12+

Кирилл Ледовский

ТОМ 1

Мы теряем контроль над сложной ERP. Что делать?

МОЯ ERP ДЛЯ АНАЛИТИКА

Кирилл Ледовский

Мы теряем контроль над сложной ERP. Что делать?

«Издательские решения»

Ледовский К.

Мы теряем контроль над сложной ERP. Что делать? /
К. Ледовский — «Издательские решения»,

Эта книга серии «Моя ERP для аналитика» посвящён проблеме утраты управляемости сложной ERP при сохранении её формальной работоспособности. Книга исследует ситуации, когда код, технические задания, настройки, отчёты, тесты и знания сотрудников физически существуют, однако предприятие уже не способно доказательно объяснить, почему система работает именно так и какое управленческое решение стоит за конкретным реквизитом, алгоритмом, доработкой или показателем.

Содержание

Мы теряем контроль над сложной ERP. Что делать?

Кирилл Ледовский

ISBN 978-5-0071-1683-1
Создано в интеллектуальной издательской системе Ridero

Введение. Как работающая ERP превращается в систему, которую никто уже не может объяснить

Сложная ERP редко сообщает о потере контроля аварией. Она продолжает запускаться утром, документы проводятся, обмены живут, производство получает задания, отчёты строятся, а бухгалтерия закрывает период. Именно поэтому проблема способна годами оставаться невидимой: техническая работоспособность создаёт ощущение, что с системой всё в порядке. **Самая опасная стадия начинается тогда, когда ERP ещё работает, а предприятие уже не может доказательно объяснить, почему она работает именно так.**

Обычно разрыв обнаруживается через маленький практический вопрос. Почему это поле обязательно? Откуда в отчёте появилась именно эта цифра? Можно ли удалить старую доработку? Почему похожая типовая возможность не позволяет спокойно отказаться от собственного механизма? Что на самом деле доказал старый тест? Какая версия системы действительно исполняется в продуктиве? После смены аналитика или подрядчика такие вопросы внезапно требуют ТЗ, кода, переписки, истории решений, знаний пользователей и текущей практики. **По отдельности это разные задачи сопровождения; вместе — один и тот же разрыв между решением предприятия и его цифровой реализацией.**

При этом знания физически никуда не обязаны исчезать. Документы остаются в папках, задачи — в системе управления работами, история изменений — в репозитории, записи совещаний — в архиве, код — в конфигурации, а опыт — у сотрудников. Проблема возникает тогда, когда эти фрагменты перестают складываться в один маршрут: от реальной потребности и принятой нормы к данным, программному механизму, проверке и фактическому исполнению. **Контроль исчезает не вместе с файлами, а вместе со связями между нормой, смыслом, данными, реализацией и доказательством результата.**

В этой книге для различения таких слоёв используются три рабочих понятия. Цифровой мастер — утверждённая модель нормы, то, как предприятие осознанно решило работать. Цифровая тень — фактическая жизнь этой нормы в реальных процессах и системе. Цифровая нить — трассируемая связь происхождения решения, его реализации, проверки и последующих изменений. Эти понятия нужны не ради новой терминологии, а ради одного инженерного вопроса: **Кому принадлежит смысл цифровой системы предприятия — самой организации или людям, документам и программному коду, между которыми этот смысл случайно распределился?**

Поэтому перед нами не руководство по кнопкам и интерфейсам ERP и не каталог настроек, которые якобы сделают большую систему простой. Промышленное предприятие сложно само по себе, и его ERP неизбежно наследует значительную часть этой сложности. Наша задача другая — исследовать границу между деятельностью предприятия и её цифровой реализацией и понять, в какой момент эта граница перестаёт быть объяснимой. **Сложность сама по себе не является лабиринтом; лабиринтом становится сложность, у которой потеряна карта.**

Именно поэтому понадобилось пятнадцать глав. В первой части потеря контроля проявляется как разрушение корпоративной памяти и появление нескольких конкурирующих версий одной реальности. Во второй мы спускаемся к конкретным объектам ERP и проверяем, сохраняют ли поле, цифра, код и старая доработка свой бизнес-адрес. Третья часть показывает цену потерянного происхождения: обновление, тестирование, работа с продуктивом, смена подрядчика и даже применение ИИ начинают с расследования того, что система должна была значить. Четвёртая часть собирает карту выхода, а эпилог делает один шаг за пределы ERP и

спрашивает, зачем восстановленная карта понадобится предприятию дальше. **Разные симптомы нужны здесь не для коллекции проблем, а для доказательства одной причины.**

Читать эту книгу полезнее всего не как отвлечённую методологию, а держа в голове один живой механизм собственной системы: старое расширение, обязательный реквизит, управленческий показатель, маршрут согласования или тест. Для него можно последовательно спрашивать: из какой деятельности он возник, какую норму защищает, где материализован в данных и ERP, чем подтверждён и как исполняется сейчас. Если на любом шаге ответ приходится угадывать по косвенным следам, перед нами уже не локальная нехватка документации, а разрыв цифровой памяти. **Книга должна работать не как ещё один архив знаний, а как способ вернуть предприятию способность самостоятельно восстанавливать доказательный ответ.**

Начнём с ситуации, в которой этот разрыв становится особенно заметным. Человек уходит, система остаётся, все основные файлы вроде бы переданы — и только через некоторое время предприятие обнаруживает, что часть смысла всё это время находилась не в ERP и не в документах, а в способности конкретного аналитика связывать их между собой. **Первый симптом ERP-лабиринта — момент, когда память о системе оказывается старше и сложнее памяти любого отдельного сотрудника.**

Часть I. Когда предприятие начинает терять собственную память

Первые три главы смотрят на систему не через код, а через память организации: кто способен объяснить происхождение решения, какая версия деятельности считается нормой и почему документы, знания людей и фактическая практика постепенно расходятся. **Потеря контроля сначала выглядит как потеря памяти: всё необходимое вроде бы сохранено, но причинная связь между фрагментами перестаёт принадлежать предприятию.**

Глава 1. Что делать со сложной ERP: уволился аналитик ERP

Почему вместе с человеком предприятие теряет часть причинной архитектуры системы, хотя код, ТЗ, задачи и записи совещаний остаются на месте

В сложной ERP-системе есть неприятный момент, который обычно обнаруживается не в день увольнения специалиста, а значительно позже. Аналитик заканчивает работу, передаёт текущие задачи, показывает преемнику документы, объясняет незавершённые вопросы и оставляет достаточно аккуратное наследство. На следующий день ERP запускается точно так же, как раньше: документы проводятся, обмены работают, производство получает задания, отчёты строятся, бухгалтерия закрывает период. **Никакой технической аварии не произошло, поэтому предприятие вполне естественно считает, что передача прошла нормально.**

Через несколько недель появляется первый вопрос, на который неожиданно трудно ответить. Пользователь спрашивает, почему определённый реквизит обязателен именно в этом документе. Затем разработчик находит в расширении старую проверку и хочет понять, можно ли её убрать. Во время подготовки обновления обнаруживается собственный механизм, очень похожий на функциональность новой типовой версии, но никто уже не уверен, действительно ли они решают одну и ту же задачу. Каждый случай по отдельности выглядит небольшим, однако постепенно становится видно, что вместе с человеком предприятие потеряло не программу, а **часть объяснения собственной программы**.

Самое неприятное состоит в том, что эта потеря почти незаметна до момента следующего изменения. Работоспособность системы не доказывает сохранность знания о ней. Код продолжает исполняться независимо от того, помнит ли кто-нибудь, почему он появился, какую проблему предприятия должен был решить и какое управленческое правило за ним стоит.

Передача дел может быть выполнена правильно и всё равно оказаться неполной

При уходе опытного специалиста обычно делают вполне разумные вещи. Ему предлагают актуализировать технические задания, передать незакрытые задачи, показать исходный код, собрать инструкции, сохранить протоколы испытаний и провести несколько встреч с человеком, который примет систему. **Если проект велся дисциплинированно, остаются также история задач, репозиторий разработки, записи совещаний и архив проектной документации.**

Поэтому проблема редко заключается в полном отсутствии информации. Наоборот, в зрелой корпоративной системе информации обычно слишком много. Разные её части распределены между ERP, таск-трекером, техническими заданиями, Git или хранилищем ERP, электронной почтой, корпоративным порталом, протоколами, тестами и видеозаписями встреч. Формально практически всё сохранено, однако новый аналитик довольно быстро обнаруживает, что найти документ и восстановить **связь между документами** — совершенно разные задачи.

Представим дополнительный реквизит производственного документа. В техническом задании можно прочитать, когда его добавили и как он должен заполняться. В коде видно, куда его значение передаётся. В отчёте легко обнаружить, где оно используется. **Но из этих трёх источников ещё не обязательно понятно, почему предприятие когда-то решило вообще ввести эту аналитику.** Возможно, она потребовалась для разделения направлений деятельности, затем стала участвовать в распределении затрат, а позже ещё и вошла в управленческую отчётность. Если эту причинную последовательность знал прежний аналитик, один

телефонный разговор связывал разрозненные артефакты в понятную историю. После его ухода все файлы остаются на месте, но восстановление той же связи начинает требовать полноценного исследования.

Документация хранит результаты решений, но не всегда историю их происхождения

В этой точке возникает естественная мысль: значит, нужно просто лучше документировать доработки. **Частично это действительно помогает, однако полностью проблему не решает.** Даже хорошее техническое задание обычно создаётся уже после аналитической работы и фиксирует согласованный результат: что система должна делать, какие данные использовать, какие ограничения учитывать и какой результат считается правильным. За его пределами могут оставаться первоначальный вопрос пользователя, несколько отвергнутых вариантов, причины выбора конкретного подхода и обстоятельства, из-за которых предприятие согласилось на определённый компромисс.

С видеозаписями совещаний ситуация ещё нагляднее. Предприятие может дисциплинированно записывать каждую конференцию и через три года иметь сотни часов прекрасно сохранившегося материала. **Но если архитектурное решение было принято на восьмидесятой минуте одной встречи и никто не знает, что именно этот фрагмент послужил основанием будущей доработки, сама запись почти не помогает.** Информация существует физически, но не включена в корпоративную память как осмысленная связь.

Репозиторий исходного кода решает другую задачу. Он великолепно показывает, что технически изменилось, кто внёс изменение и когда оно появилось. При необходимости разработчик способен восстановить последовательность коммитов и понять эволюцию программного механизма. Однако репозиторий не обязан знать, какой хозяйственный или процессный разрыв заставил предприятие эту эволюцию начать. **Точно так же таск-трекер отлично управляет работой команды, но карточка со статусом Done сама по себе не превращается в долговременное объяснение бизнес-причины реализованного поведения.**

Поэтому проблема находится не между «есть документация» и «нет документации». Она проходит между **архивом информации** и **проверяемой памятью системы**.

В голове хорошего аналитика существует то, чего часто нет ни в одном файле

Опытный аналитик постепенно строит довольно сложную внутреннюю карту. **Он знает не только объекты конфигурации и пользовательские сценарии.** Он помнит, что один статус появился из-за конфликта между двумя процессами; что определённая аналитика технически необязательна, но без неё перестаёт собираться управленческий показатель; что странный алгоритм распределения является сознательным компромиссом, а не неудачной разработкой; что конкретную проверку нельзя удалять, хотя с точки зрения кода она выглядит избыточной.

Это знание очень похоже не на документ, а на граф. В нём конкретная операция предприятия связана с пользовательским вопросом, пользовательский вопрос — с требованием, требование — с решением, решение — с технической реализацией, реализация — с тестом, а тест — с версией, которая когда-то была выпущена в продуктив. Когда возникает новый вопрос, специалист не читает весь архив проекта. **Он почти мгновенно проходит по нескольким связям внутри собственной памяти и вытаскивает нужный контекст.**

Именно поэтому иногда кажется, что документация предприятия прекрасно работает. Пока рядом находится человек, знающий историю, она действительно работает: документ становится для него ключом, который активирует значительно более широкое знание. После ухода специалиста выясняется, что часть смысла находилась не в документе, а в том, **как этот человек умел связать документ со всем остальным**.

Незаменимым оказывается не сам аналитик. **Незаменимой оказывается связь, которую предприятие не успело отделить от его памяти.**

Что именно теряет предприятие

Техническое устройство старого механизма обычно можно восстановить. **Хороший разработчик откроет конфигурацию, посмотрит расширения, запросы, движения, обработчики событий и интеграционные вызовы, после чего достаточно точно объяснит фактическое поведение программы.** Чем качественнее код и архитектура, тем быстрее это произойдёт.

Значительно сложнее восстановить бизнес-причину этого поведения. Почему механизм вообще понадобился, какое место деятельности предприятия не поддерживала прежняя версия системы, **какие варианты решения обсуждались и почему предприятие приняло именно этот — на эти вопросы исходный код отвечает плохо.** Ещё труднее восстановить методологический слой: например, почему значение определённой аналитики стало обязательным, какой хозяйственный смысл оно несёт и какой показатель потеряет достоверность, если это значение перестать собирать.

Третий потерянный слой связан с доказательностью. **Даже если найдено старое техническое задание и понятен код, нужно установить, какую именно версию проверяли, что считали успешным результатом и изменялась ли реализация после испытания.** Старый протокол может быть абсолютно корректным документом и одновременно уже ничего не доказывать относительно действующей версии.

Таким образом после ухода человека предприятие может продолжать знать **как** работает механизм и постепенно переставать знать **почему** он работает именно так.

Цифровая нить должна заменить человека как единственное место соединения истории

В книге **«Цифровая трансформация в интеллектуальное предприятие»** для связи происхождения, нормы, факта, решения и последующих изменений используется понятие **цифровой нити**. Для сложной ERP это понятие особенно удобно, потому что позволяет посмотреть на историю системы не как на папку доработок, а как на последовательность связанных управленческих событий.

Допустим, производственное подразделение сталкивается с ситуацией, которую действующая система поддерживает недостаточно. **Аналитик локализует проблему в конкретном процессе и формулирует проверяемое требование.** После обсуждения предприятие принимает определённое решение, решение получает техническую реализацию, реализация проверяется в конкретной версии, после чего изменение вводится в эксплуатацию. Позднее тот же механизм может быть исправлен, дополнен, частично заменён новой типовой возможностью или окончательно выведен из использования.

Если эта последовательность сохранена как связанная история, новому специалисту не требуется помнить весь проект. Он может начать с действующего механизма и пройти назад к тому решению, которое когда-то его породило, а затем ещё дальше — к бизнес-причине. Если такой связи нет, человеку приходится реконструировать её по сохранившимся следам, причём каждый следующий аналитик создаёт уже собственную интерпретацию прошлого.

Цифровая нить поэтому является не ещё одним журналом изменений. Её ценность в том, что она удерживает **происхождение смысла изменения**.

Почему новый аналитик неизбежно создаёт новую версию прошлого

Когда связанной истории нет, преемник начинает исследование. Он читает технические задания, изучает код, разговаривает с пользователями и постепенно собирает достаточно убедительную картину. Это нормальная профессиональная деятельность, причём сильный специалист способен восстановить очень многое. **Проблема состоит в том, что восстановленная картина уже не является исходной историей; она является интерпретацией сохранившихся следов.**

Если один важный источник не найден, пробел будет заполнен наиболее логичной гипотезой. Если двое старых сотрудников по-разному помнят причину решения, аналитик выберет ту версию, которая лучше соответствует текущему устройству системы. Если фактическая работа пользователей давно отклонилась от первоначального регламента, современный маршрут легко принять за исходный замысел. **В результате новое поколение команды получает не просто знания предшественников, а частично реконструированную версию этих знаний.**

Через несколько циклов такая реконструкция способна стать устойчивой корпоративной легендой. Система работает, объяснение выглядит логичным, документы существуют, но историческая связь между первоначальной причиной и действующим механизмом уже потеряна. **До первого серьёзного изменения это может вообще никого не беспокоить.**

Именно поэтому особенно рискованны старые кастомизированные ERP, пережившие несколько команд внедрения и сопровождения. С каждым новым поколением специалистов стоимость восстановления происхождения решений увеличивается.

Маленький реквизит способен держать на себе большой кусок методологии

Эта проблема хорошо видна не на огромных подсистемах, а на мелких элементах интерфейса. Допустим, в одном из документов существует дополнительное поле «Направление деятельности». **Пользователи давно привыкли его заполнять, поэтому новый аналитик воспринимает реквизит как обычную техническую особенность корпоративной конфигурации.**

Затем выясняется, что значение переносится в другие документы. Потом обнаруживается участие в распределении затрат, а ещё позже — зависимость управленческого отчёта. Оказывается, несколько лет назад предприятие приняло методологическое решение разделять экономический результат по направлениям, после чего потребовалось определить первую точку появления аналитики и обеспечить её дальнейшее прохождение через систему.

В интерфейсе от всей этой истории остался один реквизит.

Если удалить его как «ненужную старую доработку», техническая ошибка может проявиться не в документе, а гораздо позже — например, при закрытии периода или формировании отчётности. Именно поэтому вопрос «где используется поле?» слабее вопроса «**какое решение предприятия это поле материализует?**».

Пока опытный аналитик помнит ответ, такая архитектура кажется очевидной. **После его ухода остаётся техническая поверхность системы, а смысл приходится восстанавливать снизу вверх.**

Цифровой мастер и цифровая тень постепенно расходятся

В книге **«Цифровая трансформация в интеллектуальное предприятие»** полезно различаются ещё два понятия. **Цифровой мастер** описывает принятую модель нормы — как деятельность и поддерживающая её система должны быть устроены. **Цифровая тень** отражает фактическое прохождение деятельности: реальные документы, статусы, движения, действия пользователей, логи, изменения и исключения.

После внедрения эти два слоя неизбежно начинают взаимодействовать. Предприятие развивается, меняются требования, появляются новые сценарии, обновляется типовая конфигурация, пользователи обнаруживают исключения. **Всё это нормально, пока существенные изменения фактической системы возвращаются в принятую модель и образуют следующую понятную версию нормы.**

Потеря контроля начинается тогда, когда этот возврат перестаёт происходить. Разработчик добавил условие, пользователи начали по нему работать, перенос прошёл успешно, и фактическая цифровая тень уже стала другой. **Но если причина решения, новое правило и его зависимости не вернулись в цифровой мастер, объясняющая модель осталась в прошлом.** Одно такое изменение почти безобидно; сотни подобных изменений постепенно

создают систему, которая значительно богаче собственной документации и значительно хуже собственной объяснимости.

В какой-то момент предприятие начинает жить одновременно с двумя версиями ERP: **той, которая действительно работает, и той, которую оно думает, что имеет.**

Уход аналитика лишь делает невидимую проблему заметной

Поэтому было бы ошибкой свести проблему к текучести кадров. Даже если все специалисты остаются в компании, цифровая память постепенно разрушается, когда изменения не включаются в общую модель. Аналитик переходит на другой проект, разработчик переключается на новую подсистему, руководитель забывает подробности старого компромисса, а через несколько лет человеческая память перестаёт быть надёжным механизмом трассировки.

Дополнительная документация уменьшает риск, но не меняет принципиальной конструкции. Более подробное ТЗ полезно, хорошая структура репозитория необходима, записи совещаний могут оказаться бесценными, однако предприятие всё равно должно понимать, какие фрагменты этих материалов связаны между собой и какую действующую норму они объясняют.

Поэтому уход специалиста правильнее рассматривать как стресс-тест. Пока человек находится рядом, он способен вручную закрывать разрывы корпоративной памяти. После его ухода становится видно, какие связи действительно принадлежат организации, а какие всё это время существовали только в голове одного участника проекта.

Как быстро проверить свою систему

Для проверки не требуется проводить большое обследование. Достаточно выбрать одну важную доработку, которая живёт в ERP хотя бы два-три года и уже успела стать привычной частью работы. **Желательно брать не самый крупный проект, потому что по большим изменениям документация обычно сохраняется лучше; гораздо показательнее обычный реквизит, проверка, алгоритм или расширение.**

Сначала нужно установить, из какой реальной потребности предприятия возникло изменение и в каком месте деятельности эта потребность проявлялась. **Затем попробовать восстановить, какое требование было принято, почему выбрали именно такую реализацию, какая версия была проверена и где механизм фактически действует сейчас.** Если ответы последовательно находятся без звонков прежнему аналитику или подрядчику, в этом участке системы причинная память сохранилась.

Если же уже на втором или третьем вопросе появляются фразы «это делал Сергей», «нужно поискать старую конференцию» или «наверное, это было связано с отчётностью», проблема становится видимой. **Причём она относится уже не к качеству конкретной доработки, а к способу, которым предприятие сохраняет историю развития своей ERP.**

Сложная система должна учиться дольше, чем работает любой отдельный сотрудник

Люди будут увольняться, переходить между проектами, менять роли и подрядчиков. В этом нет ничего необычного, и невозможно строить корпоративную архитектуру на предположении, что ключевые специалисты останутся рядом навсегда. **Настоящая задача состоит в другом: опыт сильного аналитика должен после каждого проекта увеличивать память предприятия, а не исчезать вместе с его доступом к корпоративной почте.**

Цифровая нить в таком понимании нужна не для контроля сотрудников и не для бюрократизации каждой маленькой правки. Она позволяет значимым изменениям сохранять происхождение: какое место деятельности затронуто, какое решение принято, как оно реализовано, чем подтверждено и как изменялось дальше. **Тогда следующий специалист получает не готовый ответ на любой вопрос, а значительно более ценную вещь — возможность самостоятельно и доказательно этот ответ восстановить.**

Именно в этот момент знание о **ERP начинает принадлежать организации, а не только людям, которым довелось участвовать в её создании.**

Уход аналитика лишь делает проблему видимой: память системы была распределена между документами, кодом и людьми. **Следующий шаг труднее: что происходит, когда знания никуда не исчезли, но перестали складываться в единую картину?** Большой архив ещё не равен пониманию собственной ERP.

Глава 2. Что делать со сложной ERP: знания есть, а системы никто не понимает

У сложной ERP есть одна особенно коварная стадия деградации. Она наступает не тогда, когда в компании совсем нет документации, и не тогда, когда уходит последний сильный аналитик. Намного опаснее ситуация, в которой знаний вроде бы много: есть старые ТЗ, накоплены задачи, сохранены записи совещаний, в Git лежит история изменений, в папках — схемы, протоколы и презентации. На первый взгляд кажется, что система неплохо обеспечена корпоративной памятью. **Но как только возникает серьёзный вопрос о причине конкретного поведения, оказывается, что все эти материалы существуют рядом, а не вместе.**

Именно в этот момент предприятие обнаруживает неприятную правду: **знание о системе у него есть, а понимания системы нет**. Отдельные люди помнят свои фрагменты, отдельные документы описывают свои куски, отдельные задачи фиксируют частные решения. Но собрать из них один доказуемый ответ на вопрос «почему ERP работает именно так?» уже никто не может. Система становится похожей на библиотеку без каталога: книг много, а путь к нужному смыслу каждый раз приходится восстанавливать заново.

Когда архив растёт, а объяснимость падает

Обычно такая проблема рождается не на старте проекта, а через несколько лет после внедрения. Предприятие живёт, меняется, добавляет новые продукты, корректирует управленческую модель, пересматривает аналитику, перестраивает маршруты согласования, обновляет типовую конфигурацию, внедряет расширения, подключает интеграции. Каждое из этих изменений оставляет след: где-то появляется задача, где-то — переписка, где-то — код, где-то — запись встречи. По отдельности это полезные артефакты. Вместе они создают ощущение, будто система хорошо документирована.

Проблема в том, что наличие следов ещё не означает наличия связанной памяти. Если предприятие не поддерживает трассируемую цепочку между причиной изменения, принятым решением, технической реализацией, проверкой и фактическим применением, архив постепенно превращается в склад. В нём действительно можно что-то найти, но найденное далеко не всегда отвечает на тот вопрос, с которого начинается реальная работа аналитика или руководителя.

Типичная ситуация выглядит так. Нужно изменить старый механизм согласования документа, потому что бизнес-процесс изменился. У команды есть исходное ТЗ, есть несколько задач по доработке, есть коммиты разработчика, есть протокол совещания, в котором эту тему обсуждали, и есть сотрудники, которые помнят, что «там была непростая история». Однако уже через час разбора становится понятно, что никто не может уверенно сказать, какая именно бизнес-причина породила текущее правило, какую экономическую логику оно защищает, на какие параметры и реквизиты опирается, **где ещё используется и каким тестом или практикой подтверждалось.**

Документы отвечают на свои вопросы, но не на вопрос системы целиком

Это важный момент, который предприятия часто недооценивают. Каждый артефакт обычно честно решает свою локальную задачу. ТЗ фиксирует требования на момент написания. Задача отражает единицу работы в рамках конкретного процесса управления изменением. Коммит показывает, что именно делал разработчик. Протокол встречи сохраняет ход обсуждения. Запись совещания позволяет вспомнить контекст. Всё это полезно и необходимо. **Но ни один из этих объектов сам по себе не обязан объяснять систему целиком.**

Поэтому проблема не в том, что документов мало или они бесполезны. Проблема в том, что корпоративная память не складывается сама по себе из простого накопления материалов. Нужна связывающая логика, которая отвечает не на вопрос «что у нас есть?», а на

вопрос «каким путём это решение попало в систему и где теперь живёт его смысл?». Пока такой логики нет, предприятие хранит множество правильных фрагментов и одновременно теряет способность доказательно работать с их совокупностью.

Из-за этого возникает ложное чувство безопасности. Руководителю кажется, что база знаний существует. Аналитику кажется, что при необходимости всё можно восстановить. Разработчику кажется, что история в репозитории никуда не денется. **Но как только нужно пройти путь от бизнес-причины до фактически действующего механизма, выясняется, что у каждого источника свой язык, свой горизонт времени и свой контекст.** Один документ говорит о процессе, другой — о реквизите, третий — о техническом исправлении, четвёртый — о проблеме в отчёте. А вот доказательной нити, связывающей их в один маршрут, нет.

Система непонимаема не потому, что в ней мало знаний, а потому, что знания лежат раздельно

В такой конфигурации предприятие почти неизбежно начинает реконструировать смысл заново при каждом серьёзном изменении. Один аналитик поднимает старые задачи и делает вывод по формулировкам. Другой опирается на привычки пользователей. Третий исходит из текущей логики конфигурации и считает, что именно она и была исходным замыслом. **В результате организация получает не единое знание, а несколько конкурирующих версий объяснения одной и той же системы.**

Особенно заметно это на старых механизмах, которые давно вошли в повседневность. Пока они не требуют изменений, разрыв может оставаться невидимым. **Но стоит затронуть такой механизм — и сразу выясняется, что люди помнят разные причины, документы относятся к разным этапам жизни решения, а техническая реализация успела пережить несколько модификаций.** Формально знания существуют. Практически предприятие вынуждено заново собирать систему как расследование.

Отсюда и растёт известный страх прикосновения к старым доработкам. Команда боится не только сломать код. **Она боится разрушить не до конца понятую причинную архитектуру, потому что не уверена, какие куски бизнес-смысла, экономической логики, параметров и связей держатся на этом, казалось бы, небольшом фрагменте системы.**

ERP невозможно понять на одном уровне

Дополнительная сложность в том, что сложная ERP никогда не живёт в одном измерении. **Её нельзя объяснить только кодом, только регламентом или только задачами проекта.** Чтобы реально понять поведение системы, предприятие должно связывать как минимум четыре слоя, каждый из которых отвечает на свой класс вопросов.

Первый слой — **предметно-процессный**. Он отвечает за то, что именно делает предприятие: какой бизнес-предмет затронут, в каком процессе возникает действие, по какой процедуре оно выполняется и на какой операции появляется развилка. Второй слой — **экономический и учётный**. Здесь проявляется смысл хозяйственного факта: что именно признаётся, как оценивается, по каким аналитикам проходит, как влияет на распределения и отчётность. Третий слой — **нормативно-параметрический**. В нём живут правила обязательности, статусы, НСИ, настройки, аналитики, точки ввода и точки использования. Четвёртый слой — **прикладная архитектура ERP**: объекты метаданных, формы, реквизиты, расширения, регистры, отчёты, интеграции и конкретная техническая реализация.

Пока эти четыре слоя не связываются между собой, предприятие знает систему частями, но не понимает её как единый механизм. Можно хорошо разбираться в объекте метаданных и не понимать, какую норму деятельности он материализует. Можно знать процесс и не видеть, в каком реквизите он впервые входит в систему. Можно понимать отчёт и не уметь доказать, каким путём показатель доходит до него из первичной точки ввода. **Именно поэтому разго-**

воры о «хорошей документации» без вопроса о межслойной связности часто оказываются самоуспокоением.

Почему сильные специалисты пока ещё спасают ситуацию

На практике эту разорванность долго маскируют люди. **Сильный аналитик помнит, что определённое поле появилось не из технической прихоти, а как точка фиксации важной аналитики.** Опытный методолог держит в голове, почему конкретный статус влияет на управленческий смысл операции. Разработчик знает, в каком расширении зашито условие, о котором в документах написано слишком общо. Руководитель проекта помнит, при каких обстоятельствах был принят компромисс, который потом превратился в «само собой разумеющееся правило».

Пока такие люди рядом, организация может не замечать, что корпоративная память в значительной степени существует через человеческие головы, а не через собственную объясняющую модель. Но как только нагрузка растёт, проекты множатся, команды меняются, а система проживает новые циклы доработок, этот ручной способ склейки перестаёт справляться. **Тогда и становится заметно, что знания физически есть, а системы как связанного объекта понимания у предприятия нет.**

Именно поэтому вопрос не сводится к персональным рискам. Уход ключевого специалиста только делает уже существующую проблему более наглядной. Настоящая причина глубже: предприятие не превратило накопленные знания в форму, которая переживает отдельного участника проекта и позволяет следующему человеку доказательно восстановить смысл решения.

Что на самом деле нужно сохранять

Из этого следует неприятный, но очень полезный вывод. Предприятию нужно сохранять не просто документы, а **происхождение решений**. Иными словами, важен не только артефакт, но и доказуемый маршрут его появления: какая проблема возникла, в каком месте деятельности она проявилась, какое решение было принято, как оно было переведено в экономическую и учётную логику, какие параметры и аналитики затронуло, где реализовано в ERP, чем проверено и как вошло в реальную эксплуатацию.

Такую связывающую историю удобно понимать как **цифровую нить**. Она не заменяет ТЗ, не отменяет задачи, не подменяет код и не делает ненужными протоколы. Наоборот, она позволяет всем этим материалам работать вместе. Пока цифровая нить цела, предприятие может перейти от любого элемента — документа, реквизита, алгоритма, отчётной цифры, теста или задачи — к его происхождению и смыслу. Когда нить рвётся, артефакты продолжают существовать, но превращаются в изолированные свидетельства без гарантированной связности.

Это особенно важно для тех организаций, которые уже подходят к идее **цифрового двойника деятельности**. Такой двойник невозможен без связки между принятой нормой, фактической жизнью системы и историей изменений. Иначе предприятие получает лишь набор моделей и следов, но не управляемый контур, в котором можно доказательно отвечать на вопросы о собственном цифровом устройстве.

Проверка на один практический вопрос

Самый простой способ проверить качество своей корпоративной памяти — взять одну старую, но всё ещё важную доработку и попробовать пройти её целиком. Не на уровне «вот тут расширение» и не на уровне «это когда-то просили финансисты», а по полной цепочке: какая реальная потребность предприятия породила изменение, какое решение было принято, где оно закрепилось в процессе, какой экономический смысл защищает, через какие параметры и реквизиты проходит, как реализовано в ERP, **чем было подтверждено и в каких участках системы живёт сейчас.**

Если этот маршрут восстанавливается спокойно и доказательно, значит в данном узле предприятие действительно управляет своей цифровой памятью. Если же уже на втором или

третьем шаге всё упирается в фразы «надо найти старую запись», «кажется, это обсуждали с прежним подрядчиком» или «пользователи давно так работают, значит, наверное, это правильно», проблема налицо. Причём это уже не проблема одного документа и не проблема одного человека. **Это проблема архитектуры знания о собственной ERP.**

Сложная ERP должна быть понятна не только тем, кто её строил

В этом и состоит главный разворот мышления. **Задача зрелого предприятия — не просто накопить следы своей проектной истории, а сделать так, чтобы система оставалась объяснимой для следующего поколения команды.** Хорошая корпоративная память начинается там, где новый аналитик, архитектор или руководитель может не верить на слово, а пройти по доказуемой цепочке и самостоятельно восстановить, зачем существует конкретное поведение системы.

Когда это получается, знание о ERP перестаёт быть набором отдельных воспоминаний, папок и технических находок. Оно становится собственностью организации. **А это уже совсем другой уровень устойчивости: предприятие может менять людей, подрядчиков, версии и подходы, не превращая каждое существенное изменение в цифровую археологию.**

Связанная память нужна не только для поиска причин. У зрелого предприятия почти неизбежно расходятся официальная норма, текущая настройка и живая практика. **Если эти версии нельзя сопоставить и объяснить, вопрос уже не в том, сколько знаний сохранено, а в том, какую версию самого предприятия считать действующей.**

Глава 3. Что делать со сложной ERP: какая версия предприятия настоящая

У сложной ERP есть одна болезненная ситуация, которую предприятие редко замечает сразу. Она выглядит почти безобидно, пока речь не заходит о реальном изменении процесса, о споре между подразделениями или о попытке навести порядок. **В этот момент выясняется, что у компании существует не одна версия собственной деятельности, а сразу несколько.** В регламенте написано одно, в системе настроено другое, а сотрудники на практике уже давно работают третьим способом. И каждая из этих версий на каком-то уровне считается «настоящей».

Именно поэтому вопрос «какая версия предприятия настоящая?» вовсе не философский. Это очень прикладной управленческий вопрос. Он возникает всякий раз, когда нужно понять, как реально устроен процесс, где проходит граница нормы, на что опираться при изменении системы и какую логику вообще защищает предприятие. Пока ответа нет, организация живёт в режиме скрытого расслоения: документы описывают одну реальность, ERP фиксирует другую, а фактическая деятельность движется по третьей траектории.

Обычно это становится заметно не на спокойном участке, а в момент напряжения. Например, приходит новый аналитик, чтобы разобраться в согласовании закупки, ремонте, вводе аналитики, формировании себестоимости или отражении закрытия периода. Ему показывают утверждённый регламент, и там один маршрут. Он открывает ERP, а в ней статусы, роли, реквизиты и последовательность действий устроены иначе. Потом он идёт к пользователям и слышит: «По системе, конечно, так, но реально мы давно делаем по-другому, иначе невозможно работать». **В этот момент он сталкивается не просто с неточной документацией.** Он сталкивается с тремя версиями предприятия, каждая из которых живёт своей жизнью.

Когда предприятие начинает жить сразу в нескольких реальностях

Самая частая ошибка в такой ситуации — считать, что проблема сводится к устаревшему регламенту. На самом деле устаревший регламент — лишь один из видимых симптомов. Гораздо глубже лежит расхождение между тем, **как деятельность была задумана**, **как она описана**, **как она реализована в ERP** и **как она фактически исполняется людьми**. Когда эти слои расходятся, предприятие теряет единую точку опоры.

Это расхождение далеко не всегда рождается из чьей-то халатности. Наоборот, чаще всего оно вырастает из вполне разумных действий. Бизнес меняется быстрее, чем успевают переписать регламент. Команда дорабатывает систему локально, потому что нужно срочно решить практическую проблему. Пользователи находят обходной путь, потому что иначе процесс начинает тормозить. Руководитель временно разрешает исключение, чтобы не сорвать сроки. Через некоторое время все эти временные решения оседают в организации как новая повседневность. **Формально никто не объявлял новую модель работы, но фактически она уже существует.**

Именно так и возникает многослойная ситуация, в которой предприятие перестаёт понимать, какая из версий является действующей. Регламент говорит о правильном процессе. Система удерживает прошлую или частично обновлённую логику. Пользовательская практика отражает адаптацию к реальным ограничениям. **Проблема не в том, что одна из сторон обязательно права, а две другие обязательно ошибаются.** Проблема в том, что между ними теряется объяснимая связь.

«Три версии одного предприятия»

Настоящая проблема — не разница версий, а потеря управляемой нормы

Важно понять одну вещь. Разница между описанной и фактической деятельностью существует почти у любого живого предприятия. Вопрос не в том, чтобы добиться стерильного сов-

падения всех описаний. Вопрос в том, чтобы организация **видела это расхождение, понимала его происхождение и умела принимать по нему решения**. Там, где эта способность есть, расхождение становится объектом управления. Там, где её нет, расхождение превращается в скрытую зону риска.

Когда предприятие перестаёт различать норму и факт, начинается управленческая путаница. Руководитель считает, что процесс контролируется, потому что есть утверждённый порядок. ИТ-команда считает, что процесс контролируется, потому что есть работающий механизм в ERP. Пользователи считают, что процесс контролируется, потому что они научились доводить дело до результата в реальной практике. **Но все трое могут говорить о разных версиях одной и той же деятельности.** Формально разговор идёт об одном процессе, а фактически — о трёх разных объектах.

Из-за этого предприятие начинает терять качество решений. Если вносить изменения по регламенту, можно разрушить реальную практику. Если менять систему по пользовательским привычкам, можно закрепить обходные решения как новую норму, даже если они противоречат исходной логике. Если ориентироваться только на то, что уже есть в ERP, можно невольно принять прошлую доработку за стратегическое решение предприятия. **В итоге любая серьёзная корректировка превращается не в управляемое развитие, а в спор о том, какая реальность имеет право считаться основной.**

Цифровой мастер и цифровая тень: две версии, которые нельзя путать

Чтобы навести здесь порядок, полезно ввести простую, но очень сильную рамку. У предприятия есть **цифровой мастер** — утверждённая модель того, как деятельность должна быть устроена. И у него есть **цифровая тень** — след фактической жизни процесса: реальные действия людей, последовательности, обходные маршруты, точки ввода данных, правила, исключения и последствия этих исключений. Ошибка начинается тогда, когда цифровой мастер и цифровую тень либо смешивают, либо вообще не различают.

Цифровой мастер нужен не для красоты и не для отчётности. Это принятая норма, управленческое основание для изменений, обучения, контроля и развития. Он отвечает на вопрос, какую деятельность предприятие считает правильной и почему. Цифровая тень нужна не меньше. **Она показывает, как система и люди реально проживают эту норму в повседневной жизни.** Она обнаруживает места сопротивления, перегрузки, ручные обходы, неучтённые исключения, скрытые дубли и локальные компромиссы.

Проблема начинается не потому, что цифровая тень отличается от цифрового мастера. В живом предприятии это почти неизбежно. Проблема начинается тогда, когда это отличие становится невидимым, необъяснимым и неуправляемым. **Тогда компания начинает жить в раздвоении.** На уровне слов она опирается на норму. На уровне системы — на накопившуюся конфигурацию. На уровне практики — на адаптацию людей к неудобствам и ограничениям. И в какой-то момент уже никто не может уверенно сказать, где находится исходная управленческая модель.

Цифровой мастер и цифровая тень

Почему расхождение постепенно становится нормой

Самая опасная особенность этой ситуации в том, что она развивается медленно и почти незаметно. Сначала появляется одно исключение. Потом временная доработка. Затем пользовательский обход. После этого новая аналитика, новый статус, отдельная таблица, ручная сверка, временное согласование в обход маршрута, дополнительная роль, неформальный контроль в Excel, звонок вместо системного этапа, письмо вместо зафиксированного решения. Каждая из этих мер в отдельности кажется частной и даже оправданной. **Но вместе они постепенно формируют новую фактическую архитектуру процесса.**

При этом официальная картина обычно не успевает за изменениями. Предметно-процессная логика в регламенте остаётся старой. Экономико-учётная трактовка частично живёт по

новым правилам. Нормативно-параметрическая архитектура в настройках и аналитиках начинает отражать смесь старого и нового. Прикладная архитектура ERP удерживает следы решений разных периодов. **В результате предприятие уже давно живёт в обновлённой практике, но не признало её как новую норму и не пересобрало её как целостную модель.**

Отсюда рождается особый вид организационной слепоты. Компания уверена, что у неё есть «правильный процесс», хотя фактически она опирается на совокупность локальных приспособлений. Чем дольше такая ситуация тянется, тем труднее отличить временное решение от принятой логики. **Со временем обходы начинают казаться естественной частью системы, а сама система — естественным отражением бизнеса, хотя на деле между ними уже накопилось множество неразрешённых разрывов.**

Почему это особенно опасно в ERP

Сложная ERP усиливает эту проблему, потому что в ней всё связано со всем гораздо сильнее, чем кажется на поверхности. Изменение маршрута согласования — это не только шаг процесса. Это роли, статусы, реквизиты, точки ввода, контроль обязательности, аналитики, экономический смысл хозяйственного события, возможные последствия для отчётности и поведения пользователей. **Поэтому различие между «как задумано», «как настроено» и «как делают» здесь никогда не бывает чисто описательной проблемой.** Оно почти всегда влияет на качество управления.

Если предприятие не понимает, какая версия деятельности считается действующей, оно начинает ошибаться сразу в нескольких местах. Обучение новых сотрудников опирается на устаревшую норму. Контроль результатов опирается на неполную картину. Изменения в системе затрагивают не тот слой реальности, который реально удерживает процесс. Пользователи всё меньше доверяют официальным описаниям, потому что они не совпадают с повседневной работой. **А команда сопровождения начинает жить в режиме постоянной реконструкции: что именно здесь считается правильным и кем это было решено.**

В этот момент ERP перестаёт быть просто системой автоматизации и становится полем конкурирующих интерпретаций. Одни участники считают, что настоящая версия предприятия зафиксирована в документах. Другие — что она зашита в текущую настройку. **Третьи — что она проявляется только в живой практике.** Без общей рамки эти позиции не складываются в управление. Они превращаются в источник постоянных конфликтов и дорогостоящих недопониманий.

Как понять, какая версия должна считаться настоящей

Главное правило здесь довольно жёсткое: настоящей для управления не может считаться просто та версия, которая первой попалась под руку. Нельзя автоматически признать настоящим ни текст регламента, ни текущую конфигурацию, ни пользовательскую привычку. Настоящей может считаться только та версия, по которой предприятие **осознанно приняло управленческое решение** и может объяснить её на всех ключевых слоях.

Это означает, что организация должна пройти не через спор мнений, а через процедуру выяснения. Сначала нужно увидеть расхождение. Затем понять, где именно оно находится: в предметно-процессной логике, в экономико-учётной модели, в параметрах и правилах или в прикладной реализации ERP. После этого необходимо ответить на несколько неприятных, но обязательных вопросов. Что из текущей практики является вынужденным обходом, а что стало новой нормой? Какая часть отклонений возникла из-за изменений бизнеса, а какая — из-за некачественной настройки или неудачного решения? Что предприятие хочет сохранить, а что собирается вернуть к исходной модели?

Только после этого появляется право говорить о «настоящей» версии. Причём очень важно, что итогом здесь должно стать не словесное согласие на совещании, а обновлённая управляемая модель. Если фактическая практика признана правильной, значит, её нужно перевести в статус новой нормы: пересобрать описание, параметры, точки контроля, логику

системы и обучение. Если фактическая практика признана ошибочной или временной, значит, предприятие должно не просто критиковать её, а возвращать деятельность к утверждённой норме и убирать причины отклонений.

Что значит навести порядок по-взрослому

Зрелый подход к этой проблеме состоит не в поиске виноватых и не в косметическом обновлении документов. Он состоит в восстановлении связности между нормой, системой и фактом. По сути, предприятие должно собрать объяснимый контур: утверждённая модель деятельности, цифровая тень фактического исполнения, выявленные расхождения, причины этих расхождений, решения по ним и изменения, которые возвращают организацию в управляемое состояние.

Здесь особенно важна цифровая нить. **Именно она позволяет не просто увидеть три версии предприятия, а связать их между собой в одну объяснимую историю.** Почему эта норма была принята? Где она материализована в ERP? Где именно фактическая практика от неё отошла? Чем вызвано это отклонение? Какое решение по нему было принято? Что после этого должно измениться в регламенте, настройках, ролях, маршрутах, аналитиках и обучении? Пока такой нити нет, предприятие будет снова и снова натыкаться на одну и ту же проблему под разными названиями.

Важный эффект такого подхода в том, что организация перестаёт спорить о версиях и начинает управлять расхождениями. Она больше не вынуждена выбирать между «бумагой», «системой» и «живой практикой» как между тремя взаимоисключающими истинами. Вместо этого она выстраивает управляемое отношение между ними. Именно так и появляется цифровой двойник деятельности — не как красивая концепция, а как рабочая способность предприятия видеть свою норму, **наблюдать её фактическое исполнение и осмысленно развивать обе стороны этой связи.**

Как предприятие возвращает единую версию реальности

Простой практический тест для руководителя и аналитика

Проверить наличие проблемы можно довольно просто. Возьмите один важный процесс и попросите три разных источника ответить, как он устроен. Первый источник — утверждённый регламент или описание. Второй — текущая логика ERP: статусы, маршруты, обязательные поля, роли, правила, аналитики и фактические ограничения системы. Третий — живая практика пользователей, которые каждый день проводят этот процесс. **Если ответы совпадают только частично, а различия никто не умеет объяснить и классифицировать, значит предприятие уже живёт сразу в нескольких версиях себя.**

Ещё важнее следующий вопрос. Если завтра понадобится изменить этот процесс, на что именно будет опираться команда? На текст регламента, на текущую настройку, на привычки пользователей или на чьё-то устное знание? **Если в организации нет единого ответа, значит проблема уже вышла за пределы удобства и стала риском для управления изменениями.** В такой ситуации любое развитие ERP превращается в работу по зыбкому основанию.

Как только расхождения становятся видимыми, появляется практическая развилка: что в системе является типовой логикой, что — собственным решением предприятия, а что вообще возникло как временный компромисс. **Граница здесь проходит не по коду, а по происхождению решений.** Поэтому следующий шаг — научиться восстанавливать эту границу.

Часть II. Почему технические объекты теряют бизнес-смысл

Теперь мы спускаемся с уровня общей памяти к конкретным объектам ERP. Поле, показатель, фрагмент кода или старая доработка технически существуют вполне определённо, однако управлять ими безопасно можно только тогда, когда понятны их происхождение, функция и дальнейшие зависимости. **Во второй части вопрос меняется с «что мы помним о системе?» на «можем ли мы доказательно вернуть техническому объекту его бизнес-адрес?».**

Глава 4. Что делать со сложной ERP: где заканчивается типовая и начинается своя

У старой кастомизированной ERP есть очень характерный момент, когда технически простая задача неожиданно превращается в исследовательскую. Предприятие готовится к обновлению, пересматривает накопившиеся расширения или просто хочет избавиться от старой доработки, которую уже много лет никто не трогал. **Кто-нибудь замечает, что в новой типовой версии появилась похожая возможность, и возникает вполне разумная мысль: зачем продолжать сопровождать собственный механизм, если теперь примерно то же самое умеет сама ERP?**

На поверхности вопрос действительно выглядит техническим. Можно сравнить конфигурации, посмотреть изменённые объекты, найти расширение, изучить реквизиты и код. **Через некоторое время становится понятно, где именно находится собственная реализация и чем она отличается от текущей типовой версии.** Казалось бы, после этого остаётся решить, что удалить, что оставить и что заменить типовым функционалом.

Именно здесь начинается главная ошибка. **Техническое отличие показывает, где система изменилась, но почти ничего не говорит о том, зачем предприятие когда-то её изменило.** Если эта причинная связь за годы эксплуатации потерялась, команда знает местоположение доработки, но ещё не знает её границу как бизнес-решения. А без этого совершенно похожие функции легко принять за одинаковые, хотя на самом деле они могут удерживать разные процессы, разный экономический смысл и разные зависимости.

Кажется, что типовую и собственную ERP можно разделить одной линией

В начале проекта граница обычно выглядит довольно просто. Есть поставляемая конфигурация ERP, есть требования предприятия, а всё, что пришлось изменить или добавить ради этих требований, воспринимается как собственная часть системы. Пока команда внедрения рядом и история решений свежая, такая картина достаточно хорошо работает. **Люди помнят, какие ограничения типового механизма столкнули проект с необходимостью изменения, какие варианты обсуждались и почему в итоге выбрали именно эту конструкцию.**

Через несколько лет простота исчезает. Типовая ERP развивается, старые ограничения снимаются, появляются новые возможности, меняются формы, алгоритмы и сценарии. Одновременно развивается и предприятие. Собственные механизмы переписываются, часть решений уходит в расширения, часть переносится в настройки и нормативно-справочную информацию, некоторые правила вообще перестают выглядеть как доработки, **потому что технически реализуются штатными средствами системы.**

Поэтому через пять или семь лет вопрос «что у нас типовое?» уже не имеет одного технического ответа. В одном месте собственная логика действительно находится в коде. В другом предприятие использует полностью типовой объект, но настроило его под собственную методологию. В третьем типовой механизм появился позже и теперь внешне повторяет старую проектную функцию. В четвёртом собственная доработка давно удалена, но её смысл сохранился в аналитиках, статусах, параметрах и принятом порядке работы.

В этот момент прямой вертикальной границы между «типовым» и «своим» больше нет.

Собственная ERP значительно больше собственного кода

Это принципиально важно. Когда говорят о кастомизированной ERP, чаще всего внимание автоматически смещается в сторону изменённой конфигурации, расширений и программного кода. Для разработчика это естественный взгляд: именно там технически видна разница между поставкой и конкретной информационной базой. **Но с точки зрения предприятия собственная система начинается гораздо раньше.**

Допустим, организация использует штатный документ ERP без единой программной доработки, но в конкретном процессе утвердила обязательную аналитику, определила особый момент её появления, зафиксировала порядок переноса между документами и построила на ней управленческую отчётность. Технически объект типовой. **Управленчески его использование уже является частью собственной архитектуры предприятия.**

То же относится к статусам, справочникам, правилам заполнения, параметрам, направлениям деятельности, статьям, аналитикам и настройкам учётной модели. Они могут существовать внутри штатных возможностей ERP и при этом материализовывать абсолютно конкретные решения конкретной компании. **Удалять здесь нечего, сравнение конфигураций ничего подозрительного не покажет, но от изменения этой логики предприятие может получить последствия не менее серьёзные, чем от удаления крупного расширения.**

Поэтому выражение «у нас почти типовая ERP» само по себе мало что объясняет. Оно может означать небольшое количество изменённого кода, но ничего не говорит о количестве собственных управленческих решений, которые живут внутри типовых механизмов.

Похожая функция может решать совсем другую задачу

Рассмотрим простой пример. Несколько лет назад предприятие решило разделять определённый экономический результат по направлениям деятельности. Для этого понадобилось не просто добавить аналитику в итоговый отчёт. **Нужно было определить место, где эта аналитика впервые становится известна, добиться её корректного переноса по процессу и сохранить до момента, когда она участвует в распределении и формировании результата.**

На проекте появляется дополнительная логика вокруг реквизита «Направление деятельности». Возможно, где-то поле становится обязательным, где-то значение заполняется автоматически, где-то проверяется допустимость, а где-то переносится дальше. Через несколько лет вся эта конструкция становится привычной, и пользователи уже не воспринимают её как доработку. **Для них это просто нормальное поведение ERP.**

Потом выходит новая версия ERP, в которой типовой механизм тоже активно использует направления деятельности. Название аналитики совпадает. На форме существует похожее поле. Появились штатные механизмы заполнения и использования. **Возникает ощущение, что проблема решена производителем конфигурации и собственный механизм можно убрать.**

Но совпадение названия ничего не доказывает. Типовой механизм может определять аналитику позже. Он может использовать её в другом хозяйственном контексте. Она может влиять на другой набор регистров и показателей. **Она может не поддерживать именно ту раннюю точку ввода, из-за которой предприятие когда-то вообще начало доработку.** Даже если результат в большинстве обычных сценариев выглядит одинаково, один важный сценарий способен принципиально отличаться.

Поэтому сильный аналитик в такой ситуации задаёт не вопрос «есть ли теперь это поле в типовой ERP?». Его интересует другое: **какую задачу предприятия решала старая конструкция и закрывает ли новая типовая возможность именно эту задачу целиком?**

Это уже совсем другой уровень сравнения.

Сравнивать нужно не объекты, а две причинные цепочки

Старое собственное решение имеет историю происхождения. Когда-то существовала проблема или ограничение. На её основе сформулировали требование. **Затем приняли методологическое или процессное решение, определили экономический смысл, выбрали параметры и аналитику, после чего перевели всё это в техническую реализацию ERP.** Наконец, конструкцию проверили и применили в реальной работе.

У новой типовой возможности существует своя цепочка. Она тоже что-то решает, опирается на определённые параметры и реализует определённое поведение. **Но её происхождение другое: она создавалась как универсальный механизм продукта, а не как ответ на конкретную историю конкретного предприятия.**

Правильное сравнение начинается тогда, когда две эти цепочки кладут рядом. Не «наш реквизит против типового реквизита», а **наша потребность против возможностей типового механизма**. Не «наш обработчик против нового обработчика», а **наше правило против правила, которое теперь поддерживает стандартная система**. Не «наш отчёт больше не нужен», а **все ли показатели и аналитические зависимости, которые он обеспечивал, теперь воспроизводятся штатным способом**.

Именно здесь можно впервые доказательно сказать, **что типовая система действительно догнала старую собственную реализацию.**

Граница между типовой и собственной ERP проходит зигзагом

Удобнее всего увидеть это, если разложить систему на несколько смысловых слоёв. На предметно-процессном уровне предприятие определяет собственную деятельность: процессы, процедуры, операции, роли и правила выполнения. На экономическом и учётном уровне оно решает, что означают хозяйственные факты, как они оцениваются, по каким аналитикам разделяются и в какие результаты превращаются. На нормативно-параметрическом уровне эти решения получают параметры, статусы, справочники, обязательность и точки ввода. **Только после этого они материализуются в прикладной архитектуре ERP.**

Типовая конфигурация способна покрывать разные части этой конструкции. Она может полностью дать техническую форму и алгоритм, но предприятие задаст собственные параметры. Может предоставить универсальную аналитику, но компания по-своему определит момент её использования. **Иногда типовая система покрывает практически весь маршрут, а собственным остаётся только бизнес-правило применения.** Иногда наоборот: верхняя методология остаётся прежней, а для её реализации требуется существенная техническая доработка.

Поэтому реальная граница никогда не похожа на аккуратную вертикальную линию. Она проходит через все уровни системы и постоянно меняется вместе с развитием конфигурации и предприятия.

Через несколько лет происхождение решений стирается

Если бы каждая доработка сохраняла понятную историю происхождения, проблема была бы существенно проще. Команда могла бы открыть решение пятилетней давности и увидеть, какое ограничение типовой системы тогда существовало, какую потребность сформулировал бизнес, **какие альтернативы рассматривались и почему была выбрана именно эта реализация.** После этого сравнение с современным функционалом превращалось бы в нормальную инженерную работу.

Но обычно история распадается. Исходное ТЗ лежит в одной папке. Несколько последующих изменений — в задачах сопровождения. Одна важная оговорка осталась только в протоколе совещания. Код пережил рефакторинг. Часть механизма перенесли в расширение. Настройки несколько раз менялись непосредственно в информационной базе. **А сотрудники, которые помнили первоначальную причину решения, давно занимаются другими проектами или вообще ушли из компании.**

Техническая реализация остаётся, а её происхождение постепенно становится гипотезой.

Именно поэтому старая кастомизированная ERP так часто вызывает страх. Команда видит механизм, который выглядит избыточным, но не может доказать, что он действительно больше ничего не защищает. Любая попытка его убрать начинает сопровождаться вопросами: а вдруг это используется при закрытии месяца, а вдруг на этом держится интегра-

ция, а вдруг какая-то аналитика попадает отсюда в отчёт, а вдруг существует редкий сценарий, который просто никто сейчас не вспомнил?

Этот страх нельзя убрать призывом «лучше документировать код». **Код способен показать технические зависимости, но он не восстановит сам по себе причину, почему зависимость когда-то считалась необходимой.**

Обновление превращается в археологию не из-за количества доработок

Большое количество собственных изменений, безусловно, усложняет обновление. **Но количество кода — не самая тяжёлая проблема.** Гораздо хуже, когда у кода нет бизнес-адреса.

Если команда знает, что определённое расширение реализует конкретное правило процесса, связано с определённой экономической моделью, использует такие-то аналитики и подтверждено такими-то сценариями, даже сложную доработку можно исследовать достаточно рационально. **Понятно, что с чем сравнивать и какой результат должен сохраниться.**

Но маленькая функция без происхождения способна оказаться опаснее большого модуля. Она выглядит незначительно, её назначение никто не помнит, а где-то далеко по цепочке на ней держится отчётный показатель или важная проверка. **Именно такие места делают обновление похожим на археологию: приходится извлекать фрагменты прошлого и по косвенным признакам угадывать, для чего их когда-то создали.**

Отсюда следует ещё один важный вывод. **Сложность обновления определяется не только количеством отклонений от типовой системы, но и качеством памяти об этих отклонениях.**

Типовая система может полностью догнать старую доработку — и это хороший результат

Вторая крайность тоже вредна. **Иногда команда настолько привыкла считать собственный механизм уникальным, что продолжает сопровождать его даже тогда, когда типовая ERP давно решила ту же задачу лучше и глубже.** Возникает историческая привязанность: «это наше, мы этим пользуемся много лет, значит удалять опасно».

Такой подход превращает ERP в музей прошлых проектных решений. **Каждое обновление становится дороже, собственный код продолжает расходиться с развивающейся типовой конфигурацией, а предприятие тратит ресурсы на сопровождение того, что больше не создаёт уникальной ценности.**

Поэтому задача цифровой нити состоит не в том, чтобы навечно сохранить каждую доработку. Наоборот, хорошая причинная история позволяет **безопасно от неё отказаться**, когда типовая система действительно закрывает исходное требование. Известно, зачем механизм был создан, поэтому можно доказательно проверить, что новая типовая возможность выполняет ту же функцию и сохраняет нужный результат.

В таком случае удаление собственной реализации — **не потеря проектного наследия, а нормальное развитие архитектуры.**

После сравнения возможны три совершенно разных вывода

Первый вариант — современная типовая возможность полностью покрывает исходное требование, включая ключевые сценарии и зависимости. **Тогда собственную реализацию действительно можно выводить из эксплуатации, предварительно подтвердив переход испытаниями и обновив принятую модель системы.**

Второй вариант встречается едва ли не чаще: типовая функциональность покрывает большую часть старой доработки, но не всю. **Тогда вместо сохранения целого собственного механизма имеет смысл оставить только небольшую добавочную логику поверх типового решения.** Архитектура становится проще, а предприятие одновременно сохраняет свои действительно специфические требования.

Третий вариант — внешнее сходство оказалось обманчивым. Типовой механизм решает похожую, но другую задачу или не удерживает критически важную зависимость. **Тогда собственное решение следует сохранить, а главное — наконец нормально зафиксировать его происхождение, чтобы через очередные несколько лет та же дискуссия не началась заново.**

Во всех трёх случаях решение принимается не по внешнему сходству интерфейса и не по количеству совпадающих объектов метаданных. **Оно принимается по покрытию смысла.**

Цифровая нить превращает сравнение в инженерную задачу

Именно здесь становится особенно понятен смысл цифровой нити. Она соединяет не просто версии кода. Она хранит происхождение изменения: какая потребность существовала, какую норму предприятия затронули, какое решение приняли, как перевели его в параметры и техническую реализацию, чем подтвердили и где оно действует сейчас.

С такой историей обновление выглядит совершенно иначе. Аналитик видит новую типовую возможность и не начинает с предположения «похоже на наше». Он поднимает исходную цепочку собственного решения и последовательно проверяет покрытие. Что из старой потребности закрывается? Что стало неактуальным из-за изменения самого бизнеса? Что типовая ERP теперь делает штатно? Что всё ещё остаётся специфическим? **Какие сценарии нужно подтвердить до переключения?**

В этом режиме предприятие перестаёт бояться собственной системы. **Оно может менять её, потому что понимает не только текущее устройство, но и происхождение важных решений.**

Где же всё-таки заканчивается типовая и начинается своя

После всего сказанного ответ становится несколько неожиданным. Типовая ERP заканчивается не в точке первого изменённого программного модуля. **И собственная ERP начинается не там, где разработчик написал первую строку корпоративного кода.**

Граница определяется тем, какие решения предприятие приняло само и какие части своей деятельности **оно материализовало с помощью системы.**

Иногда такое решение потребует собственного расширения. Иногда — особой настройки типового механизма. Иногда — собственного состава НСИ или аналитик. Иногда весь технический маршрут окажется стандартным, а уникальным будет только правило, определяющее момент и условия его применения. **Но во всех этих случаях существует собственная архитектура предприятия, потому что существует собственная норма деятельности.**

Поэтому зрелая организация должна хранить не только список доработок, но и **карту собственных решений**. Тогда можно в любой момент увидеть, какие из них всё ещё требуют собственной технической реализации, какие уже поддерживаются типовой системой и какие вообще перестали быть актуальными.

Сравнение конфигураций отвечает только на половину вопроса

Сравнение конфигураций остаётся необходимым техническим инструментом. **Оно показывает, какие объекты отличаются, где изменился код, какие реквизиты добавлены и какой технический объём нужно исследовать.** Без этого невозможно качественно сопровождать сложную ERP.

Но оно отвечает только на вопрос «чем наша система отличается от типовой сегодня?». Для управления развитием предприятия нужен ещё один ответ: **«почему она вообще стала другой?»**

Вот этот второй вопрос уже невозможно решить одним техническим сравнением. Для него требуется восстановить цифровую нить между потребностью предприятия, принятым решением и текущей реализацией. И чем старше система, тем важнее эта связь.

Именно поэтому настоящая граница между типовой и собственной ERP — это не граница кода. **Это граница происхождения решений.**

Восстанавливать происхождение решений удобнее всего с маленьких технических точек, которые кажутся очевидными. Одна из лучших — обязательный реквизит. **Красная звёздочка показывает, что система чего-то требует, но не объясняет, кто и зачем сформулировал это требование.**

Глава 5. Что делать со сложной ERP: почему это поле обязательно

В сложной ERP есть вопросы, которые внешне выглядят почти смешными. Пользователь открывает документ, доходит до очередного реквизита со звёздочкой и спрашивает аналитика: «А зачем мне это заполнять?» Аналитик смотрит на форму и видит очевидный технический факт: без значения система не позволяет продолжить работу. **Самый простой ответ поэтому звучит вполне убедительно — поле обязательное, так настроено, иначе документ не проводится.**

Проблема начинается, когда пользователь задаёт следующий вопрос: **«Хорошо. А почему оно обязательное?»**

На него интерфейс уже не отвечает. Красная звёздочка показывает наличие требования, но ничего не говорит о его происхождении. Она не объясняет, нужен ли реквизит самой платформе, прикладной логике ERP, экономической модели предприятия, конкретному сценарию работы или просто остался обязательным после решения пятилетней давности, **смысл которого никто больше не помнит.**

Именно поэтому маленькое поле способно оказаться хорошей диагностической точкой всей сложной ERP. Пока организация понимает, зачем значение требуется, кто должен его определить, в какой момент процесса оно становится известным и куда идёт дальше, система остаётся объяснимой. Когда вместо этого появляется ответ «так исторически сложилось», предприятие сталкивается уже не с особенностью интерфейса, а с потерянным фрагментом собственной цифровой архитектуры.

Звёздочка показывает требование, но скрывает причину

Представим обычную ситуацию. В документе существует поле «Направление деятельности». Пользователь оформляет операцию и система требует его заполнить. Для самого пользователя это может выглядеть странно: товар известен, контрагент известен, подразделение указано, сумма рассчитана. **Зачем ещё одно значение, без которого нельзя закончить документ?**

Если спросить нескольких участников процесса, легко получить несколько разных объяснений. Один сотрудник скажет, что поле необходимо бухгалтерии. Другой вспомнит, что без него неправильно формируется какой-то отчёт. Третий уверен, что это обычное требование. **Четвёртый вообще не знает причины, но много лет выбирает одно и то же значение, потому что так показали при обучении.**

При этом все они могут добросовестно работать в системе.

Сам факт обязательности создаёт иллюзию, что причина где-то надёжно определена. Кажется: если не пускает дальше, значит существует объективная необходимость. **Но это не обязательно так.** Система способна очень жёстко контролировать правило, происхождение которого организация давно перестала понимать.

В результате обязательность постепенно превращается в ритуал. Пользователь не вводит информацию потому, что осознанно фиксирует важный факт деятельности. Он вводит её потому, что иначе программа не разрешает перейти к следующему шагу.

Для качественной ERP это принципиально разные ситуации.

Одинаковая звёздочка может означать совершенно разные вещи

Сложность состоит ещё и в том, что внешне разные причины выглядят совершенно одинаково. Пользователь видит один элемент интерфейса: поле нельзя оставить пустым. **Но архитектурно за этим могут стоять принципиально разные классы требований.**

В одном случае значение действительно необходимо самой технической конструкции. Без него невозможно однозначно определить объект операции или корректно сохранить дан-

ные. **Это наиболее простой вариант: обязательность непосредственно вытекает из устройства системы.**

В другом случае платформа технически могла бы сохранить пустое значение, но выбранная хозяйственная операция в ERP предполагает определённую аналитику или объект. Здесь требование создаёт уже прикладная логика. **Система знает выбранный сценарий и поэтому понимает, какие данные необходимы для его корректного выполнения.**

Но значительно интереснее третий случай. ERP могла бы провести операцию без значения, однако предприятие самостоятельно решило, что такой хозяйственный факт считается недостаточно определённым. Например, руководство хочет видеть финансовый результат в разрезе направлений деятельности, поэтому определённая аналитика должна появиться раньше и пройти через всю цепочку документов. Тогда обязательность возникла не в программе. **Она возникла в экономической и управленческой модели предприятия, а программа только материализовала это решение.**

Наконец, существует сценарная обязательность. **Поле может быть совершенно не нужно в большинстве операций и становиться критическим только при определённых условиях.** Для одного вида деятельности аналитика известна сразу, для другого появляется позже. В одном процессе её определяет инициатор, в другом — финансовая служба. Один сценарий требует детализации, а второй может быть корректно выполнен без неё.

На форме эти четыре ситуации способны выглядеть одинаково. **Но для аналитика это четыре разные архитектурные причины и четыре разных способа работы с требованием.**

Почему ответ «так требует ERP» почти всегда недостаточен

Фраза «это требует ERP» удобна тем, что завершает разговор. Она переводит требование из области обсуждаемых решений в область почти физических законов. **Пользователь может быть недоволен, но спорить вроде бы не с кем: программа устроена именно так.**

Для сложной ERP такой ответ опасен.

Если обязательность действительно является типовым прикладным правилом ERP, хороший аналитик способен установить, с каким сценарием оно связано и что именно система не сможет однозначно выполнить без значения. Если же правило появилось на проекте, его необходимо связать с решением предприятия. Если требование зависит от сценария, должна быть понятна граница этого сценария.

Иначе техническое поведение подмеяет объяснение.

Через несколько лет это начинает создавать проблемы уже при изменениях. Новый аналитик видит обязательное поле и принимает существующую реализацию за объективную норму. Никто не задаёт вопрос, актуальна ли ещё исходная причина. Возможно, управленческая модель давно изменилась. Возможно, значение теперь можно получить автоматически. Возможно, типовая ERP научилась определять его иначе. **Возможно, обязательность вообще была поставлена временно, чтобы пережить конкретный этап внедрения.**

Но если причина не сохранилась, звёздочка продолжает жить собственной жизнью.

Четыре причины одной обязательности

Поэтому при исследовании обязательного реквизита полезно сначала определить не то, **где установлен контроль**, а то, **какого класса это контроль**.

Техническая обязательность отвечает на вопрос, может ли система вообще однозначно существовать без значения. Прикладная обязательность показывает, требует ли его конкретный механизм ERP. Методологическая обязательность связана с нормой самого предприятия: значение необходимо для правильной классификации, оценки, распределения или управленческой интерпретации хозяйственного факта. **Сценарная обязательность определяет условия, при которых требование включается.**

Это разделение важно не ради классификации как таковой. Оно определяет, что можно делать с полем дальше.

Техническую обязательность невозможно просто отменить без изменения самой конструкции. **Прикладную нужно сопоставлять с выбранным типовым сценарием.** Методологическую необходимо проверять относительно действующей модели предприятия. Сценарную — правильно привязывать к моменту и условиям возникновения факта.

Если четыре этих причины смешаны в одно неопределённое «поле обязательное», изменение становится опасным. **Никто не понимает, какую именно норму он собирается нарушить.**

Самый важный вопрос — когда значение становится известным

Есть ещё одна проблема, которая часто остаётся незаметной даже там, где причина обязательности в целом понятна. **Предприятие знает, что аналитика необходима для будущего расчёта или отчёта, и делает логичный на первый взгляд вывод: раз значение понадобится потом, лучше потребовать его как можно раньше.**

Так возникают обязательные поля в первых документах процесса.

На схеме всё выглядит аккуратно. Аналитика будет заполнена сразу, дальше автоматически перейдёт по цепочке и нигде не потеряется. Контроль качества данных кажется даже сильнее.

Но есть одно условие: пользователь, которому предъявили обязательность, должен **реально знать правильное значение**.

Если в этот момент оно ещё неизвестно, система создаёт не качество данных, а давление на пользователя. **Операцию нужно выполнить, документ не проводится, а правильного ответа пока физически нет.**

У человека остаётся довольно предсказуемый выход — **выбрать то, что позволит продолжить работу.**

Конец ознакомительного фрагмента.