

Текст
ЛАРИСА СТАНКЕВИЧ,
ИДЕОЛОГ РЕШЕНИЙ АРТ-ФИНТЕХ

Получается парадокс: технология, обещавшая бизнесу свободу, порой сама становится источником ограничений. Значит ли это, что от Low-Code пора отказаться? Не обязательно.

ЗА ЧТО ПОЛЮБИЛИ LOW-CODE

У Low-Code были веские основания для популярности. Он предложил бизнесу то, чего всегда не хватает, — время. Быстрее вывести продукт на рынок, изменить тариф, настроить процесс, не превращая каждую небольшую задачу в многонедельный проект с техническим заданием и очередью к разработчикам.

Допустим, банку нужно изменить комиссию за операцию. Специалист открывает форму, выбирает условия, сохраняет настройки. Если необходимая возможность уже заложена в систему, задача решается без программирования. Визуальная логика также облегчает обсуждение между бизнесом и ИТ, передачу знаний и сопровождение.

Но за похожими интерфейсами могут стоять принципиально разные механизмы. В классическом Low-Code распространены два подхода. Первый — параметрический: поставщик заранее определяет, что разрешено настраивать. Пока требования банка укладываются в предложенный набор, все работает быстро. Понадобилось новое условие — приходится ждать доработки.

За год снижение интереса крупных российских компаний к Low-Code уменьшилось:

66% → 34%

Второй подход — алгоритмический. Пользователю дают специальный макроязык, на котором можно описать более сложную логику. Гибкости становится больше, но появляется другая зависимость: язык и исполняющий его интерпретатор принадлежат экосистеме поставщика. Ограничения поставщика остаются.

ЦЕНА УДОБСТВА

Банковский бизнес не стоит на месте: сегодня требуется пересмотреть комиссию, завтра — внедрить дополнительную проверку, послезавтра — изменить бизнес-процесс. Но если каждое новое требование выходит за рамки возможностей платформы и вынуждает обращаться к вендору, Low-Code постепенно теряет то, ради чего его внедряли, — скорость и гибкость изменений.

Не менее важна производительность. Интерпретация параметров и макрокоманд создает дополнительные издержки, которые при высокой транзакционной нагрузке могут влиять на время отклика системы.

Но наиболее чувствительный вопрос — информационная безопасность. По данным Банка России, в 2024 году регулятор получил более 750 сообщений о компьютерных атаках на финансовые организации. Уязвимости ПО поставщиков создают дополнительные риски, а закрытость платформ затрудняет независимый аудит кода. При этом ответственность за<безопасность данных и непрерывность банковских операций остается за банком.

Наконец, архитектура. Классический Low-Code нередко строится на монолитной архитектуре, а монолит, как известно, неповоротлив: даже локальные изменения могут затрагивать всю систему. Ситуацию усугубляет закрытый технологический стек — собственные языки и механизмы исполнения затрудняют интеграцию с привычными инструментами разработки, тестирования и развертывания.

И здесь возникают вопросы, которые прежде оставались в тени: насколько банк свободен в развитии собственной системы? Сможет ли он масштабировать ее, менять архитектуру, контролировать исходный код и не зависеть от решений поставщика? Постепенно на первый план выходит уже не удобство настройки, а технологическая независимость.

ОТ ОБЕЩАНИЙ К РЕАЛЬНОСТИ

Показательна и динамика рынка. По данным исследования BPMSoft, в 2025 году доля крупных российских компаний, использовавших

или планировавших внедрение Low-Code, составляла 66%, а в 2026 году — 34%. Эти цифры заставляют задуматься: одного обещания быстрой разработки рынку уже недостаточно. Банкам нужны не просто удобные инструменты настройки, а свобода развивать собственные системы, контролировать их архитектуру и не зависеть от технологических ограничений поставщика.

> **На первый план выходит уже не удобство настройки, а технологическая независимость**

ОТ НАСТРОЕК — К ИСХОДНОМУ КОДУ

Но что если проблема вовсе не в самой идее Low-Code, а в том, что происходит после настройки? Что если сохранить все, за что Low-Code когда-то полюбили, — наглядность, простоту и скорость, — но отказаться от закрытого механизма исполнения?

Именно такой подход реализован в АРТ-Финтех: Low-Code становится не средой исполнения бизнес-логики, а инструментом генерации полноценного исходного кода. В основе решения — четыре последовательных этапа.

Первый — создание Low-Code-среды. Технические специалисты самого банка-заказчика определяют архитектуру решения, состав микросервисов, API, формы и шаблоны генерации. Таким образом, банк самостоятельно формирует среду разработки под собственные задачи и требования, сохраняя контроль над ее архитектурой и дальнейшим развитием.

Второй — бизнес-настройка. Специалисты предметной области работают с понятными формами, справочниками и параметрами. Например, сотрудник банка самостоятельно задает условия тарификации, а последовательность действий при необходимости описывает с помощью BPMN-схемы. Писать программный код ему не требуется.

Третий — генерация исходного кода. Настройки и визуальные модели автоматически преобразуются в код на современных языках программирования. Принципиально важно, что сгенерированный код не редактируется вручную: при изменении требований корректируются настройки или шаблоны, после чего генерация запускается заново. Это позволяет избежать расхождений между моделью и ее реализацией.

Четвертый этап — проверка и развертывание. Сгенерированный код поступает в Git, проходит сборку, автоматизированные тесты и проверки безопасности. Затем готовое приложение разворачивается в рабочей среде как самостоятельный микросервис, взаимодействующий с другими компонентами через API.

ЧТО ЭТО МЕНЯЕТ ДЛЯ БАНКА

Главное изменение — банк получает не просто инструмент настройки, а возможность самостоятельно управлять развитием своих ИТ-систем. На практике это дает несколько преимуществ:

- Скорость вывода продуктов на рынок. Бизнес быстрее запускает новые продукты и меняет действующие, не привлекая разработчиков к каждой типовой операции.
- Контроль над технологиями. ИТ-команда сохраняет контроль над архитектурой и исходным кодом, получая возможность самостоятельно развивать и сопровождать систему.
- Информационная безопасность. Служба ИБ может анализировать сгенерированный код и проверять его привычными инструментами.
- Гибкость и масштабируемость. Микросервисная архитектура позволяет развивать, обновлять и масштабировать отдельные компоненты, не затрагивая каждый раз всю систему. Это особенно важно для банков, где изменения должны происходить непрерывно, а обслуживание клиентов — оставаться стабильным.

Пожалуй, в этом и заключается главное достижение: сохранить все, за что когда-то полюбили Low-Code, и преодолеть ограничения, которые со временем стали сдерживать его развитие.

Ведь эволюция технологий — это не отказ от прошлого, а умение взять из него лучшее и шагнуть дальше.

Прежний Low-Code умер. Но сама идея обрела новую жизнь — со свободой разработки, возможностями промышленного кода и без прежних границ. Б.О