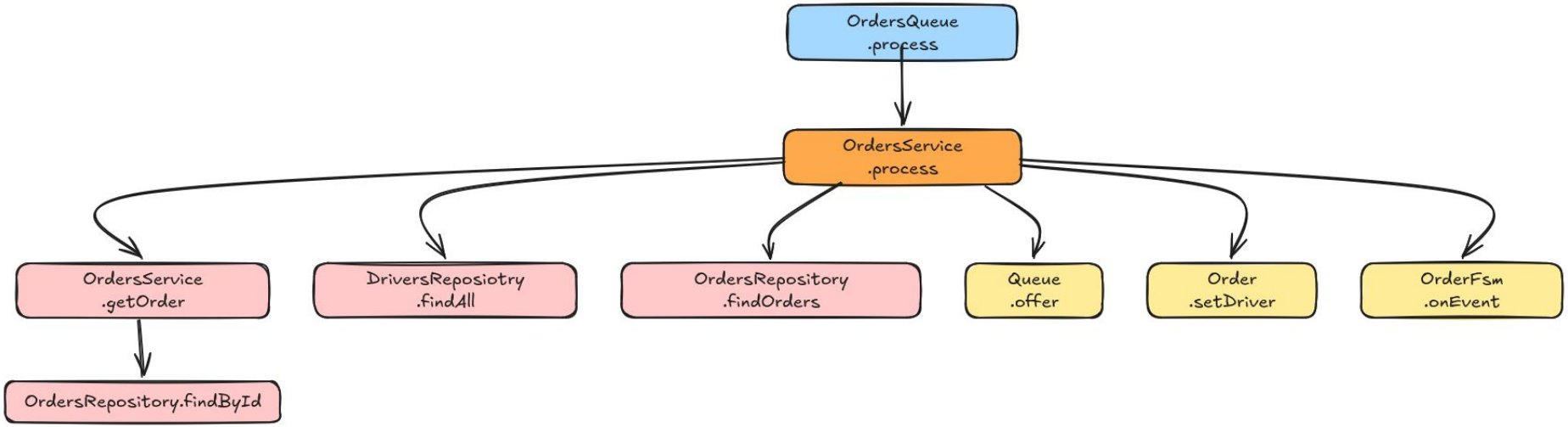
Кейс №2: Project Daniel

Project Daniel

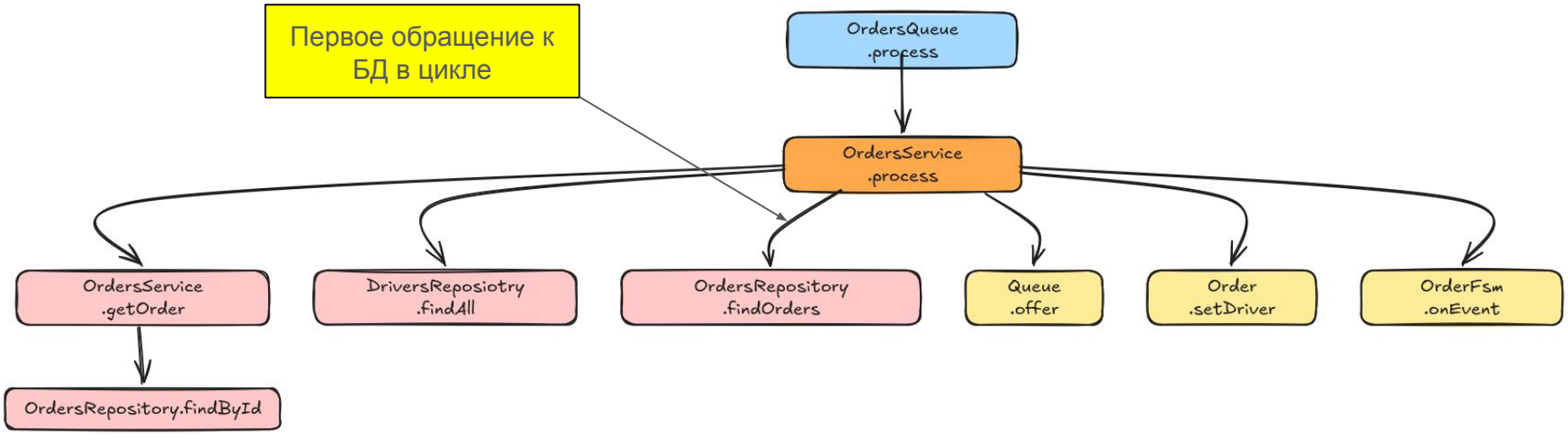
Операция назначения водителя t0 + 10 дней



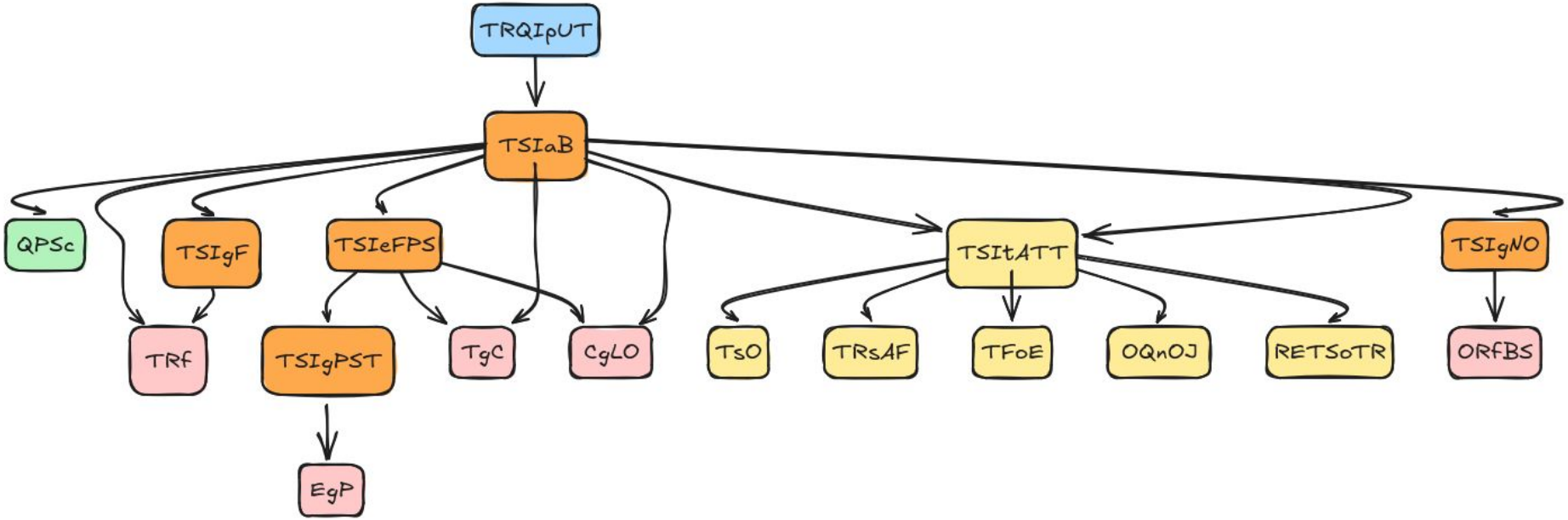
Операция назначения водителя t0 + 20 дней



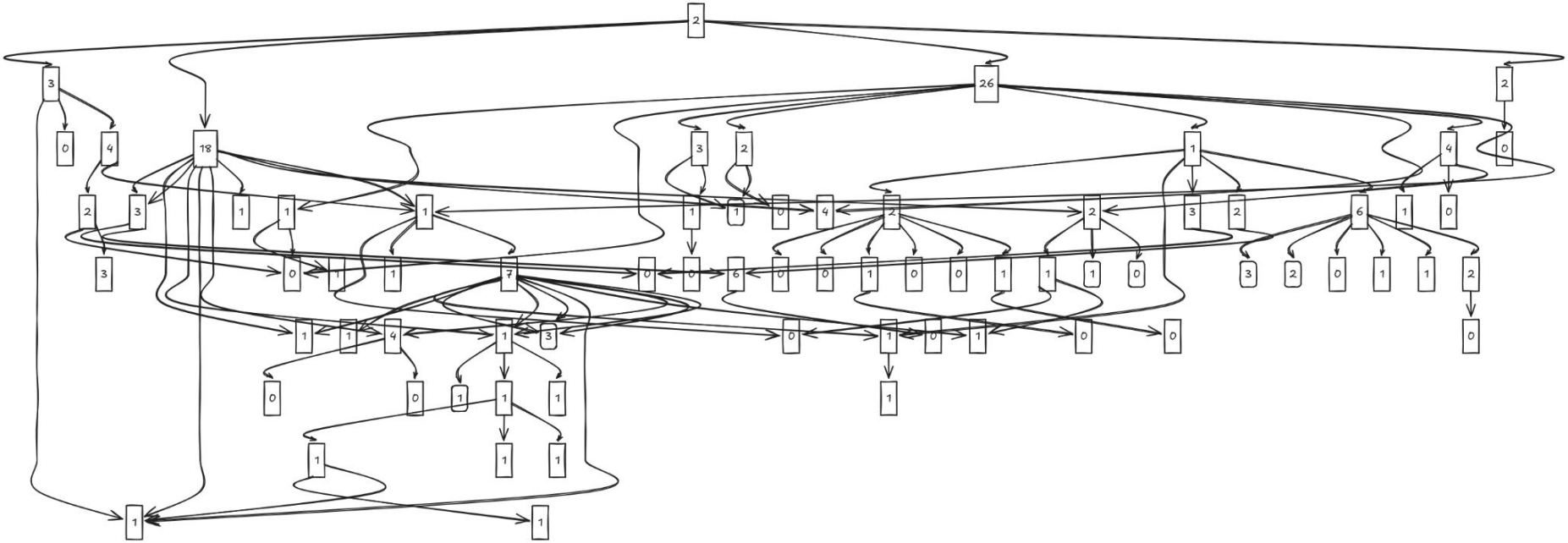
Операция назначения водителя t0 + 20 дней



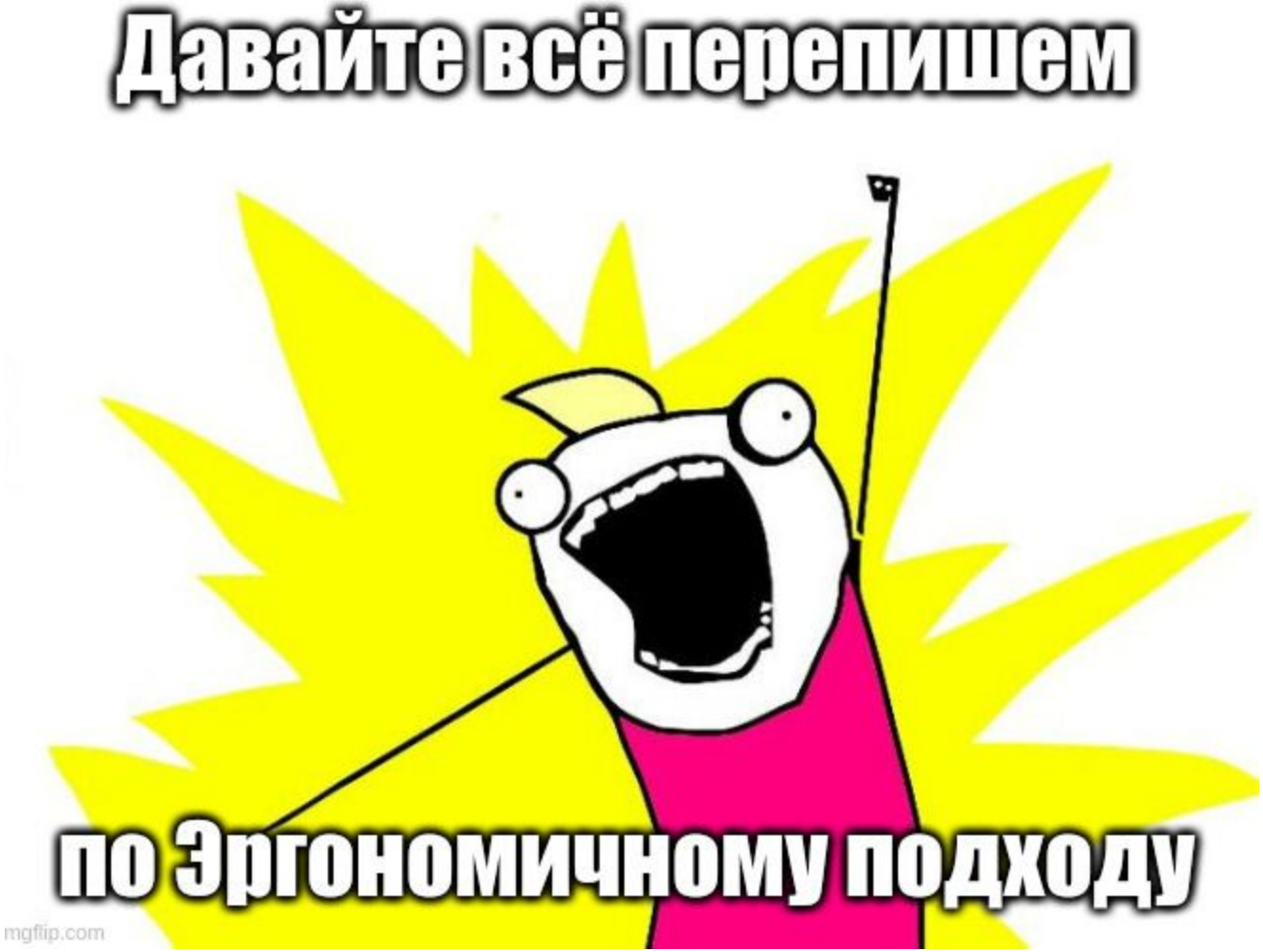
Операция назначения водителя t0 + 3 месяца



Операция назначения водителя t0 + 5 лет



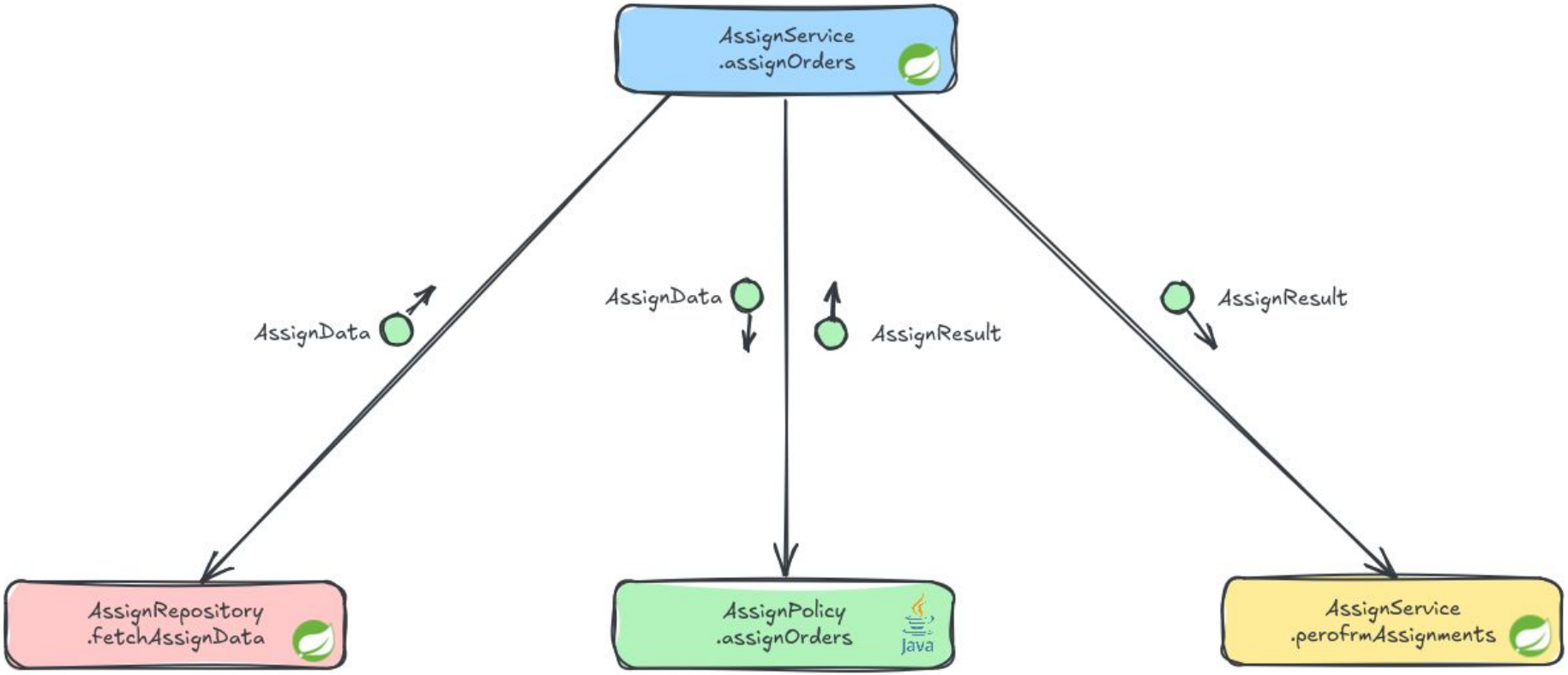

UMMM, YEAH...
I'M GONNA NEED YOU TO FIX THAT ASAP
memecrunch.com


Давайте всё перепишем
по Эргономичному подходу
mgflip.com

Было

```
public void assignOrders() {
    List<Order> orders = getAssignCandidates();
    if (orders.isEmpty()) {
        return;
    }
    var ordersGroups = orders.stream()
           .groupBy(this::isEligibleForPersonalService));
    handlePersonalAssingments(ordersGroups.get(PERSONAL));
    handleStandardAssignMents(ordersGroups.get(STANDARD));
}
```

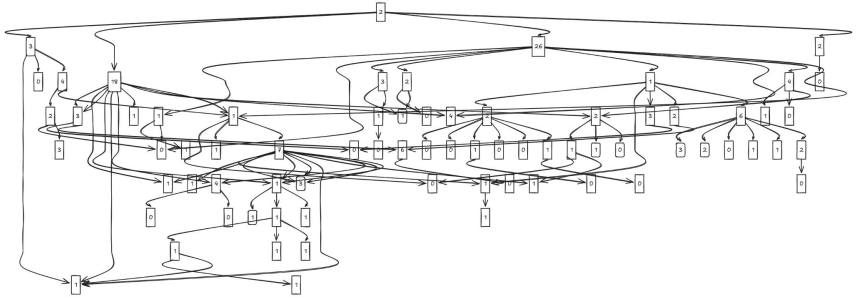
Сбалансированная форма назначения водителей

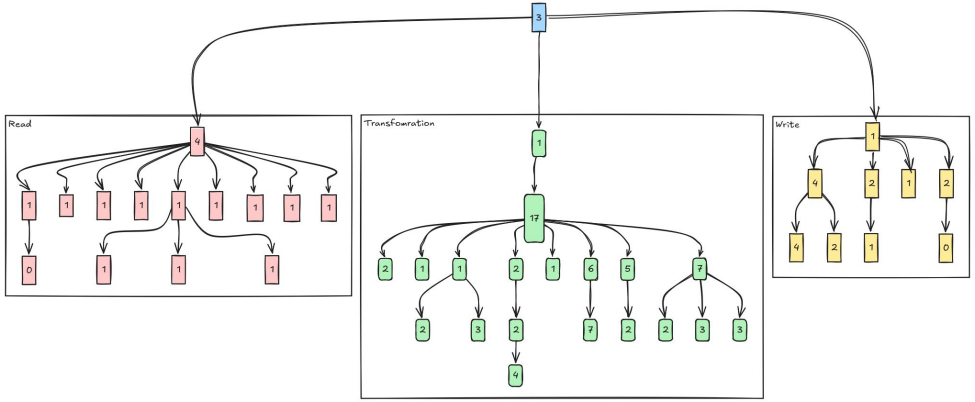


Стало

```
public void assignOrders() {
   AssignData assignData =
       assignRepository.fetchAssignData();

   final AssignResult assignResult =
       new AssignPolicy(assignData)
           .assignOrders(Instant.now());

   performAssignments(assignResult);
}

public class AssignData {
   public final List<OrderAssignData> queue;
   public final ActiveDrivers activeDrivers;
   public final Map<Long, Rule> assignRules;
}

public class AssignResult {
   public final List<Assignment> assignments;
   public final Set<Order> notAssigned;
}
```

Структурная схема после





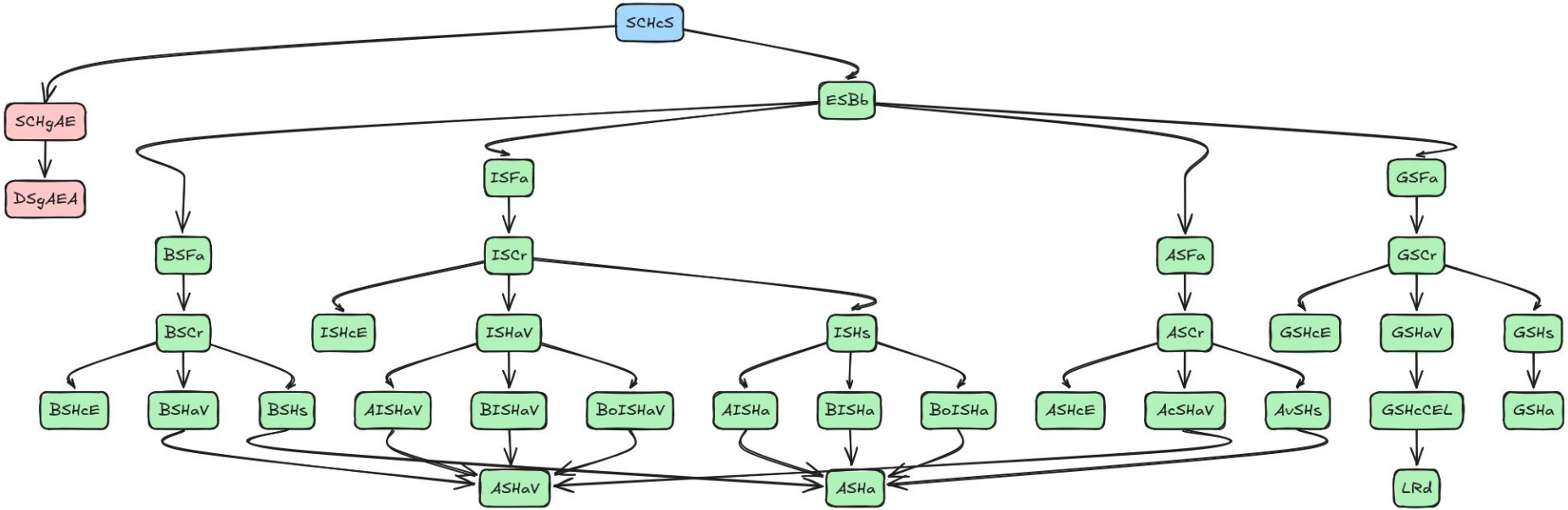
Производительность (заказов в секунду)	5	1500

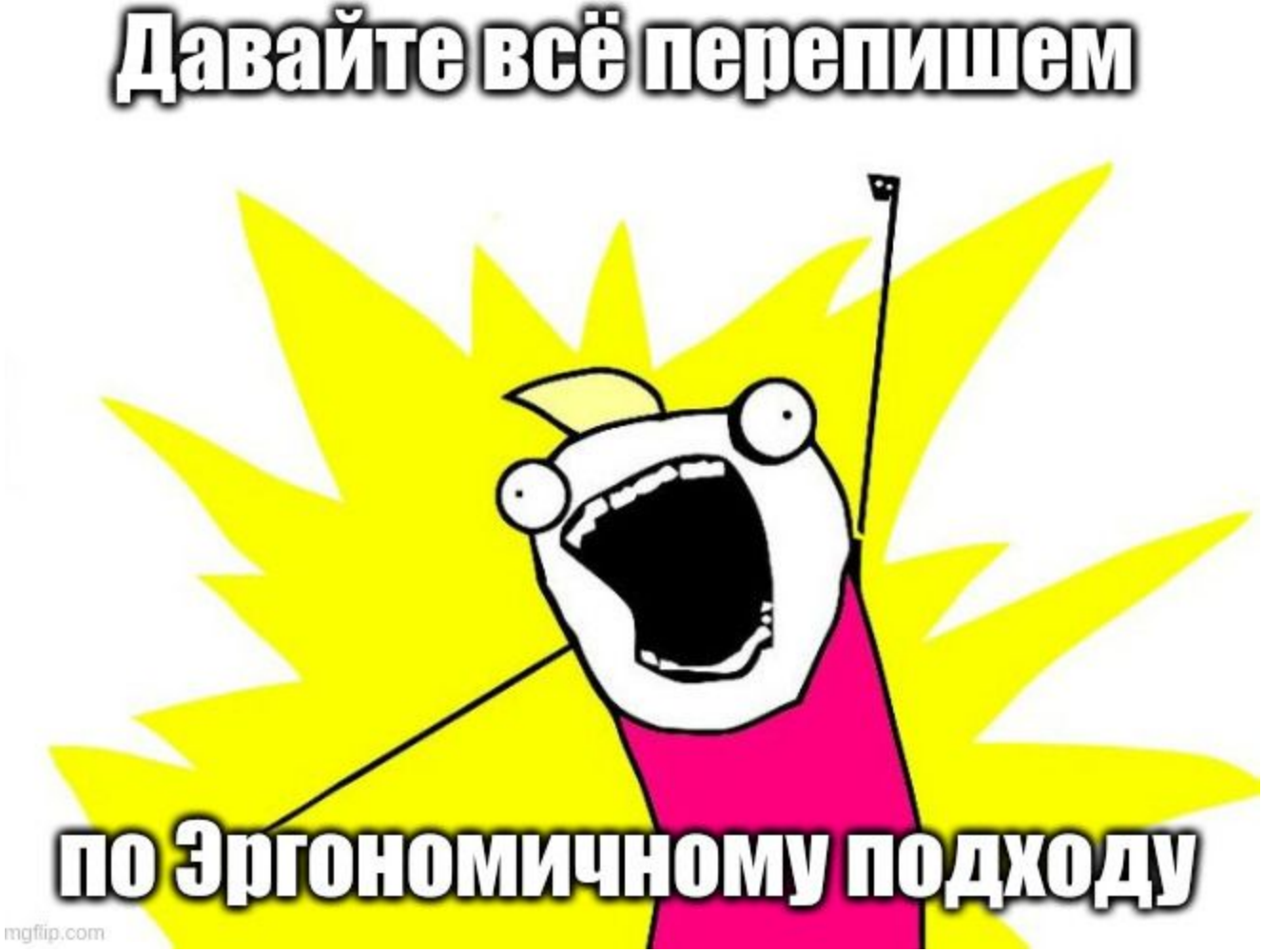
Кейс №3: Project E

Project E


DDD
Шаблоны ООД
Микросервисы
MediatR
Скучающий мидл
Сложность задачи
Задача

Отчёт по дневнику энтерпрайз эдишн

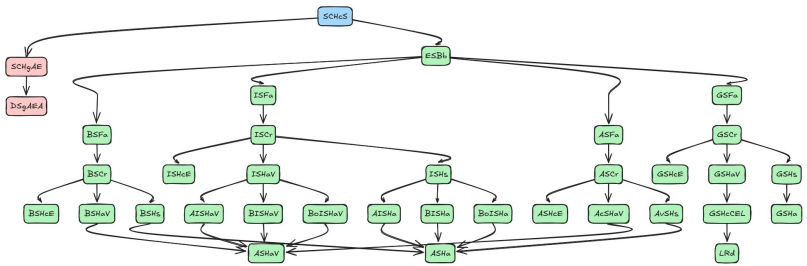


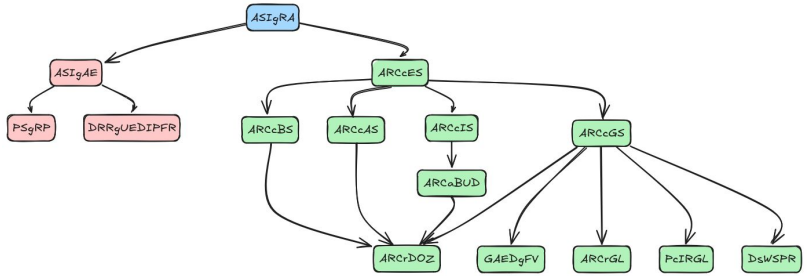

Давайте всё перепишем
по Эргономичному подходу
mgflip.com

Project E

Стек и архитектура	C#, Микросервисы, вертикальная архитектура	Kotlin, монолит, функциональная архитектура
Реализация базовой версии	189 ч/дней мидла	145 ч/дней (~75%) юниора
Покрытие тестами	0 тестов	100% покрытие эндпоинтов

Подробности: https://t.ly/lodx2


SCHcS
SCHgAE
DSgAEA
ESBL
BSFa
BSCr
BSHcE
BSHaV
BSHs
ISFa
ISCr
ISHcE
ISHaV
AISHaV
BISHaV
BoISHaV
ISHs
AISHa
BISHa
BoISHa
ASHaV
ASHa
ASFa
ASCr
ASHcE
AcSHaV
AvSHs
GSFa
GSCr
GSHcE
GSHaV
GSHcCEL
LRd
GSHs
GSHa


ASIgRA
ASIgAE
PSgRP
DRRgUEDIPFR
ARCcES
ARCcBS
ARCcAS
ARCcIS
ARCaBUD
ARCcGS
ARCrDOZ
GAEDgFV
ARCrGL
PcIRGL
DsWSPR

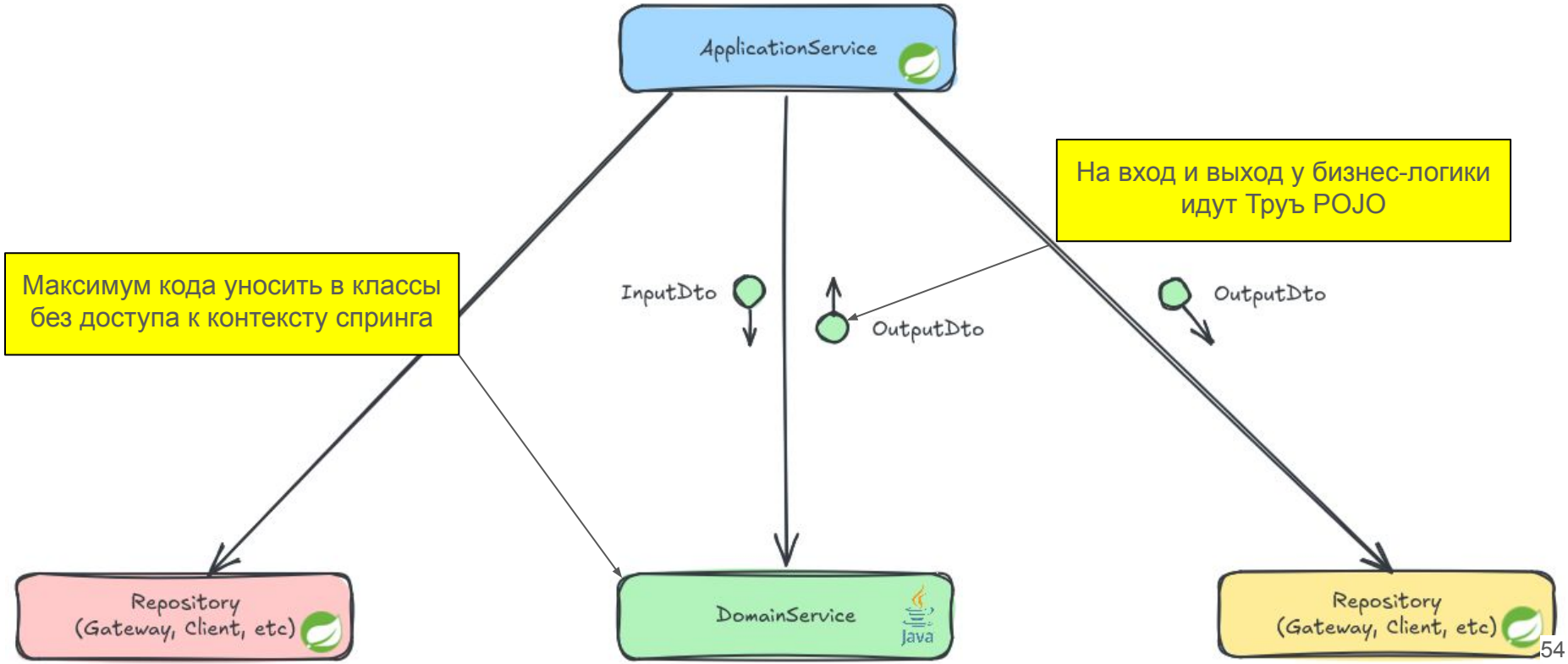
Project E

	Последние 3 месяца доработок v1 - v1.1	**Последние 3 месяца доработок v1.1 - v1.2**
Количество выполненных задач	14 шт. (за 526 ч/час)	52 шт. (за 497 ч/час)
Медианные трудозатраты на задачу	16 ч/час	5 ч/час
Среднее количество багов на задачу	1.5 шт.	0.5 шт.

Подробности: https://t.ly/lodx2

Итоги

TLDR - суть подхода



TLDR - область применения

Подходит когда

1. Весь набор данных помещается в памяти
2. Львиная доля данных используется во всех ветках бизнес-логики
3. Координация организована в императивном стиле (один поток или корутины)

TLDR - Минусы*

1. Надо учиться
   a. SQL
   b. Техники функционального программирования
2. Надо писать больше кода
   a. SQL
   b. DTO

* Относительно обычного подхода на базе Spring и JPA с “прогулками” по графу объектов с корнями из Spring Data-репозиториев

TLDR - плюсы

1. Повышение эффективности работы команды
2. Повышение эффективности работы кода

Как научиться делать сбалансированные системы?

Как научиться делать сбалансированные системы?

Как научиться делать сбалансированные системы?

Подборка книг по прагматичному ФП

- Мой канал в Телеграм (https://t.me/ergonomic_code)
- Grokking Simplicity (https://t.ly/YNCr0)
- Принципы юнит-тестирования (https://t.ly/B_flt)
- Java to Kotlin: A Refactoring Guidebook (https://t.ly/RhCaG)
- Domain Modeling Made Functional (https://t.ly/uwFIt)
- Data-Oriented Programming (https://t.ly/7_Njg)
- Структура и интерпретация компьютерных программ (https://t.ly/QaMDu)
- Functional Design: Principles, Patterns, and Practices (https://t.ly/foQRZ)
- Structured Design (https://t.ly/BHhpa)

Реальный проект в прагматичном ФП-стиле

https://github.com/ergonomic-code/Trainer-Advisor