УДК 004.438Kotlin+004.432.42+004.42.046

МИГРАЦИЯ С JAVA НА KOTLIN: ПОШАГОВОЕ РУКОВОДСТВО И СЛОЖНОСТИ

Пестрякова М. А., студент
Научный руководитель – ст. преподаватель Кондратенок Е. В.
Белорусский национальный технический университет
Минск, Республика Беларусь

Аннотация. Рассматривается комплексное исследование процесса миграции программных проектов с языка Java на язык Kotlin. На основе анализа современных научно-технических публикаций рассматривается стратегическая целесообразность перехода, обусловленная преимуществами Kotlin в контексте разработки надежных и сопровождаемых backend-систем. Детально освещаются этапы подготовки проекта к миграции, включая аудит кодовой базы и инфраструктуры. Анализируются различные стратегии проведения миграции, в частности, инкрементальный подход, позволяющий минимизировать риски. Выявляются характерные подводные камни и частые ошибки, возникающие на практике, такие как генерация неидиоматичного кода и проблемы при интеграции с популярными фреймворками. Предлагаются рекомендации по использованию инструментария и лучших практик, направленные на обеспечение бесшовного и эффективного перехода, итогом которого становится существенное повышение производительности разработки и качества кода без нарушения работоспособности существующего приложения.

Ключевые слова: миграция программного обеспечения, Java, Kotlin, JVM, интероперабельность, инкрементальный рефакторинг, backend-разработка.

Миграция программного обеспечения с одной технологической платформы на другую представляет собой сложный и многогранный процесс, сопряженный с необходимостью решения ряда методологических и технических задач. В контексте современной экосистемы JVM переход с устоявшегося языка Java на более молодой и выразительный Kotlin является актуальным трендом, обусловленным стремлением к повышению производительности разработки, снижению объемов шаблонного кода и усилению статической типобезопасности. Несмотря на декларируемую

JetBrains стопроцентную интероперабельность между Kotlin и Java, что теоретически позволяет осуществлять миграцию поэтапно, практическая реализация данного процесса содержит множество нюансов и потенциальных подводных камней. Успешность перехода критически зависит от корректного планирования, выбора оптимальной стратегии рефакторинга, глубокого понимания различий в парадигмах и идиомах обоих языков, а также осознания последствий для существующей инфраструктуры проекта, таких как системы сборки, инструменты непрерывной интеграции и библиотеки зависимостей. Как отмечается в исследованиях, современные языки, такие как Kotlin, предлагают ряд преимуществ, включая более лаконичный синтаксис, расширенную систему типов для повышения надежности кода и мощные средства функционального программирования, что в совокупности способствует увеличению производительности разработки и снижению количества ошибок на этапе компиляции [1]. Обоснование необходимости перехода проистекает из стремления использовать эти конкурентные преимущества для поддержания высокой скорости развития и долгосрочной поддерживаемости крупных программных комплексов, особенно в условиях, когда традиционные Java-решения начинают демонстрировать ограничения, связанные с избыточной вербозностью и унаследованными подходами.

Фундаментальным этапом, предваряющим непосредственную миграцию, является тщательная подготовка, которая, по аналогии с опытом миграции в других технологических контекстах [2], требует всестороннего аудита существующей кодовой базы и инфраструктуры. На этой стадии критически важно обеспечить полную работоспособность системы сборки, интегрировав поддержку Kotlin в используемые инструменты, такие как Maven или Gradle, а также провести регрессионное тестирование для формирования надежного базиса для последующих изменений. Не менее значимым является анализ зависимостей проекта на предмет их совместимости с Kotlin, поскольку, несмотря на декларируемую интероперабельность, отдельные библиотеки могут демонстрировать непредсказуемое поведение в смешанной среде. Этот подготовительный процесс можно рассматривать как проектирование новой архитектурной реальности проекта, где, подобно принципам, изложенным при проектировании сложных систем [3], закладывается основа для бесшовной интеграции двух языковых парадигм.

Выбор адекватной стратегии миграции является определяющим фактором для минимизации операционных рисков. Наиболее распро-

страненным и рекомендуемым подходом является инкрементальный, или поэтапный, переход, при котором новые модули или функциональные блоки сразу пишутся на Kotlin, а существующий Java-код подвергается конвертации постепенно, по мере необходимости его модификации. Данная стратегия, схожая с подходами, применяемыми при смене фреймворков [4], позволяет распределить нагрузку на команду и снизить disruptive-воздействие на процесс разработки. Ключевым принципом здесь является сохранение постоянной работоспособности приложения после каждого этапа миграции, что обеспечивается скрупулезным модульным и интеграционным тестированием. Альтернативной, но более рискованной стратегией является «биг-бэнг»-конвертация, которая может быть рассмотрена для небольших, хорошо изолированных сервисов, однако в крупных проектах она сопряжена с высокой вероятностью возникновения каскадных ошибок и длительной стабилизации.

Практический процесс миграции неизбежно сопряжен с рядом подводных камней и типичных ошибок. Одной из наиболее существенных проблем является иллюзия полной семантической эквивалентности сгенерированного кода, когда автоматизированные инструменты конвертации формально переводят Java-конструкции в Kotlin, но не производят их последующую оптимизацию в соответствии с идиоматикой нового языка. Это приводит к появлению так называемого «Kotlin-синтаксиса с Java-акцентом», который не раскрывает его истинных преимуществ и может даже ухудшить читаемость. Другой частой ошибкой является некорректная работа с nullable-типами, которая, будучи фундаментальным аспектом системы типов Kotlin, при неправильном подходе порождает избыточное использование операторов безопасного вызова или необоснованные non-null утверждения, нивелируя преимущества компиляторной проверки на null-безопасность. Проблемы могут возникать и на уровне взаимодействия с фреймворками, такими как Spring, особенности конфигурации которых, подробно изучаемые в контексте создания приложений [5], требуют адаптации под Kotlin-специфичные механизмы, например, coroutines для асинхронного выполнения.

Для успешного преодоления указанных challenges необходима опора на специализированные инструменты и строгое следование лучшим практикам. Интегрированная среда разработки IntelliJ IDEA предлагает встроенный, хотя и не идеальный, конвертер кода, использование кото-

рого должно сопровождаться обязательным последующим рефакторингом, направленным на идиоматизацию. К числу лучших практик относится приоритетное использование возможностей стандартной библиотеки Kotlin для замены циклов и проверок на более выразительные функции высшего порядка, а также грамотное проектирование доменных моделей с применением data-классов. Важнейшим аспектом является обучение команды, поскольку переход – это не только смена синтаксиса, но и освоение новых концепций, таких как корутины для асинхронного программирования, что позволяет эффективно использовать современные асинхронные фреймворки, подобные Ktor, чей сравнительный анализ представлен в научной литературе [4].

Проведенный анализ позволяет констатировать, что процесс миграции с Java на Kotlin представляет собой не просто техническую замену одного языка программирования другим, но комплексную организационно-технологическую трансформацию, требующую системного подхода и тщательного планирования. Как демонстрирует опыт исследований в смежных областях, будь то миграция в мобильной разработке или сравнительный анализ backend-фреймворков, успех подобных инициатив определяется глубиной проработки фундаментальных этапов, включающих аудит кодовой базы, модернизацию инфраструктуры сборки и формирование надежной тестовой страховки. Стратегическая целесообразность такого перехода, обоснованная в контексте эволюции языков для backend-разработки, заключается в долгосрочных выгодах от повышения выразительности кода, усиления статической типобезопасности и сокращения объема шаблонных конструкций, что в совокупности способствует ускорению темпов разработки и повышению сопровождаемости программного продукта. Однако реализация этой стратегии сопряжена с необходимостью преодоления ряда объективных сложностей, от риска генерации неидиоматичного кода при автоматической конвертации до проблем интероперабельности в смешанных проектах и необходимости адаптации команд к новым парадигмам, таким как активное использование корутин.

Список использованных источников

1. Коновалов, М. Ю. Языки программирования для backend-разработки: плюсы, минусы и перспективы / М. Ю. Коновалов [и др.] // Ответственный редактор. – 2025. – № [номер]. – С. 16.

2. Чумиков, И. В. Миграция iOS-приложения с кроссплатформенного фреймворка на нативную архитектуру на SwiftUI: инженерный кейс / И. В. Чумиков // Актуальные исследования. – 2025. – № 26. – С. 21–26.

3. Перевалова, С. Л. Проектирование мобильного приложения на Android для симуляции криптотрейдинга / С. Л. Перевалова, А. А. Яковлев // Лучшие практики общего и дополнительного образования по естественнонаучным и техническим дисциплинам : материалы конф. – 2023. – С. 421–424.

4. Орлов, Д. Ю. Сравнительный анализ фреймворков Ktor и Laravel для backend серверов / Д. Ю. Орлов // Междунар. науч.-техн. конф. молодых ученых БГТУ им. В. Г. Шухова, посвящ. 170-летию со дня рождения В. Г. Шухова : материалы конф. – 2023. – С. 248–252.

5. Протченко, К. О. Создание приложения с использованием фреймворка Spring: приложения по бронированию мест в офисе / К. О. Протченко, Е. О. Винокурова, С. М. Писарев // Внедрение передового опыта и практическое применение результатов инновационных исследований : материалы конф. – 2023. – С. 17–19.

УДК 378.13

МОТИВАЦИЯ И СТИМУЛИРОВАНИЕ ТРУДА ПРЕПОДАВАТЕЛЕЙ ВУЗОВ В УСЛОВИЯХ ИННОВАЦИОННОГО РАЗВИТИЯ ВЫСШЕГО ОБРАЗОВАНИЯ

Соболенко И. А., ст. преподаватель
Кондратова И. И., канд. техн. наук, доцент
Белорусский национальный технический университет
Минск, Республика Беларусь

Аннотация. Мотивация и стимулирование труда ППС высшего учебного заведения в условиях инновационного развития необходимо для его эффективного функционирования и надлежащего предоставления качественных образовательных услуг.

Ключевые слова: мотивация, стимулирование, высшее образование, инновационное развитие.

Постоянно изменяющиеся социально-экономические условия развития белорусского общества предъявляют повышенные требова-