

Савчик Владимир

Управление конфигурацией программных продуктов

Научно-техническая литература

Москва
2017

УДК XXX.XX.X-XX

ББК XX(XX)X-X

XXX

XXX Савчик В. В.

Управление конфигурацией программных продуктов.

— М.: ООО «XXX», 2017. — XXX с.

ISBN XXX -X-XXXXXX-XX-X

«Управление конфигурацией программных продуктов» — краткое описание книги

ISBN XXX -X-XXXXXX-XX-X

© Савчик В.В., 2017

© XXX, 2017

ОГЛАВЛЕНИЕ

Оглавление.....	3
Введение	5
Программные продукты	17
Операции.....	39
Программный код	41
Сборка программного кода	53
Выпуск релизов.....	67
Создание и поддержка сред	99
Установка изменений.....	129
Мониторинг.....	163
Документация	175
Релизные процессы.....	191
База данных конфигурационного управления.....	211
Заключение	225
Содержание	229

Введение

ПРОБЛЕМА ПОСЛЕДНЕЙ МИЛИ

РЕЛИЗНЫЕ ПРОЦЕССЫ

ПАРАДИГМЫ РЕЛИЗНЫХ ПРОЦЕССОВ

УЧЕТНАЯ СИСТЕМА КАК ИНСТРУМЕНТ

ЕДИНАЯ МЕТАМОДЕЛЬ

ИДЕОЛОГИЯ, МЕТОДОЛОГИЯ И ТЕХНОЛОГИЯ

Целью данной книги и описываемых в ней методов является помочь множеству специалистов по разработке и эксплуатации программных систем в части создания устойчивых и эффективных процессов взаимодействия участников, находящихся в разных подразделениях или разных организациях, для решения задач управления выпуском релизов, управления изменениями и управления средами программных продуктов, которые неизбежно возникают при работе со средними и крупными программными системами.

Благодаря решению этих задач написанный программный код превращается в программный продукт, что позволяет говорить о профессиональном подходе как на организационном, так и на технологическом уровне.

Книга дает определение основных понятий и аспектов деятельности при решении задач управления выпуском релизов, управления изменениями и управления средами

Проблема последней мили

В электротехнике проблемой последней мили называют множество задач, которые встают, когда канал связи уже протянут на немалое расстояние, осталась одна миля до города, и здесь начинаются вопросы логистики - где поставить окончное оборудование, кто его будет обслуживать, как осуществить разводку от этого оборудования по районам и к пользователям, какие технологии при этом используются.

Количество политических, организационных, коммерческих и технических проблем возникает такое, что по времени решения и затратам они приближаются к стоимости всего протянутого до этой точки канала. Аналог этой проблемы находят в разных областях, и в том числе в информационных технологиях, при выпуске программных продуктов.

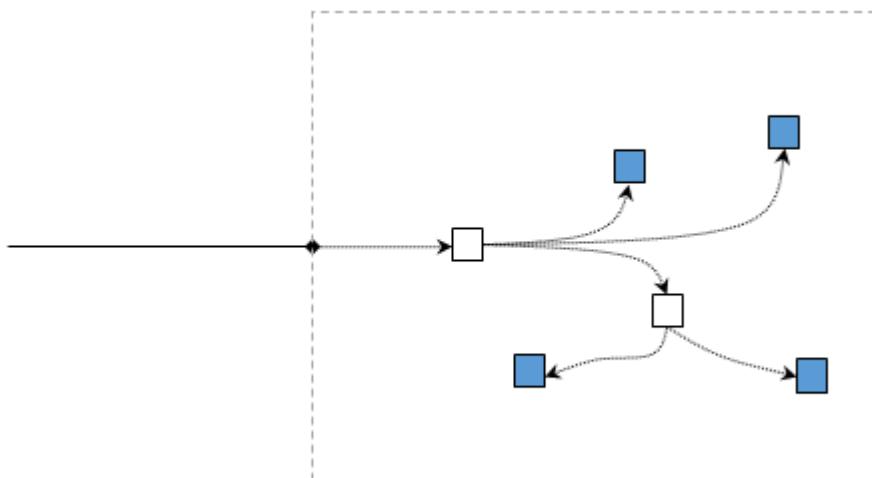


Рис. 1. Последняя миля

Чтобы программный код превратился в продукт, т.е. в что-то понятное, надежное, подходящее для использования, приходится выстроить процессы, использовать соответствующие инструменты, и применять определенную технологию.

Продукт не появляется случайно, создание продукта - это осмысленный, целенаправленный процесс, имеющий свою структуру, принципы и методы. Так или иначе для всех продуктов, которые претендуют на широкую аудиторию или на использование в критически важных операциях, такой процесс приходится выстраивать и лучше иметь представление о нем заранее, чем учиться методом проб и ошибок, изобретая велосипед. Данная книга показывает наиболее существенные проблемы последней мили при создании программного продукта и предлагает методы их решения.

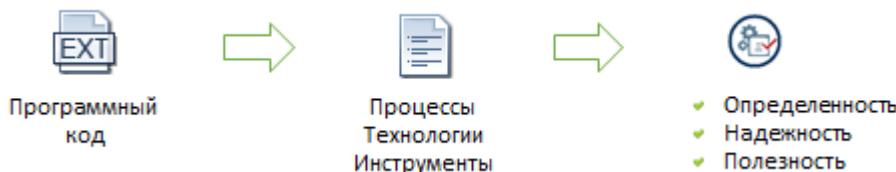


Рис. 2. От программного кода к программному продукту

Релизные процессы

Первый релиз программного продукта неизбежно проходит весьма сумбурно, и будет наивным надеяться, что кому-то удастся этого избежать. Однако за первым следует второй, третий и десятый, и в конце концов, участники событий так или иначе начинают избавляться от связанных с релизом рисков и

вырабатывают некоторый регулярно применяемый подход к выпуску последующих релизов, что и порождает то, что называется релизным процессом.



Рис. 3. На пути к релиезному процессу

Строго говоря, процессом можно называть регулярную деятельность, находящуюся в одних руках с точки зрения управления, что позволяет анализировать и улучшать используемые методы.

Если одна организация отвечает за выпуск релиза, а другая - за его установку и использование, то это не один процесс, а несколько разных процессов. В методологии ITIL это примерно соответствует процессам управления релизами и установкой и управления изменениями, а в стандарте ISO/IEK 12207 - управление выпуском и поставкой как часть управления конфигурацией.

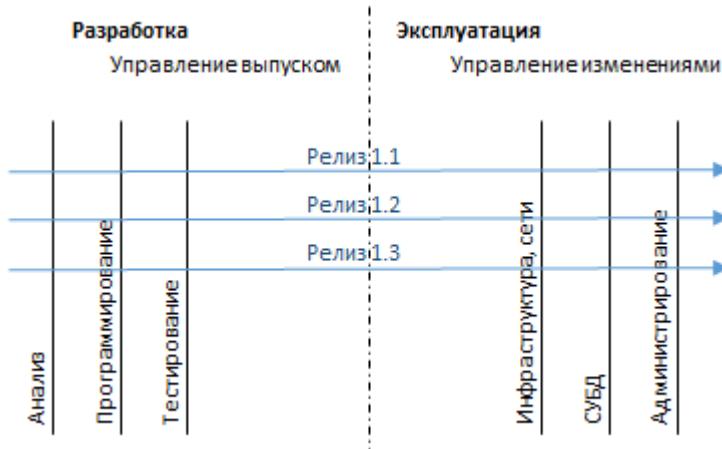


Рис. 4. Релиз в разработке и эксплуатации

С другой стороны, если за среду промышленной эксплуатации отвечает та же организация, что и разрабатывает продукт, то организация может выделить специальную роль релиз-менеджера, чтобы отвечать за весь жизненный путь релиза, включая выпуск и установку.

Это создает ситуацию одного релизного процесса, который включает управление выпуском, но не включает, а лишь взаимодействует с процессом управления изменениями, поскольку кроме релизов, есть другие причины для изменений в среде промышленной эксплуатации. Если продуктов несколько и у них разные релиз-менеджеры, то это разные процессы, хотя возможно и с единой методологией.

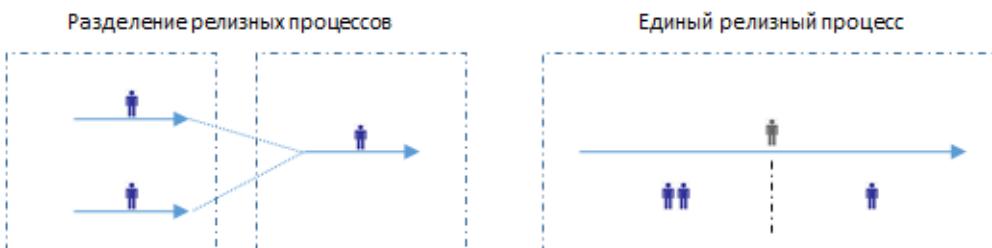


Рис. 5. Состав релизных процессов

Данная книга предлагает общую схему организации релизных процессов и объясняет цели, структуру работ, роли участников и принципы оценки эффективности по каждому виду активности.

Парадигмы релизных процессов

Существует две крайних точки зрения на то, как надо выпускать релизы. Первая требует выпускать релизы редко, после тщательной их подготовки и всестороннего тестирования, а вторая призывает быть ближе к пользователям и заказчику, рынку, быстрее удовлетворять их запросы, делая релизы более частыми и мелкими, а иногда и более рискованными.

Сторонники быстрых релизов, которые ассоциируются с методологиями типа Agile (XP, Scrum и т.п.) не планируют и не выпускают больших релизов, а сторонники строгих каскадных моделей, которым относится, например, де-факто стандартный в России ГОСТ 34, отказываются от возможности создания продукта в процессе его фактического использования, предпочитая строгие и конкретные крупные обязательства.



Рис. 6. Преимущества релизных моделей

Как и любая крайность, и те, и другие проигрывают. Сторонники редких релизов выпускают хорошо отлаженный код, но к моменту запуска в эксплуатацию он успевает устареть - лица, которые формулировали требования, уже ушли из организации-заказчика, на рынке появились новые технологии, сменились приоритеты по функциям. Кроме того, в отрыве от пользователей, программисты зачастую создают совсем не то, что нужно и удобно конечным пользователям, и любые внешние ресурсы и доступное время этого пробела восполнить не могут.

Но и сторонники быстрых релизов сталкиваются с недостатками своего подхода - при быстрых релизах очень сложно менять архитектуру, которая не является идеальной и по той причине, что люди не всегда идеальны при выборе технических решений, и по той, что добавляемые новые функции требуют изменения архитектуры всего продукта, поскольку то, что было оптимальным для старого набора функций, может быть неэффективно для нового набора функций.

Практика изменения архитектуры "потихоньку" - так называемый «рефакторинг», чревата сторонними эффектами, которые приводят к сбоям и рискам, и которые не получается выявить за короткое время релиза.



Рис. 7. Недостатки релизных моделей

Книжка предлагает метод, как сочетать быстрые и редкие релизные циклы для одного и того же продукта, и таким образом распределить задачи между релизами.

Учетная система как инструмент

Почти все знают словосочетание "база данных конфигурационного управления" БДКУ (CMDB), но с какой целью и какая именно информация размещается в ней, как правильно организовать эту

информацию - объяснить уже сложнее. Стандарты об этом умалчивают, а методологии отдельываются достаточно туманными размышлениями.

Например, с сайта HP "Этот набор инструментов CMDB разработан, чтобы собирать, хранить, обновлять и отображать данные о конфигурации ПО и услуг, а также управлять ими. С его помощью можно снизить расходы и количество рисков для организации". Кто должен собирать данные? Почему выбраны именно эти данные? Кто использует эти данные и с какой целью? Какие риски имеются в виду и какие расходы? Дальнейшее описание на эти вопросы полноценно так и не отвечает.



- Какие данные хранятся в CMDB?
- Кто собирает данные?
- Кто использует данные?
- Что обеспечивает актуальность данных?

Рис. 8. База данных конфигурационного управления

Тем не менее, если посмотреть на вопросы конфигурационного управления с точки зрения такого понятия как "отчуждение", то многое становится понятным. В головах и навыках участников процессов хранится множество информации о том, что и как делается. Несмотря на стандартизацию по ИСО 9000, далеко не вся информация о фактических процессах, устройстве систем и применяемых технологиях фиксируется на независимых от участников носителях, а один раз созданная для целей сертификации, быстро устаревает.

Более того, участники фактически заинтересованы в сокрытии такой информации, поскольку это повышает риски компании в случае увольнения сотрудника и этот факт ими используется, чтобы таким нечестным способом повысить свою стоимость, независимо от эффективности.

Особую значимость вопрос приобретает, когда цепочка взаимодействия затрагивает несколько компаний и возникает зависимость от поставщика, когда несмотря на то, что конкретный контракт закрыт, заказчик не имеет возможности выбрать другого поставщика из-за того, что ключевая информация находится в руках поставщика и он последовательно применяет тактику, которая все более и более ограничивает заказчика в информации.

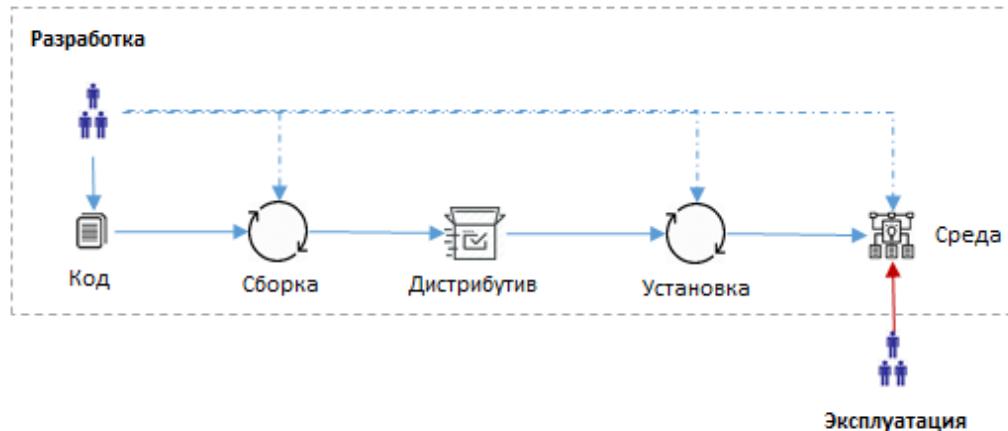


Рис. 9. Отчуждения не произведено, DevOps

Вся эта очевидно неблагополучная ситуация может подаваться в форме, где все выглядит весьма радужно. Например, хорошо известная в последнее время и широко рекламированная технология DevOps, когда в команде разработчика появляются специальные люди, которые берут под свою ответственность сборку, установку, управление конфигурацией среды промышленной эксплуатации и иногда и саму эксплуатацию, таким образом замыкая все вопросы и компетенции на команду разработки, которая становится так называемой "одной точкой отказа".

Поскольку нет использования конфигурационной информации вне команды разработки, то нет необходимости и передавать ее куда-либо, и даже если что-то формально передается, качество этой информации невозможно проверить из-за отсутствия компетенции у получающих информацию и отсутствия лиц, заинтересованных в такой информации как реальных участников процесса.



Рис. 10. Нарушение принципов ISO 9000

Поэтому база данных конфигурационного управления для релизных процессов должна содержать отчуждаемую информацию, необходимую заказчику для возможности смены поставщика, в своем процессе.

Например, системный интегратор должен иметь в БДКУ информацию, источником которой являются подрядчики. Компания, эксплуатирующая заказной программный продукт (с передачей кода), должна, кроме самого кода, иметь достоверную информацию об алгоритмах сборки и установки.



Рис. 11. Частные способы передачи информации и кусочная автоматизация

Никакие инструкции не сделают информацию БДКУ актуальной, если это не будет нужно самим сотрудникам. Поэтому решением в такой ситуации будет то, что учетная система должна служить средством для передачи информации по цепочке в процессе и заменить собою альтернативные механизмы - почту, документы, всевозможные порталы с информацией, включая печально известный Google Docs, в который выкладываются технические детали множества систем, закрытых в том числе, где это все существует практически в бесконтрольном виде. Чтобы это происходило, БДКУ должна быть удобным и адекватным хранилищем информации, где формат данных соответствует фактическим системам.

Еще один, самый главный, удар по неформальному обмену информацией нанесен совмещение функций автоматизации и учета конфигураций в одной системе. Автоматизация облегчает работу людям, страхует их от ошибок, и они готовы будут ради этого вести учет и обеспечивать актуальность информации. Сразу можно сказать, что при внедрении такой системы будет попытка отстоять ситуацию, когда автоматизация обеспечивается другими средствами, не требующими учета.

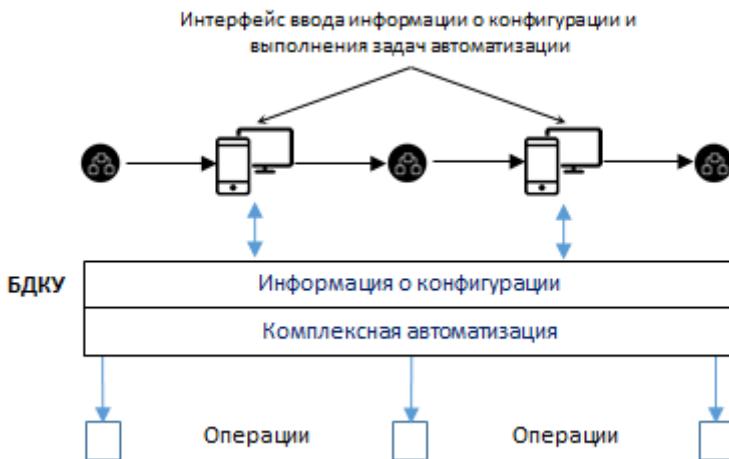


Рис. 12. Комплексная автоматизация на основе единой CMDB

Книга дает конкретные рекомендации по структуре и составу БДКУ, необходимой для отчуждения информации, связанной с релизными процессами, которые в ходе создания и эксплуатации программных систем являются как бы межцеховым конвейером, соединяющим разные "цеха" большого ИТ-завода.

Единая метамодель

Если слова, которыми описан процесс или технология, являются специфичными для данного продукта, то они не смогут быть частью методологии, которую можно применять к разным продуктам. А ведь тщательно отлаженные процессы и технологии являются весьма ценным материалом, содержащим мудрость и знания организации. Проходить тот же путь для каждого нового продукта, использовать каждый раз новые слова или те же, но с разным смыслом, приведет к проблемам с обменом знаниями в организации и понизит ее эффективность.

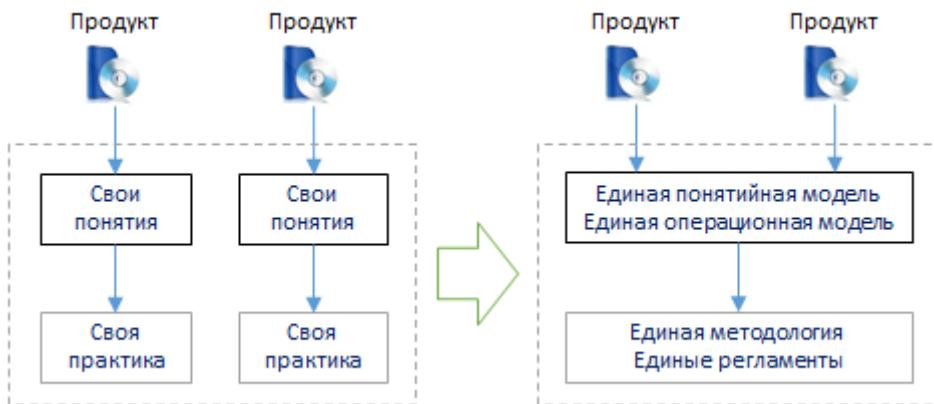


Рис. 13. Единая методология

Если нам удалось описывать разные продукты одними и теми же словами, то кроме организационных эффектов, появляются и технические следствия. Алгоритм, выполненный на основе одинаковых понятий, может применяться к разным продуктам. Т.е. для автоматизации процессов можно будет использовать одни и те же средства.

Так возникает термин "метамодель", которая объединяет понятия и зависимости между ними. Описание конкретного продукта в терминах данной метамодели является моделью продукта в данной метамодели.

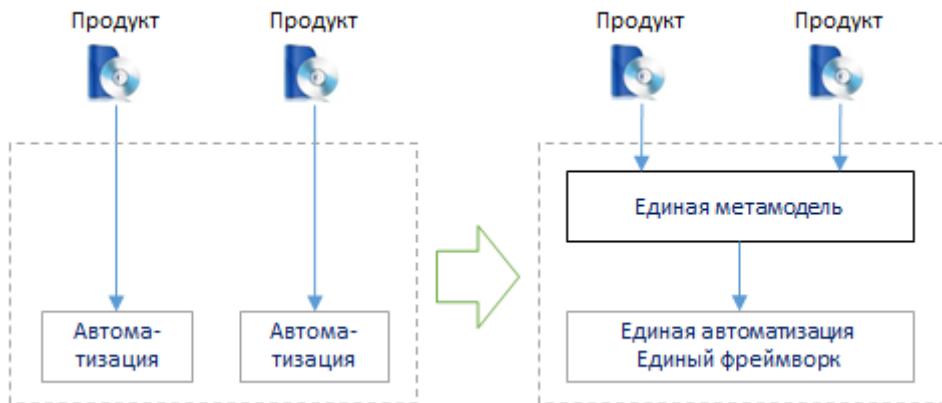


Рис. 14. Единая автоматизация

Становится видна дилемма - либо мы делаем автоматизацию под каждый продукт отдельно, не создавая метамодели и плодя процессы, разнообразные скрипты и утилиты, либо мы начинаем делать единую автоматизацию. Продукты могут быть технически очень разные и для всех операций это приведет к потере эффективности.

С другой стороны, если операции одного рода выполняются разными средствами, это также приводит к беспорядку и проблемам в согласованности действий и данных. Это подводит нас к логическому выводу о том, что единая автоматизация полезна, но там, где виды активностей можно унифицировать. И если активности унифицированы, то в них необходимо отказываться от частных средств автоматизации.

Также единая метамодель может оставлять место для взаимодействия с частным решением, тогда эти средства автоматизации нормально сосуществуют. Например, в метамодели может быть определена операция запуска сервера, а средство ее реализации может быть частным - конкретный и специфичный для данного сервера скрипт.

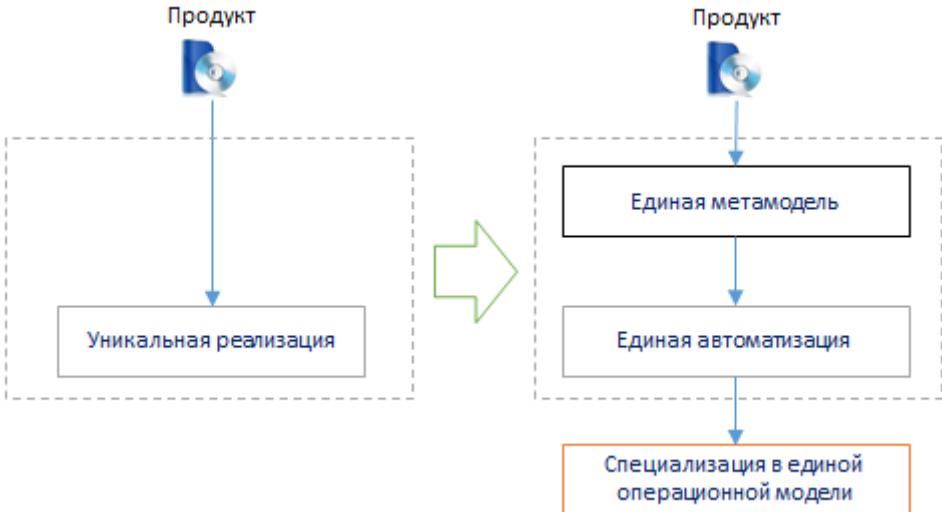


Рис. 15. Специализация в единой операционной модели

Чем больше элементов и связей в метамодели, тем больше полезных операций мы можем на ней определить. Чем больше операций мы покрываем в нашем процессе на уровне метамодели, чем более связанным и прозрачным становится процесс, тем больше автоматизация, тем меньше ручных операций и связанных с ними рисков.

Разумеется, в какой-то момент оказывается, что новый элемент метамодели (или новое правило взаимосвязи элементов) противоречит процессу какого-либо продукта. Это означает одно из двух - либо вы зря включили этот элемент в метамодель, либо процесс требует коррекции. Придется делать выбор, оценивая плюсы и минусы того или иного решения. Без этого элемента, возможно, вам не удастся автоматизировать какие-то важные операции, а изменение существующего процесса может быть трудоемким или медленным делом.



Рис. 16. Управление отклонениями

Тем не менее, именно достаточно полная единая метамодель предоставляет возможности эффективной автоматизации и унификации процессов. Данная книга предлагает такую модель в отношении релизных процессов для программных продуктов, что может послужить хорошей исходной точкой для определения единой метамодели в вашей организации.

Идеология, методология и технология

В любой теории есть ключевые идеи, которые делают эту теорию уникальной и вокруг которых строится все остальное. В данной книге это:

- Подход к рассмотрению продуктов в целом, а не их отдельных сервисов, поскольку продукты содержат в себе определенную платформу (ядро) и отдельные сервисы, которые при практическом рассмотрении не являются независимыми. Именно платформа экономит затраты за счет повторного использования и именно поэтому абстрактная модель микросервисов, которая является альтернативой идеи программных продуктов, не является удобной с точки зрения эффективности и в реальности остается больше умозрительной конструкцией.
- Рассмотрение сквозного релизного процесса, начинающегося с активностей в разработке и заканчивающегося активностями в эксплуатации и центрального предмета этого процесса - релиза программного продукта. Именно релиз является объектом, который объединяет все активности.
- Использование распределенных сред как целостных управляемых экземпляров продукта, с собственной уникальной конфигурацией, а не просто множеств отдельных виртуальных машин, с которыми работают независимо друг от друга.

Микросервисы

- ✗ Нет платформы
- ✗ Скрытые зависимости
- ✗ «Релизы» компонентов

Программные продукты

- ✓ Полные продукты
- ✓ Полные среды
- ✓ Полные релизы



Рис. 17. Микросервисы и продукты

Также важны указанные выше идеи сочетания релизных циклов, БДКУ как инструмента автоматизации и единой метамодели. Все это определяет идеологию программной инженерии как использование комплексного описания мира программных систем, не как единого монолита и не как разрозненных низкоуровневых элементов, а как групп сервисов, объединенных в группы по объективным, инженерным и организационным причинам. Таким образом данная идеология позволяет подняться от низкоуровневых технических деталей и, с другой стороны, структурировать существующие программные системы, соединив уровень управления системами и уровень их реализации.

Полные продукты	Полные релизы	Полные среды
✓ Бинарные файлы	✓ Группа изменений	✓ Серверы приложений
✓ Архивы	✓ Комплексное тестирование	✓ Базы данных
✓ Конфигурационные файлы	✓ Зависимости сборки	✓ Зависимости функционирования
✓ Изменения баз данных	✓ Релизные циклы	✓ Распределенные системы

Рис. 18. Программные продукты

Однако одних идей мало. Необходимо эти идеи совместить с лучшими практиками современного ИТ, применить их к стандартным в ИТ функциям - разработки, тестирования, сборки, установки программного обеспечения, создать взаимно согласованную модель методов и операций, связанных с релизами - то, что можно будет назвать методологией релизного процесса. Для удобства использования структура книги отражает отдельные функции в релизном процессе.

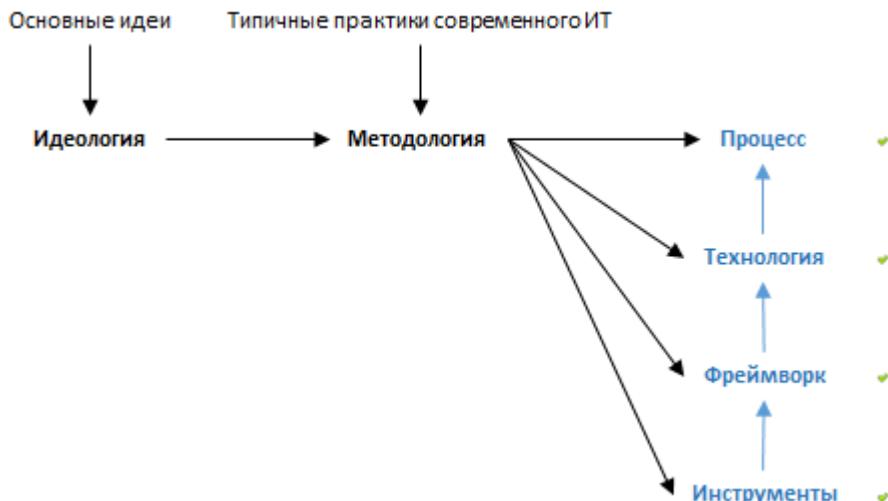


Рис. 19. Связь основных понятий

Наконец, необходимы технические средства как инструментарий, который обеспечивает внедрение и следование методологии. Важно понять, что процесс без средств автоматизации внедрить не получится - и от выбора инструментов, их правильной настройки и последовательного перехода к обязательному их использованию зависит, будет достигнут результат или нет. Здесь необходимо знать, что у каждого инструмента есть своя идеология и методология, и успех и трудоемкость внедрения зависят от того, насколько они сочетаются с той методологией, которую вы пытаетесь реализовать. Сформированный и настроенный фреймворк из таких инструментов определяет технологию выполнения операций релизного процесса, которая приводит весь ИТ-завод в работающее состояние.

Программные продукты

ПОНЯТИЕ ПРОДУКТА

ПОСТАВКА ПРОГРАММНЫХ ПРОДУКТОВ

ТЕХНОЛОГИЧЕСКИЙ СТЕК

ИНФОРМАЦИОННАЯ СИСТЕМА КАК НАБОР ПРИКЛАДНЫХ ПРОДУКТОВ

УПРАВЛЕНИЕ ИЗМЕНЕНИЯМИ

ЖИЗНЕННЫЙ ЦИКЛ ПРОДУКТА

ПРОДУКТ И ПРОЕКТ

ПОРТФЕЛЬ И ЛИНИЯ ПРОДУКТОВ

УПРАВЛЕНИЕ КОНФИГУРАЦИЯМИ

В рамках данной главы вводятся основные понятия и темы, касающиеся программных продуктов и операций с ними, о которых пойдет речь далее.

Понятие продукта

Что такое программный продукт? Что называть продуктом? Зачем нужен такой термин?

Название "продукт" тем более путает, поскольку в русском и английском языке это достаточно разные корни. В русском - от слова "продавать", в английском - от слова "produce", производить. Совершенно разный фокус порождает, например, мнение некоторых менеджеров, что программу для внутреннего использования нельзя называть продуктом, поскольку ее нельзя продать за деньги. То же самое получается, когда мы берем общедоступный компонент из Open Source. Вроде взяли бесплатно, продукт ли это?



Рис. 20. Продукт как предмет производства и использования

С другой стороны, рассматривая разнообразный код в репозиториях GitLab, SourceForge или во внутренних репозиториях компаний, скрипты, которые пишут администраторы, зачастую интуитивно понятно, что нельзя это назвать продуктом. Ведь в каком-то случае не только нет никакой документации, но и даже назвать, как-то идентифицировать этот код невозможно.



Рис. 21. Продукт с точки зрения полезности

Далее, что считать продуктом - программный код или результат его компиляции? Почему Windows 98 и Windows 2000 называются разными продуктами, а не версиями одного продукта?

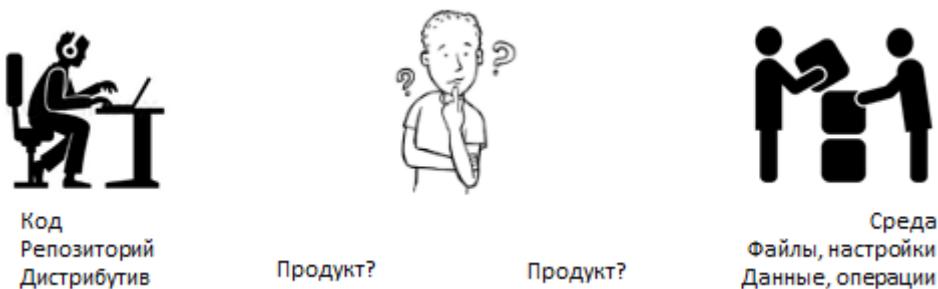


Рис. 22. Программный продукт в разработке и в использовании

Абсолютных истин не существует и полезные правила в отношении того, что является программным продуктом и как с ним лучше всего работать, являются лишь точкой зрения с высоты определенного опыта, подкрепленной конкретными ситуациями и логикой. Для удобства весь материал ниже будем называть методологией UM (URM Methods, Unified Release Management Methods). Сформулируем критерии для программного продукта с точки зрения UM с пояснением:

- **Программный продукт это в первую очередь программный код.** Причина этого в том, что в отличие от полностью готовых и необновляемых изделий типа зубной щетки, программы передаются пользователям в гораздо менее законченном состоянии - и с точки зрения наличия всех необходимых функций, и с точки зрения отсутствия ошибок. Поэтому понятие продукта на практике включает поток изменений. А для этого нужно владеть кодом, иметь возможность вносить изменения.
- **Наличие собственного исходного кода.** В современном мире активного повторного использования чужой код - это нормальная практика. Но отсутствие своего кода является признаком элементарного воровства.
- **Концептуальные рамки.** Продукт имеет конкретный смысл своего существования, цель, которую можно выразить в одном предложении. Иначе отсутствует критерий того, что входит в продукт и что должно входить. Продукт превращается в хаотический набор функций и разваливается, возникает дублирование функций, один и тот же код приписывается нескольким продуктам.
- **Наличие явных физических границ программного кода и дистрибутива.** В рамки продукта входит собственный код, который не является частью никакого другого продукта в качестве кода. При этом в сборке дистрибутива может использоваться код других продуктов в чистом виде или в качестве библиотек, библиотеки могут также поставляться в дистрибутиве продукта. При этом в продукт не входит другой продукт целиком. Например, JDK (Java Development Kit) или Apache Tomcat не могут входить в состав вашего дистрибутива, о чем явно говорится в соответствующих лицензиях, поскольку потоки изменений этих продуктов имеют свою собственную независимую логику и правила.
- **Идентификация, наличие собственного имени.** Дав чему-то конкретное имя, мы можем по отношению к нему в целом применять все остальные технологические понятия, например, протестировать X, установить X, выпустить новую версию X, выгрузить код X, создать пользовательскую документацию для X и т.п., а также мы фиксируем эти активности и факты в виде записей где-либо (так называемые "quality records", записи о качестве). В противном случае, мы применяем эти термины в отношении отдельных частей и теряем понимание, что происходит в целом. Продукт становится нечетким понятием, с неизвестными качествами и состоянием, он перестает быть инженерным объектом.
- **Однозначная ответственность за развитие и поддержку.** Продукт не может развиваться случайным образом - нескоординированные источники изменений неизбежно приходят к конфликту. Поэтому независимо от юридических деталей программным продуктом может называться только то, что имеет признанный всеми участниками однозначный источник изменений - компания или группа лиц с известным контактом. При отсутствии такой однозначности использование продукта в реальной деятельности становится рискованным
- **Документация.** Под документацией можно понимать очень разные вещи, но в данном случае это тот источник информации, который определяет некоторые обязательства продукта по отношению к его пользователям и администраторам. Т.е. если в документации определена функция, то она должна работать. Если для установки продукта необходимо использовать

конкретную версию CryptoPro, то именно с этой версией продукт был протестирован и должно быть понятно, откуда эту версию взять. Если нет документации, то нет и продукта.



Рис. 23. Основные свойства программного продукта

Таким образом, **программный продукт** - идентифицированная совокупность собственного программного кода и внешних программных объектов из других продуктов, находящаяся под контролем известных лиц, документированная и эволюционирующая во времени благодаря потоку изменений, сохраняя при этом концептуальные рамки. Именно продукт - то, что можно последовательно использовать, на чем строить какие-то бизнес-функции и т.д.. Практический вывод - все, что не является программным продуктом в ИТ-инфраструктуре, должно быть превращено в продукты или заменено на продукты.

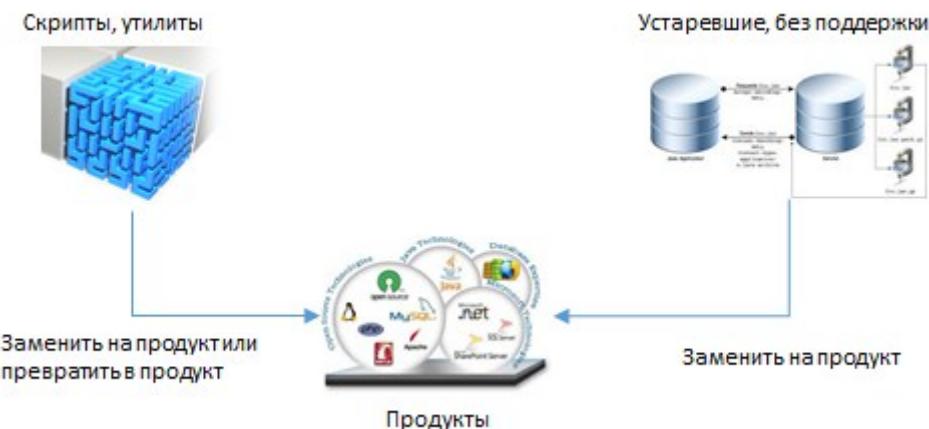


Рис. 24. Ландшафт ИТ-систем и программные продукты

Путь от вроде бы готового кода до промышленной эксплуатации может быть очень сложным, поскольку именно на стадии эксплуатации становятся важны ограничения доступа, производительность, наличие логов, документации и прочее. Если же не дойти до эксплуатации, то все усилия по разработке становятся напрасными вне зависимости от качества самой программы. Методология UM фокусируется на этом пути от кода до программного продукта, так называемой "последней милю" для программного обеспечения. На том, как превратить код в продукт, как доставлять продукт в среду эксплуатации и как эксплуатировать программные продукты.

Поставка программных продуктов

Что такое поставка? Кто, кому и когда поставляет? Каким образом изменения попадают в среду промышленной эксплуатации?

Программные продукты, как уже было сказано, стараются отдавать пользователям еще до того, как они достигнут определенной зрелости. Зрелый продукт обладает всеми практически нужными функциями в пределах своих концептуальных рамок, он практическищен ошибок и максимально

удобен. Но чтобы стать по-настоящему удобным и полезным, продукту необходимо практическое использование. В отрыве от пользователей создать зрелый продукт нельзя. Поэтому де-факто, в начале своего жизненного цикла, в руки пользователя попадает продукт со множеством ошибок, неудобный и с десятой долей количества функций, которые будут у продукта в зрелом состоянии. Если не будет быстрого потока изменений, то пользователь просто прекратит использовать продукт. Поэтому после одной начальной установки будет множество обновлений, которые будут полностью или частично менять продукт, и поток обновлений гораздо важнее первоначальной установки. Эту простую логику зачастую не видят те, кто привык просто "сдавать" проекты (разумеется, необходимость самой процедуры это не отменяет).

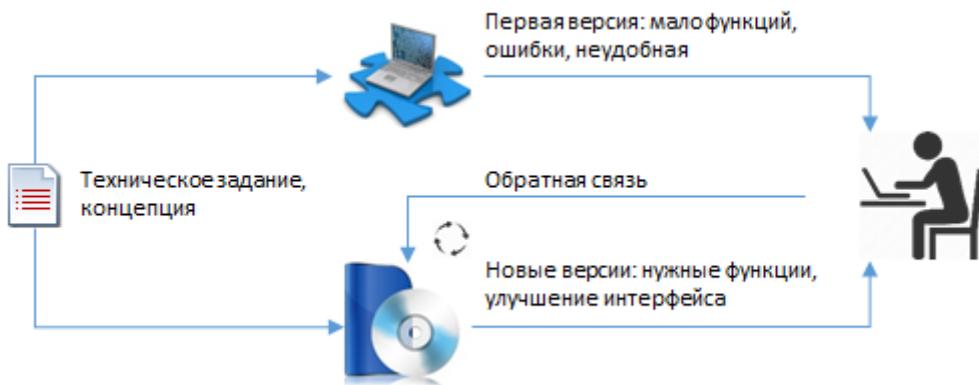


Рис. 25. Развитие программного продукта

Процесс передачи обновлений продукта от разработчика (команды программистов или целой группы компаний) команде или юридическому лицу, отвечающему за эксплуатацию, назовем поставкой. Сам процесс поставки делится на две стадии - выпуск релиза с получением готового дистрибутива и установка дистрибутива в среду промышленной эксплуатации.

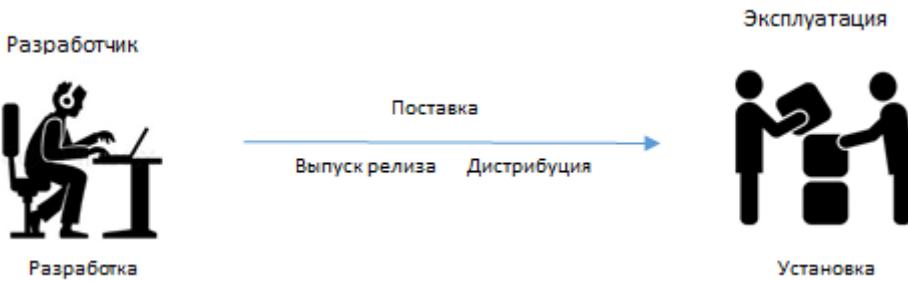


Рис. 26. Процесс поставки программного продукта

Для получения готового дистрибутива обычно нужен итеративный процесс с начальной разработкой и тестированием в специальной среде выпускающего тестирования. Информирование о готовности к установке является признаком готовности дистрибутива. До этого дистрибутив может изменяться. Получение готового дистрибутива выполняется в соответствии с установленным релизным процессом - правилами, которые определяют частоту релизов и их типы.

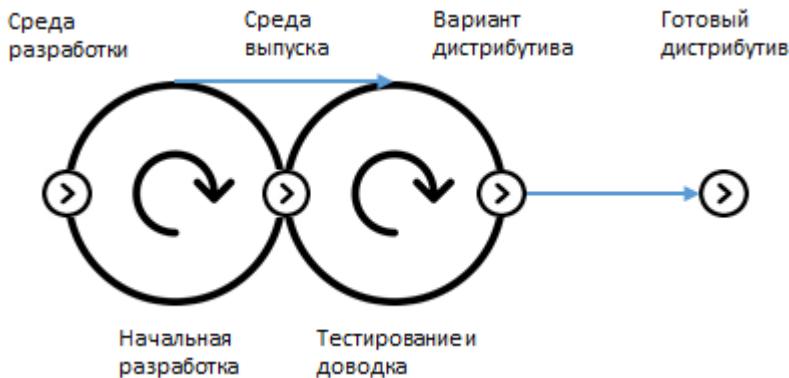


Рис. 27. Итеративный процесс получения дистрибутива

Установка дистрибутива состоит из подготовительных работ (например, публикации дисклеймера, информирования участников установки), изменения файлов в среде, проверки работоспособности среды после установки и фиксации изменений. Поскольку хаотическое изменение среды промышленной эксплуатации, в которой работают пользователи, является плохо мотивированным риском для работы этих пользователей, время установки релизов должно быть ограничено, количество изменений должно быть минимизировано, имеет смысл выделить окна, когда обновление производится, на основании информации о времени, когда нагрузка системы минимальна.

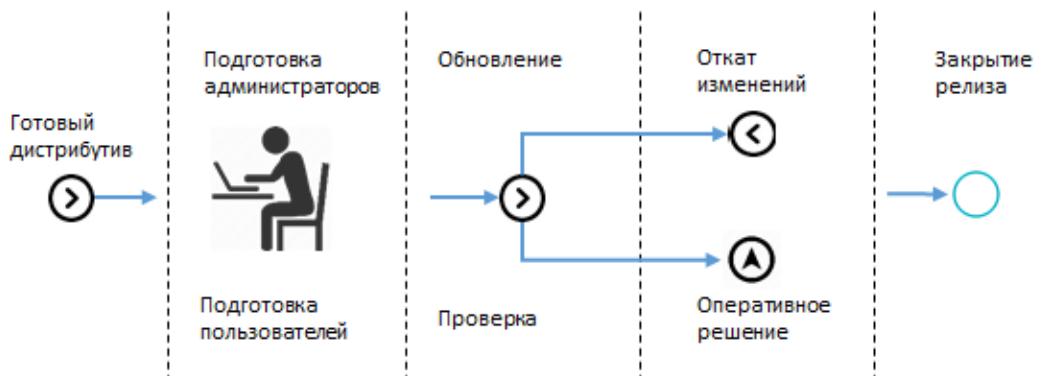


Рис. 28. Установка релиза

Начнем с вопроса о том, кому поставляется продукт. Здесь есть следующие варианты:

- **Персональный продукт.** Здесь частное лицо получает и ставит себе лично программный продукт для своего личного использования и выполнения конкретной прикладной функции. Если функция не относится к системному администрированию, то пользователь не является специалистом-администратором и по сути, профессиональной эксплуатации в данном случае нет. Особенность поставки заключается в том, что именно пользователь - владелец компьютера, и только он может инициировать и разрешить обновление. Процедуры автоматического обновления всегда раздражают пользователей, поскольку это зачастую происходит не вовремя и нарушает чувство собственности пользователя. Продукт является тиражируемым, и разработчик продукта не знает точно, у каких пользователей и какая версия продукта стоит. Релизный цикл заканчивается выпуском дистрибутива и уведомлением для пользователя. Инструкция по обновлению пишется в расчете на обычного пользователя, используются стандартные инструменты для установки типа msi. В методологии UM такие продукты не рассматриваются, хотя многие элементы остаются применимыми.
- **Корпоративный продукт.** Например, антивирус для рабочих мест компании, в котором так же существует сервер, через который происходит обмен информацией и обновление антивирусных баз данных. Использование персональное, но цель корпоративная и контроль происходящего на персональном компьютере тоже идет со стороны администраторов

компании, которые отвечают за профессиональную эксплуатацию. Это смешанный вариант, где есть программы и для серверов, и для персональных рабочих мест. Также возможны корпоративные продукты только для серверов и только для рабочих мест. Т.е. персональное использование не означает, что это персональный продукт. Общее в корпоративных продуктах, что за установку и обслуживание отвечает профессиональная отдельно выделенная команда администраторов, которая является получателем продукта в процессе поставки.



Рис. 29. Кому поставляется продукт

Корпоративные продукты, в свою очередь, подразделяются на несколько существенных с точки зрения поставки вариантов:

- **Заказные продукты.** В данном случае речь идет о продукте, который размещается не на персональных компьютерах, а на серверах и имеет ровно одну установку для промышленной эксплуатации. Частным случаем является случай, когда один заказчик, но у него несколько одинаковых сегментов, в каждом из которых установлен продукт. Например, функция, доступная через локальный датацентр. Для снижения издержек предполагается, что в нормальном режиме везде должна быть одинаковая версия продукта, но фактически процесс обновления растягивается во времени так, что какое-то время в разных сегментах - разные версии. Тем не менее эталонная версия одна и поддерживается ровно одна версия. Релизный цикл заканчивается установкой в среду промышленной эксплуатации, что как правило, ограничивается по времени.
- **Кастомизируемые продукты.** В этом случае заказчиков и инсталляций несколько, но конфигурация и версии известны разработчику. Разработчик на своей территории поддерживает или может воспроизвести основные настройки конкретного заказчика. Выпуск релиза и его установка у заказчика отслеживается разработчиком или даже производится разработчиком (модель DevOps). Релизный цикл заканчивается сменой версии у всех заказчиков, что может затягиваться, но все-таки ограничено. Для кастомизируемых продуктов достаточно типична ситуация, когда основа продукта одна, но под каждого заказчика есть особенная версия кода. Это позволяет иметь разные функции и разную реализацию, но резко усложняет отслеживание изменений и сам процесс разработки. Фактически в данном случае конфигурация программного продукта выполняется не на уровне конечных файлов и настроек, а на уровне программного кода.
- **Тиражируемые продукты.** В этом случае заказчики устанавливают релизы по своему усмотрению, и разработчик не знает, что стоит у заказчика. Релизный цикл заканчивается

выпуском дистрибутива и уведомлением для пользователей. Единственной разумной стратегией сокращения затрат со стороны разработчика является поддержка какого-то количества последних версий.

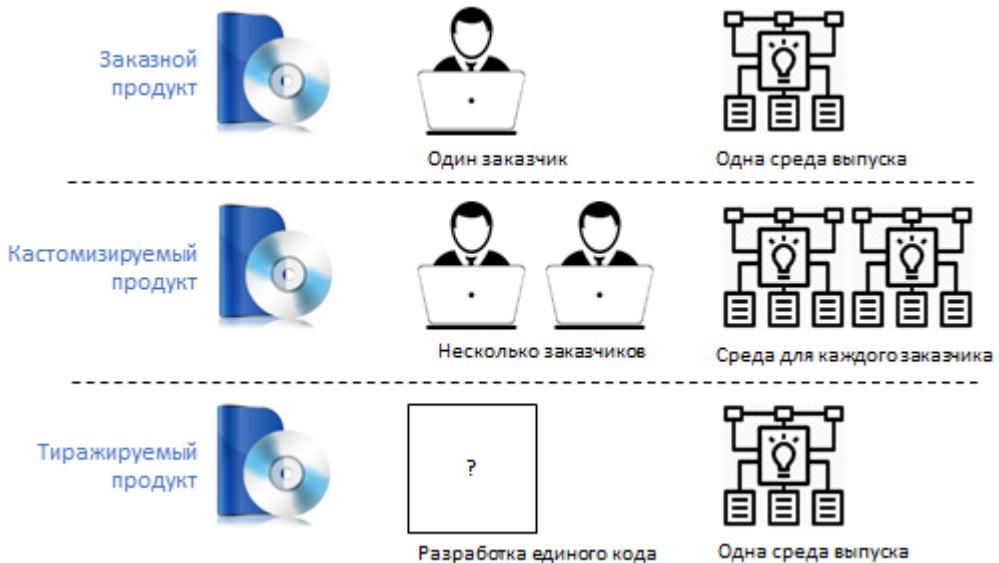


Рис. 30. Виды корпоративных продуктов

Ответ на то, кто поставляет продукт, не такой простой и подразумевает варианты:

- **Разработчик.** Продукт может передаваться как программистом, так и выделенным специалистом, который отвечает как за поддержку процесса разработки, так и за процесс установки и обновления, то что сейчас носит название DevOps. Частным случаем является ситуация, когда выделена команда, которая занимается только релиз-инжинирингом, т.е. выпуском релизов и обеспечением механизма их установки.
- **Интегратор.** Если команды-разработки несколько и, тем более, если они в разных компаниях, за получение дистрибутива продукта, подлежащего установке, и за его итоговое качество, отвечает выделенный интегратор. К данному случаю относится и вариант, когда есть головной разработчик и субподрядные работы. Интегратор может также вообще не заниматься разработкой. В этом случае скорее всего присутствует команда релиз-инжиниринга.
- **Служба заказчика.** Как правило, до начала эксплуатации у заказчика не существует точного понимания о том, каким должен быть продукт, нет нужной экспертизы и сложно проконтролировать качество работы разработчика или интегратора. Заказчику удобно иметь представителя, не связанного с разработкой, который выступает на его стороне и обладает экспертизой по прикладной или по технической части. Продукт проходит через руки службы заказчика и для последнего именно служба заказчика выполняет поставку. В данном случае тоже может существовать команда релиз-инжиниринга, но уже подчиненная эксплуатации.

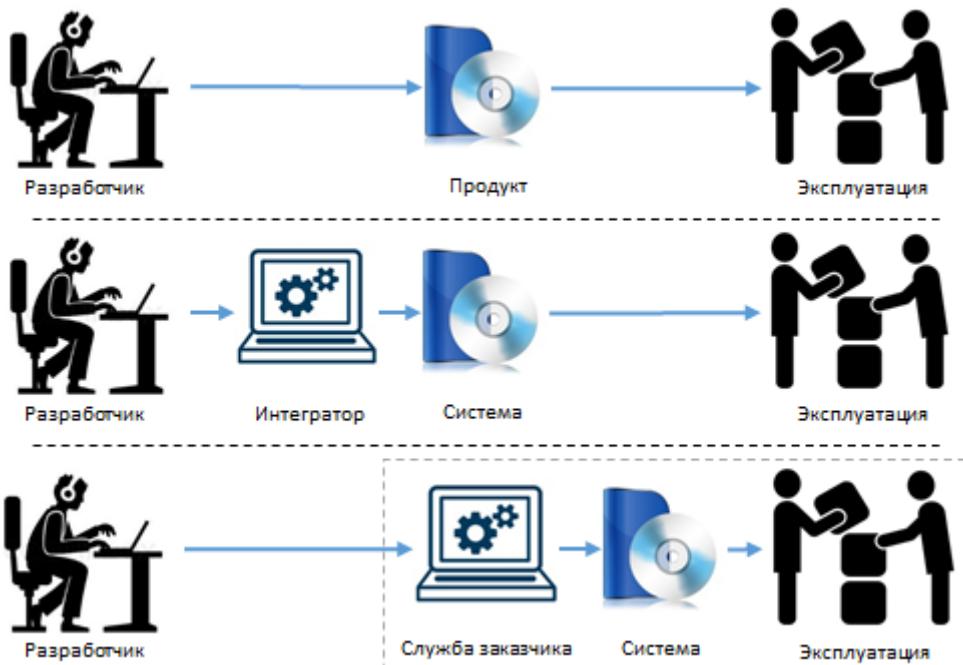


Рис. 31. Поставщик программных продуктов и информационных систем

При начальном развертывании продукт ставится целиком. Зачастую вообще нельзя говорить о начальной установке как о какой-то версии продукта, поскольку в период опытной эксплуатации исполнитель зачастую развертывает продукт своими силами, итеративно. Особенно, если надежная и полностью документированная процедура установки отсутствует. При обновлениях, надо понять принципиально разные варианты того, что будет поставляться:

- **Без дистрибутива.** Практика, при которой существует процедура установки, но не существует (по крайней мере у заказчика) выделенного дистрибутива, соответствующего данному обновлению. Файлы могут скрытым способом устанавливаться в среду промышленной эксплуатации из внешних ресурсов, что конечно же порождает дополнительные риски с точки зрения безопасности. Еще одной проблемой является то, что в такой ситуации нет объекта, на котором можно построить версионный контроль, каким является дистрибутив. В методологии UM такой подход не рекомендуется и такие продукты не рассматриваются, хотя многие элементы остаются применимы.
- **Полный дистрибутив.** В данном случае поставляется соответствующая версия полного дистрибутива продукта, которую можно использовать и для начальной установки и для обновления.
- **Инкрементальный дистрибутив.** Если продукт относительно большой и состоит из множества устанавливаемых элементов, то для частых изменений полный дистрибутив будет избыточным и его обработка (копирование, установка) будет затратной по времени и ресурсам. Чтобы изменить часть элементов среды, достаточно иметь дистрибутив только с данными изменениями. Еще более опасными являются изменения базы данных, где объекты не просто выкладываются в новой версии, а изменяются по алгоритму в скриптах преобразования. Практически невозможно поставлять изменения базы данных кроме как инкрементальным способом. Инкрементальное обновление дает правильный результат только при применении к той версии продукта, для которой оно было создано. Этот вариант является основным по методологии UM.
- **Кумулятивный дистрибутив.** Специальным случаем инкрементального дистрибутива является кумулятивный дистрибутив, который позволяет обновлять до нужной версии продукты сразу нескольких версий. С кумулятивным дистрибутивом не связана какая-либо разработка, это техническое объединение элементов нескольких дистрибутивов так, что установку этих нескольких релизов можно выполнить одной операцией.

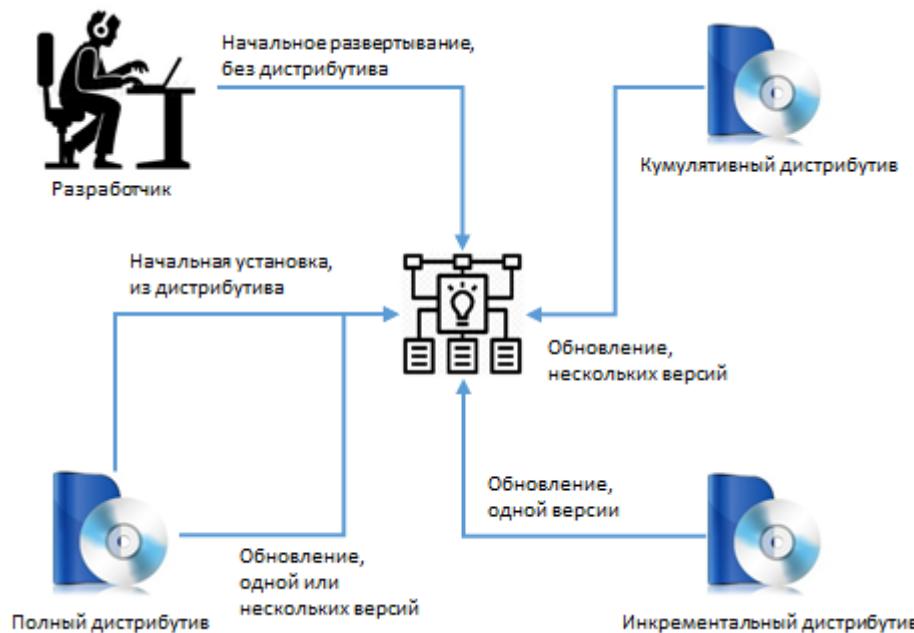


Рис. 32. Виды дистрибутивов

Последний вопрос - когда происходит поставка. Установка дистрибутива подразумевает следующие варианты:

- **Объединенная с выпуском.** Это должно происходить только при выпуске критических обновлений, когда требуется как можно быстрее установить изменение. Во всех остальных случаях выпуск релиза производится по времени отдельно от установки.
- **Синхронизированная с выпуском.** В этом случае факт выпуска сопоставляется с фактом установки и поставка каждого релиза отслеживается как единая процедура. Выпуск следующего релиза невозможен до установки предыдущего релиза. Это важный момент - релиз является изменением продукта и для инкрементального релиза важно понимание исходного состояния. Но при установке в реальную среду промышленной эксплуатации могут вскрыться факты, которые потребуют откат релиза или каких-то срочных изменений, что неизбежно должно повлиять на следующий релиз. Чтобы установка не держала следующий релиз, необходимо ограничить временное окно для установки. Этот вариант является рекомендуемой формой по методологии UM.
- **Свободная установка.** Если возможными проблемами от предыдущего релиза пренебрегают, то выпуск релизов происходит вне связи с их установкой и заказчик сам решает, когда устанавливать релиз. Разумеется, при этом необходимо следовать правилам применимости релиза. Два не установленных инкрементальных релиза приведут к необходимости их последовательного применения, если нет кумулятивного дистрибутива или соответствующего полного дистрибутива.

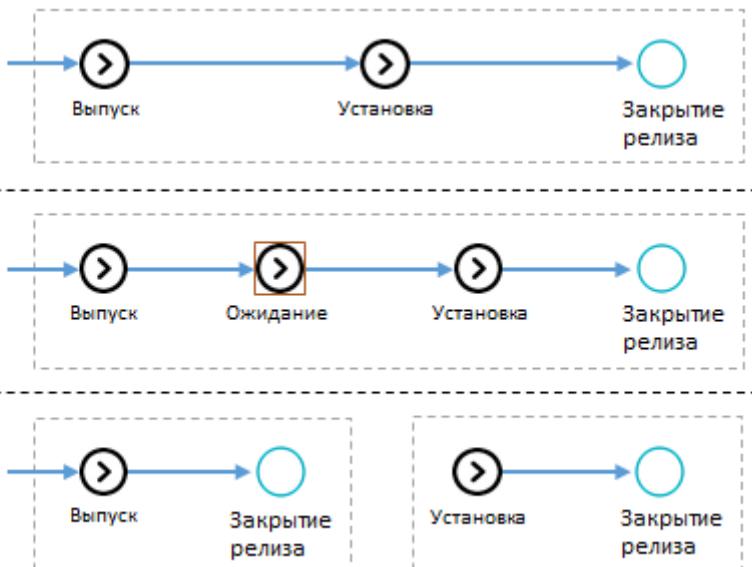


Рис. 33. Синхронизация выпуска и установки релиза

Технологический стек

Чтобы управлять программными системами, необходимо определить их структуру, принципы, по которым части системы взаимодействуют друг с другом и роль, которую эти части играют по отношению друг к другу. Разные роли требуют разного управления и подхода. Деление происходит как горизонтально, так и вертикально.

- **Горизонтальное деление.** Определяет прикладную архитектуру систем как разбиение на функции, которые определены на верхнем уровне, и которые сопоставлены с инфраструктурным расположением - датацентр, подсеть, хост. Для того, чтобы части системы друг с другом взаимодействовали, необходимо наличие прикладного и технологического протоколов взаимодействия. Чтобы сделать развитие систем устойчивым и снизить затраты, используют готовые технологические протоколы, решающие типовые задачи физического взаимодействия, такие как гарантированная доставка, синхронное и асинхронное взаимодействие, маршрутизация, авторизация и т.п., и готовые программы, которые реализуют эти протоколы - такие как COM/CORBA/EMS/ESB. Данные продукты обычно содержит выделенные процессы, через которые проходит сигнал от одной части системы к другой, и библиотеки, при помощи которых прикладные программы соединяются с этими технологическими процессами и поверх них уже реализуют конечные прикладные протоколы. Это то, что называется программным обеспечением промежуточного слоя (middleware).
- **Вертикальное деление.** Определяет технологическую архитектуру, которая обеспечивает реализацию прикладных функций. В мире современного прикладного программирования существует множество готовых библиотек, систем управления базами данных, типовых конфигурируемых процессов, серверов приложений, фреймворков, которые применимы для разных прикладных задач. Эти программы могут быть бесплатными или недорогими по сравнению с вложенными в них ресурсами, и поэтому их используют, даже если приходится под них подстраиваться или есть какие-то проблемы. Эти готовые продукты в свою очередь могут опираться на другие продукты. Для решения схожих задач одна команда разработки как правило применяет единый комплект готовых решений.



Рис. 34. Технологический стек

Совокупность готовых используемых продуктов с учетом их зависимостей и называется технологическим стеком. Разные части системы могут быть реализованы на разных технологических стеках. Повторно используемая совокупность платформенных продуктов и технологического стека называется технологической платформой.

Информационная система как набор прикладных продуктов

Что из себя представляют упомянутые выше части системы, если исключить из рассмотрения продукты технологического стека? Как управлять изменениями в системах? Система или системы, что такое подсистемы? Как связана инфраструктура и информационные системы?

Как было сказано в начале, необходимо все программное обеспечение сводить к программным продуктам. Поэтому де-факто система состоит из прикладных продуктов и так называемого унаследованного программного обеспечения (Legacy), которое на данный момент не подпадает под определение программного продукта, но тем не менее существует. Методология UM не отвечает на вопрос, как работать с унаследованным ПО.

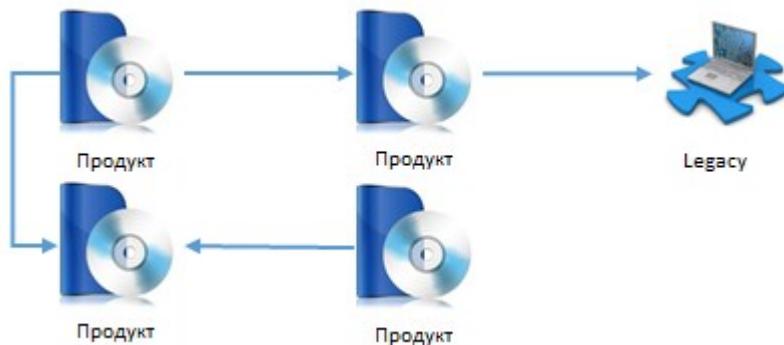


Рис. 35. Информационная система

Что касается программных продуктов, то деление системы на них может быть выполнено разным способом (если это по каким-то другим причинам еще не сложилось). Принципы деления должны быть следующие:

- **Единое владение кодом.** Когда одним продуктом занимаются несколько команд, возникают дополнительные издержки на коммуникации и совмещение разных технологических решений.
- **Выделенный поток изменений.** По методологии UM, продукт имеет выделенную релизную историю как совокупность версий и связанных с ними дистрибутивов, и в любой момент любой

компонент продукта в среде промышленной эксплуатации относится к определенной версии продукта.

- **Инкапсуляция реализации.** Если способ реализации какой-то функции сосредоточен в одном продукте, то это облегчает модификации при технологическом развитии.
- **Минимизация внешних связей.** Так же как и для низкоуровневых программных компонентов, имеет смысл группировать в продукт набор элементов, между которыми больше связей, чем между группой и остальной частью системы.
- **Концептуальная определенность.** Продукт по определению имеет заранее определенную концепцию. Если в системе есть часть с четко определенными рамками функций, то эту часть является кандидатом на учет в качестве продукта.

Концептуальные рамки

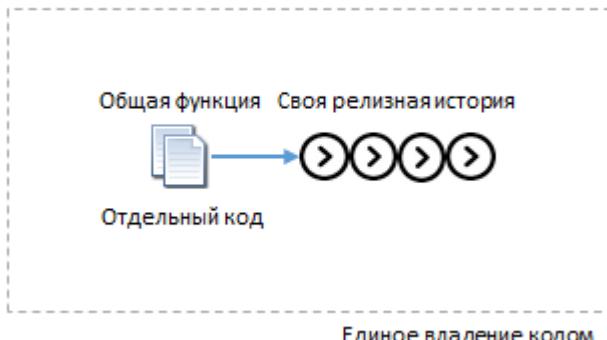


Рис. 36. Идентификация программных продуктов

Если для вместо готового продукта технологического стека был создан продукт собственной разработки, который используется в нескольких других продуктах, то такой продукт называется платформенным. Платформенный продукт может также представлять собой лишь набор библиотек, используемый при сборке прикладных продуктов, и к дистрибутивам которого никогда не применяется процедура установки. В итоге, если исключить унаследованное ПО, технологический стек и платформенные продукты, система распадается на линейный список прикладных продуктов. Необходимо учесть, что в состав дистрибутива одного продукта могут также входить элементы из другого продукта, попадающие туда в процессе сборки первого продукта.



Рис. 37. Платформенные продукты

Элементы кода продукта и элементы дистрибутива продукта составляют "конфигурацию продукта".

- Конфигурация собственного кода (состав проектов, артефакты сборки, зависимости)
- Конфигурация элементов других продуктов (код, библиотеки)
- Конфигурация собственного дистрибутива (элементы дистрибутива)
- Параметры конфигурации (опции, варианты, зависимость от среды)

Рис. 38. Конфигурация продукта

Понятие системы плохо определено с точки зрения ее рамок. Системой могут называть как большой конгломерат приложений, относящихся к какой-то тематике, так и один компонент, выполняющий достаточно узкую функцию. При развитии информационного пространства эти системы интегрируют друг с другом, и на этом множестве определяют подмножества, давая им отдельные имена.

В принципе эта логическая группировка, иногда юридически обоснованная, с технической точки зрения не очень важна. Если система соответствует одному потоку изменений, и по сути соответствует определению продукта, мы ее можем считать продуктом. Объединение всех систем в области внимания можно считать одной системой с подсистемами, соответствующими используемым именам. Подсистемы в общем случае вложены друг в друга и пересекаются.

Для упрощения идентификации из всего списка подсистем имеет смысл выделить список непересекающихся подсистем больше чем из одного продукта, которые полностью покрывают всю систему. Иными словами, из множества используемых названий сформировать невырожденный двухуровневый список - подсистема/продукт. Это даст нам "конфигурацию прикладного ПО".

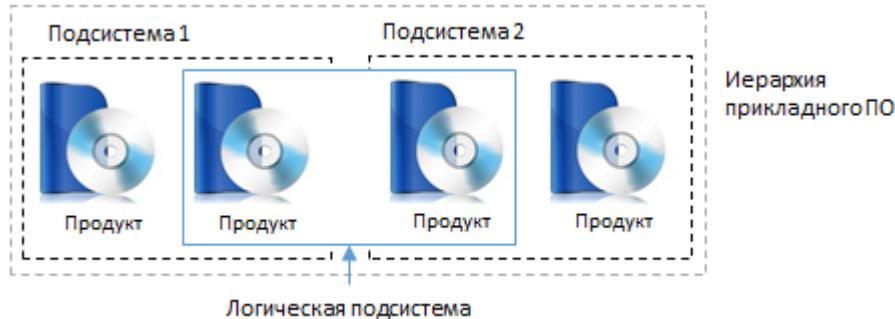


Рис. 39. Конфигурация прикладного ПО и подсистемы

Элементы продуктов располагается на элементах инфраструктуры, которые также имеют иерархический вид - датацентр (отдельно обслуживающийся и располагающийся физический массив компьютерного оборудования)/подсеть/хост. Т.е. элементы одной иерархии на элементах другой иерархии. При этом на одном элементе инфраструктуры могут находиться элементы нескольких продуктов. Но терминалные элементы продукта определяются так, чтобы один элемент находился ровно на одном элементе инфраструктуры, на одном хосте. Иерархия инфраструктурных элементов - это "конфигурация инфраструктуры".

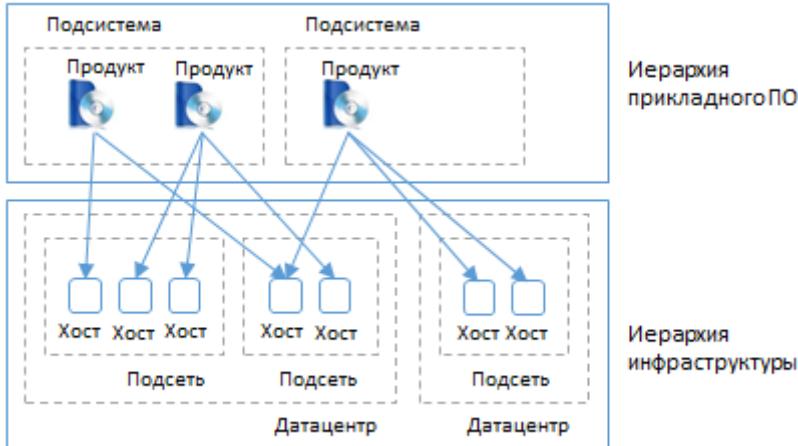


Рис. 40. Продукты и конфигурация инфраструктуры

Управление изменениями

Если рассмотреть среду промышленной эксплуатации с точки зрения изменений, то в ней может меняться продукт или инфраструктура. Продукт меняется в части своей версии, размещения на инфраструктуре и в области своих данных, порождаемых операциями с продуктом. Данные делятся на обязательные, конфигурационные и операционные. Обязательные данные появляются и изменяются при установке версии продукта. Конфигурационные данные обеспечивают привязку продукта к данной промышленной среде эксплуатации. Операционные данные появляются в результате прикладных операций, которые продукт реализует.

Вспомним, что оригиналом продукта является код, который хранится вне среды промышленной эксплуатации. Это означает, что поток изменений в среде эксплуатации должен быть увязан с потоком изменений кода. Рассинхронизация приведет к невозможности поддерживать среду промышленной эксплуатации.

Обеспечение согласованности и непротиворечивости изменений, соответствия документации, минимизация воздействия на пользователей и связанных рисков, планирование соответствующих работ составляет основное содержание дисциплины "управления изменениями".



Рис. 41. Изменения в среде промышленной эксплуатации

Жизненный цикл продукта

Продукт рождается и умирает. В ходе жизни его развитие можно проследить по версиям. В процессе развития он может стать зрелым, приблизиться к полноте реализации своей концепции, а может быстро умереть. В жизненном цикле продукта можно выделить следующие стадии:

- Определение концепции
- Создание и выпуск в свет первой версии.
- Развитие продукта.
- Эксплуатация практически без изменений
- Миграция на альтернативные продукты.
- Прекращение эксплуатации.

Хотя вечных продуктов и не бывает, срок жизни продукта характеризует качество концепции и определяет суммарную пользу от продукта. Польза может иметь денежное или иное воплощение. Для увеличения пользы, как уже говорилось, необходимо максимально обратить внимание на развитие и на выпуск первой версии, позволяющий войти в стадию развития.



Рис. 42. Жизненный цикл продукта

Продукт и проект

Что такое проект? А что такое проект сборки? Чем проект отличается от продукта и процесса?

Данные понятия часто путают. Еще больше путаницы создает то, что один репозиторий кода для сборки тоже называют проектом. Попробуем более четко определить, что есть проект и как он связан с продуктом, определенным выше.

Если продукт - это объект, обладающий некоторыми свойствами, то проект - это совокупность действий. Методология UM предпочитает соглашаться с теми методологиями, где проект заведомо ограничен по срокам. У проекта всегда есть конечный результат, ради которого проект затевался. Проект, в котором конечного результата нет, а есть какая-то постоянно генерируемая польза, называется процессом.

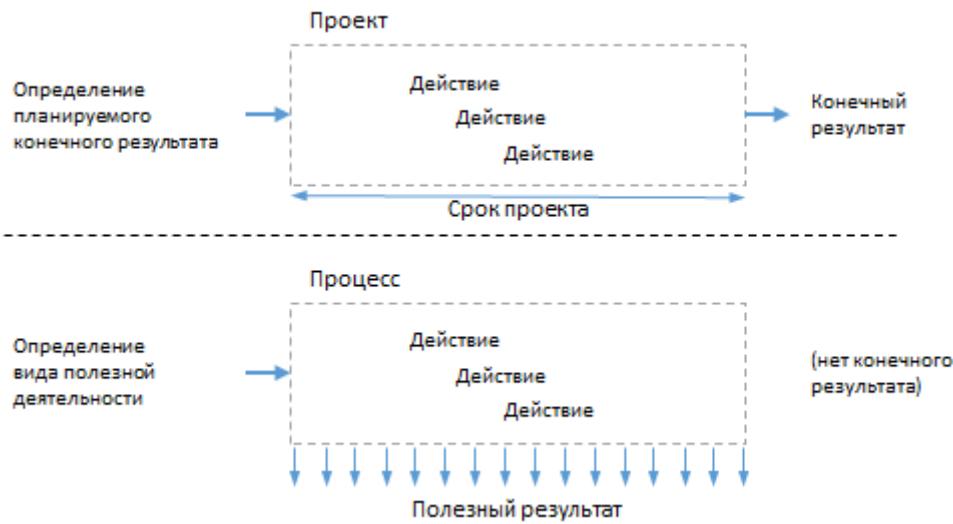


Рис. 43. Результаты проекта и процесса

И процесс, и проект - группировка действий, но проект порождает результат в своем конце (иногда проект разбивают на известные заранее этапы), а процесс служит для организации потоков однотипных операций, каждая из которых имеет результат. При этом извне на вход процессу согласно известному интерфейсу приходит запрос, который порождает такую типовую операцию, которая дает результат и завершается. Процесс может быть организован на ограниченный период времени или без исходного ограничения. Выполнение конкретной операции процесса может иметь настолько сложный и уникальный характер, что для ее реализации применяется проект. И наоборот, проект может организовать часть активностей внутри себя как один или несколько процессов.

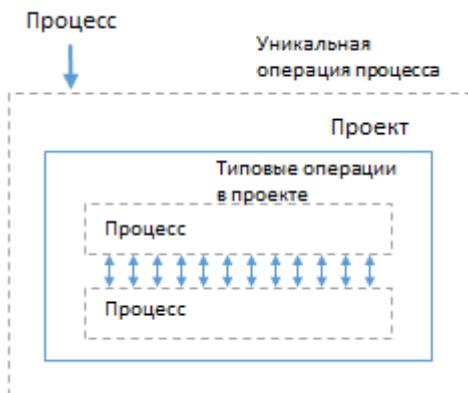


Рис. 44. Взаимосвязь проекта и процесса

Есть частый случай, когда по какой-то теме начинают работы, выделяют ресурсы, но нет ни проекта, ни процесса. Данное состояние называют направлением работ. Впоследствии, для повышения рациональности расходования средств, работы внутри направления группируют в проекты и процессы.

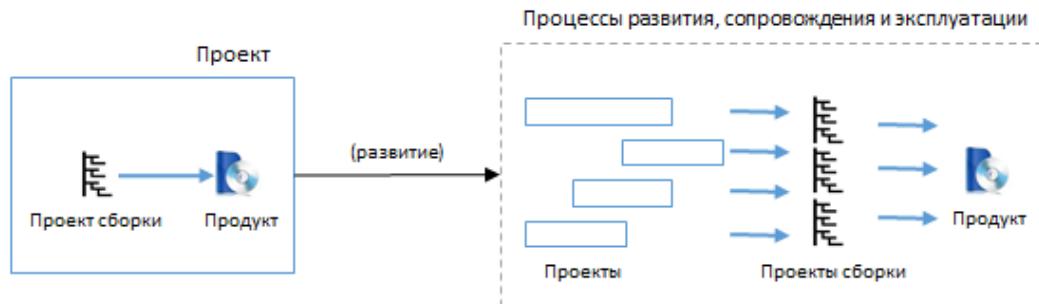


Рис. 45. Проекты, процессы и продукт

Проект сборки - это репозиторий кода, к которому можно применить операцию сборки. Данный термин в корне отличается от проекта как группы активностей. Однако все три термина - продукт, проект и проект сборки могут относиться к одному и тому же для небольших проектов с одним репозиторием кода, сборка которого дает версию одного продукта. Это частый случай на начальном этапе большинства разработок и поэтому эти понятия так часто путают. Но если продукт развивается, то эти понятия постепенно расходятся - для развития или внедрения продукта организуется несколько проектов, код продукта разрастается до нескольких проектов сборки, организуются процессы сопровождения и эксплуатации.

Портфель и линия продуктов

Группа продуктов, находящаяся в ответственности одной организационной единицы, называется портфелем продуктов. Если в информационной системе ответственность распределена по нескольким группам, то возникает естественная задача учета каждой группы продуктов отдельно. Системные действия в соответствующей организационной единице также применяются в рамках своих полномочий - применительно к своему портфелю продуктов. В таком случае целесообразно ставить конфигурационное управление и применять данную методологию отдельно по каждому портфелю продуктов.

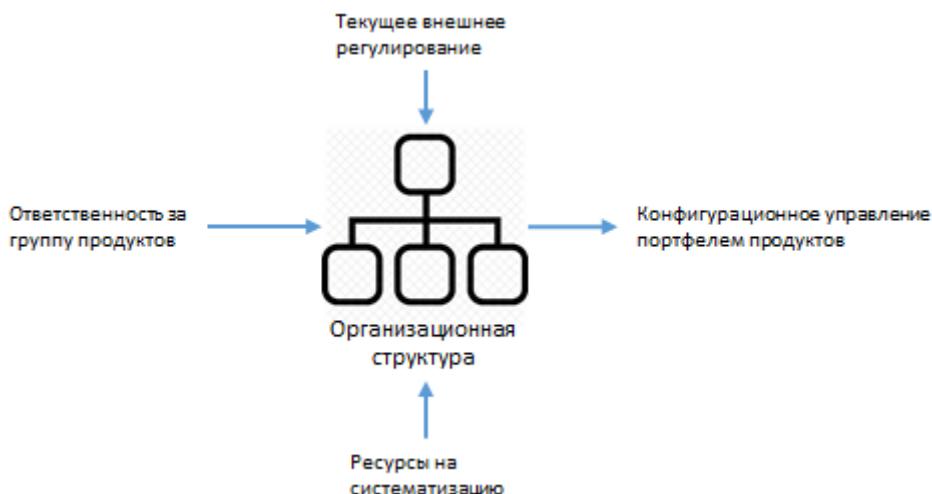


Рис. 46. Портфель продуктов

Термин "линия продуктов" (product line) может иметь две причины для своего существования и два толкования. Первой причиной является организация бизнеса в какой-то области с точки зрения маркетинга, и к одной такой линии продуктов относятся продукты, в общем случае совершенно разные по способу реализации (например, по технологическому стеку). Второй причиной и вторым видом линии продуктов является технологическое единство продуктов, входящих в линию. В этом случае продукты построены на единой технологической платформе. Более того, конечные продукты линии могут быть

лишь разными конфигурациями одного платформенного продукта. Единая технологическая платформа позволяет существенно сократить затраты на разработку линии разных продуктов, придать им единый стиль или общие полезные свойства.

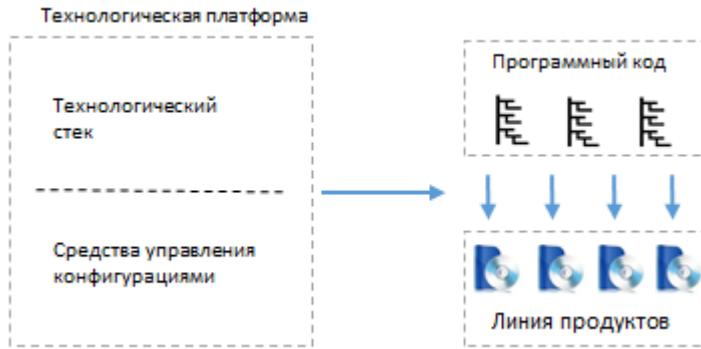


Рис. 47. Линия продуктов

Управление конфигурациями

Краеугольными камнями конфигурационного управления являются идентификация элементов конфигураций, учет смены состава элементов, их взаимоотношения и их версий. Но что за элементы имеются в виду, у каких есть версии, что надо учитывать, о каком взаимоотношении идет речь? Можно ли что-то не учитывать и к чему это приведет? Что такое база данных конфигурационного управления?

Почти все в мире имеет какую-то идентификацию, даже если наблюдателю это сразу не видно - например, каждая функция в программе однозначно идентифицируется компилятором. На учет "всего существующего" можно затратить существенные ресурсы, и очень важно определить цель, область действия учета и того, кто этим будет заниматься. Для упомянутых выше конфигурационных элементов методология UМ определяет следующее:

- **Конфигурация прикладного ПО.** Определяет список продуктов, сгруппированный по подсистемам для удобства запоминания. Целью является создание единой карты происходящего в разработке, чтобы можно было понять, какой текущий результат фактически получен от проектов. Областью действий является соответствующий портфель продуктов. Занимается этим руководство соответствующей организационной единицы.
- **Конфигурация продукта.** Определяет проекты сборки - репозитории кода, зависимости от других продуктов, элементы дистрибутива. Позволяет контролировать программный код и выпускать релизы. Относится к релизному процессу и тем, кто за него отвечает.
- **Конфигурация среды.** Определяет какие функции продукта доступны в среде, какая степень масштабируемости и отказозащищенности реализована. Еще один случай - уникальные параметры среды, передаваемые программе (например, название домена). Необходима для управления ресурсами и сервисом. Выполняется владельцем соответствующей среды - тем, кто отвечает за то, чтобы среда функционировала и продукт выполнял соответствующие функции для определенных пользователей в соответствии с существующими обязательствами.
- **Конфигурация программы.** Это параметры в конфигурационных файлах, которые связаны не с привязкой к параметрам среды, а определяют ветки логики программы, которые включены или выключены в данной среде. Как правило, программы пишутся с созданием области гибкости, так, что если исходно вероятны несколько вариантов использования, то вместо выбора одного варианта, программируются оба и в конфигурационный файл выводится переключатель - какой вариант использовать. Как правило, это зона управления разработчиков, но некоторые параметры документируются и сотрудникам эксплуатации дается возможность самим определить эти параметры.

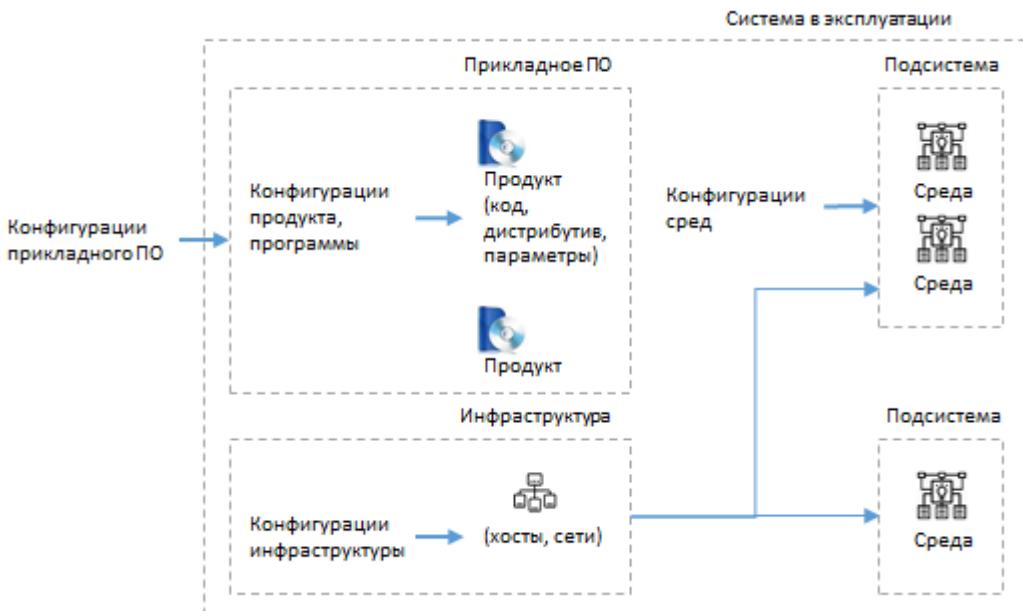


Рис. 48. Области управления конфигурациями

По методологии УМ версии ведутся только применительно к продукту. Разумеется, при хранении элементов в системе контроля версий версии будут у любых отдельных файлов, но это не те версии, в присвоении которых участвует человек. Все части продукта имеют версию продукта - элементы дистрибутива, конфигурационные файлы. Как следствие инкрементальных релизов, конкретная среда состоит из элементов продукта, относящихся к конкретным релизам продукта, т.е. взятых из соответствующих дистрибутивов. И здесь есть варианты:

- **Линейный выпуск релизов.** Упорядоченность версий продукта отражает содержание - каждая последующая версия продукта содержит изменения предыдущих версий. Версии устанавливаются строго в порядке возрастания. В этом случае всегда можно сказать, какой версии продукт установлен на данный момент. Эта версия соответствует версии последнего установленного дистрибутива, а содержание равно кумулятивному дистрибутиву из последнего полного дистрибутива и последующих инкрементальных дистрибутивов, что в свою очередь соответствует текущему состоянию кода (если не учитывать изменения, еще не выведенные в релиз). Этот вариант является рекомендуемой формой по методологии УМ.
- **Релизы с ветвлением.** В этом случае существуют отдельные цепочки дистрибутивов (например, ветка для одного заказчика, ветка для другого). В данной ситуации по сути каждая ветка соответствует отдельному продукту и затраты на разработку и управление конфигурацией существенно возрастают.
- **Релизы компонентов.** Данный вариант - когда каждый дистрибутив меняет какую-то часть системы, и установка является необязательной или необязательно в порядке версий. В результате в среде возникает конфигурация, которой нельзя сопоставить конкретную версию продукта и можно говорить лишь о версиях частей продукта. Т.е. продукт перестает быть целостным и распадается на части.

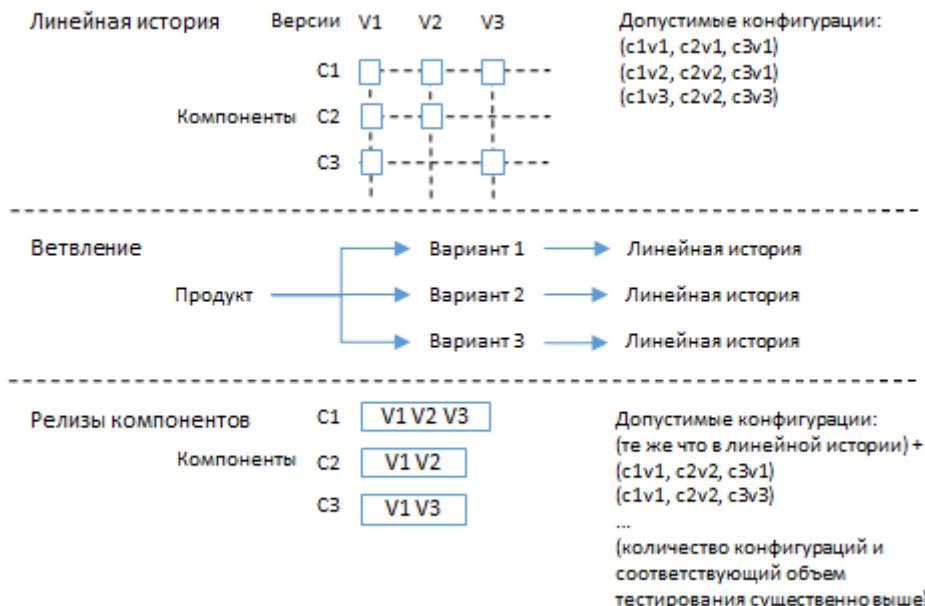


Рис. 49. Варианты выпуска релизов как изменений конфигурации

База данных конфигурационного управления обеспечивает хранение идентифицированных элементов конфигурационного управления. По методологии УМ здесь хранятся все выше указанные виды конфигурационных единиц.



Рис. 50. База данных конфигурационного управления

ОПЕРАЦИИ

Программный код	41
Сборка программного кода	53
Выпуск релизов.....	67
Создание и поддержка сред	99
Установка изменений.....	129
Мониторинг.....	163
Документация	175

Программный код

Исходный код и исходные материалы программного продукта
Ресурс контроля версий и репозитории системы контроля версий
Репозиторий исходного кода
Ветки и теги репозитория исходного кода
Компилируемый и некомпилируемый код
Код и библиотеки из сторонних продуктов
Код баз данных
Конфигурируемый программный код

В рамках данной главы описывается что из себя представляет программный код в программных продуктах, какую он имеет структуру, какое его место в исходных материалах продукта, в дистрибутиве, в среде промышленной эксплуатации.

Исходный код и исходные материалы

Слово "код" очень мудро уводит от четкого определения, что имеется в виду. Самое простое восприятие, что это текст на некотором языке программирования, оказывается неверным после помещения в репозиторий кода картинок, документов (которые попадают внутрь программы), сертификатов, готовых библиотек и прочих объектов достаточно произвольного формата. Поэтому сначала стоит сказать, что исходный код является частью исходных материалов программного продукта и определиться с последним понятием.



Рис. 51. Материалы

Если рассмотреть конкретный программный продукт, то вокруг него вращаются практически все виды активностей ИТ-жизни. Эти активности порождают артефакты или иными словами, материалы. Важно отметить, что хотя эти активности практически непрерывно меняют продукт, для внешнего мира продукт дискретен в виде своих релизов. Поэтому материалы делятся на два существенных вида - на те, что являются промежуточными при создании релизов и на те, что входят в состав релиза. Промежуточные материалы можно условно назвать материалами проекта, а материалы релизов - материалами продукта. Далее, некоторые материалы являются исходными, а некоторые - результатом преобразования исходных. Например, "to be design" это проектная документация, "as is design" - документация продукта, код на языке C++ - исходный материал продукта, а исполняемый exe-файл - просто материал продукта.

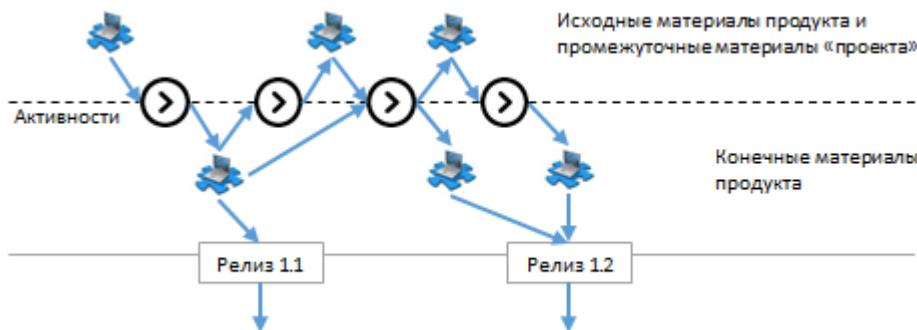


Рис. 52. Материалы продукта и проекта

Материалы продукта с другой стороны можно разделить на технические, абсолютно необходимые для функционирования продукта (программный код, дистрибутив) и организационные, которые являются лишь прототипом с точки зрения реального процесса (методические материалы, инструкции, коммерческие конфигурации). Технические исходные материалы и являются исходным кодом программного продукта.

Ресурс контроля версий и репозитории

Материалы продукта обычно держатся под контролем, поскольку они меняются от релиза к релизу и важно не потерять результаты предыдущих работ. Внутри выпуска релизов, для отслеживания и управления задачами также фиксируются промежуточные состояния при помощи систем управления версиями, таких как Subversion, Git (для управления кодом), или таких как Confluence, PowerPoint (для управления документацией).

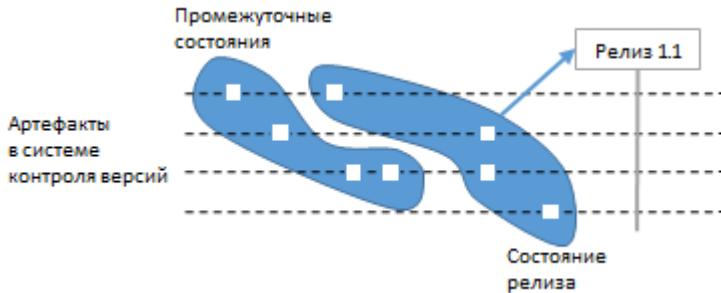


Рис. 53. Состояния контроля версий

Для управления версиями в разных инструментах применяются хоть и похожие, но иногда существенно разные методики. Сервер системы контроля версий, на котором реализована эта логика, хранит единую базу данных своего формата. Эта база данных делится на репозитории системы контроля версий, каждый из которых полностью независим от других репозиториев. База данных на одном сервере в методологии UM считается отдельным ресурсом контроля версий.

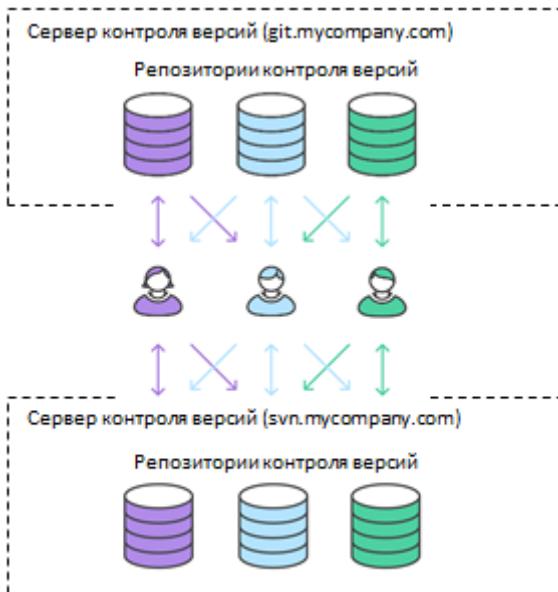


Рис. 54. Ресурсы контроля версий

Репозиторий системы контроля версий хранит то, что решит его владелец - материалы проекта и материалы продукта. Методология UM рекомендует независимо от конкретной структуры избегать ситуации, когда папки структуры репозитория верхнего уровня содержат и материалы продукта, и материалы проекта.

Репозиторий исходного кода

Исходный код является частью материалов продукта и также хранится внутри репозитория системы контроля версий. Исходный код имеет свою структуру, которая тесно связана с процедурой сборки дистрибутивов. Для выпуска инкрементальных релизов выполняется сборка части исходного кода, однако здесь обычно есть технические ограничения, которые заставляют выполнять сборку более крупных единиц кода, чем часть, подвергшаяся изменениям, т.е. есть минимально допустимая единица сборки при выпуске релизов. Именно эта минимальная единица называется репозиторием (проектом) исходного кода - это то, что может быть выгружено из репозитория контроля версий и к которому может быть применена процедура сборки релиза.

В результате в одном репозитории системы контроля версий в разных подкаталогах может храниться несколько репозиториев исходного кода. В остальных каталогах могут храниться другие материалы продукта и проекта. Самым простым случаем является ситуация, когда в репозитории системы контроля версий хранится только проект исходного кода и ничего более. Но с развитием продукта код усложняется, появляются новые модули и независимые от первого проекты сборки, накапливаются материалы продукта, где-то приходится хранить промежуточные материалы проекта, заводятся новые репозитории контроля версий.

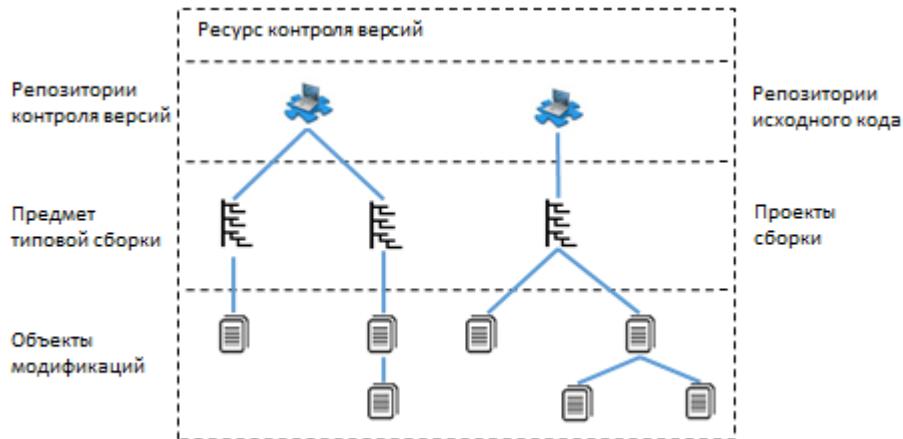


Рис. 55. Репозитории исходного кода и проекты сборки

При выборе системы контроля версий для хранения материалов нужно учесть их особенности. Например, git ориентирован именно на хранение исходного кода и при создании тегов и веток делает копию всего репозитория. Поэтому как правило в репозитории git либо хранится только код, либо только не-код. Subversion-репозиторий более гибкий - в нем ветки и теги являются просто копиями ветки любого уровня и в таком репозитории могут хранить разные виды материалов. Также возможно, что для хранения документов и иных организационных материалов используется специализированная система контроля версий, например, Confluence.

Ветки и теги репозитория исходного кода

К репозиторию исходного кода могут быть применены операции ветвления. В исходном состоянии репозиторий содержит одну основную ветку и какие-то версии файлов в ней. Операция ветвления берет состояние какой-либо ветки репозитория и копирует ее в новую ветку. Если ветка уже существует, ее необходимо удалить. Система контроля версий обеспечит возможность восстановления удаленной ветки при необходимости. В результате возникает несколько копий репозитория, каждая из которых может изменяться независимо от другой. Так же возможна операция переноса всех или части изменений из одной ветки в другую. Ветки обеспечивают поддержку ситуации, когда продукту нужно одновременно несколько состояний - например, текущее состояние основной разработки, состояние в среде промышленной эксплуатации, состояние разработки отдельной задачи, которая не должна влиять на основную разработку и т.п.

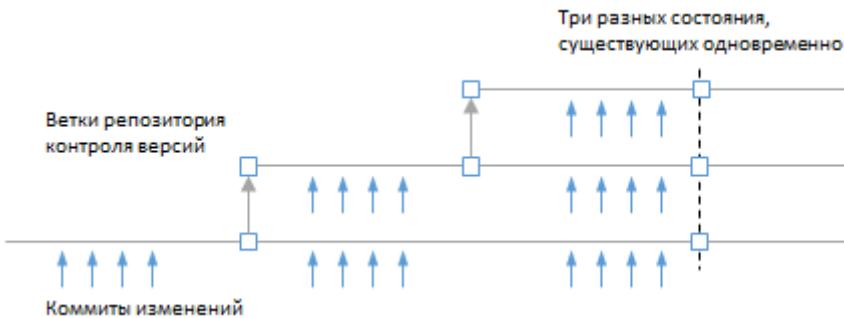


Рис. 56. Ветки и состояния в системе контроля версий

На ветке можно фиксировать текущее состояние в виде тега - неизменяемого объекта, который соответствует состоянию данной ветки репозитория на определенный момент времени. Тег также можно скопировать в новую ветку.

Методология UM рекомендует следующую систему веток и тегов:

Для процесса с тремя видами релизов - основными плановыми (например, раз в квартал), текущими плановыми (например, раз в неделю) и срочными (по необходимости):

- Ветка основного потока разработки: master в git, trunk в svn, из этой ветки никогда релиз не делается
- Ветка основного потока релизов: служит для относительно небольших изменений между основными релизами, и в нее копируется (с полной заменой) основной поток разработки перед выпуском основного релиза, только из нее выпускаются плановые релизы. Изменение текущего планового релиза копируется в первую очередь сюда и лишь затем переносится (нетривиальный merge) в основной поток разработки.
- Ветка специальной разработки: при необходимости вести разработку, от которой надо иметь возможность отказаться
- Тег выпуска релиза: состояние кода ветки основного потока релизов, соответствующее сборке выпущенного релиза. Если данный репозиторий не менялся в релизе, то - копия тега предыдущего релиза
- Ветка внепланового релиза: при необходимости выпуска внепланового релиза, копируется из тега нужного выпущенного релиза. После этого изменение копируется в основной поток релизов и затем в основной поток разработки.

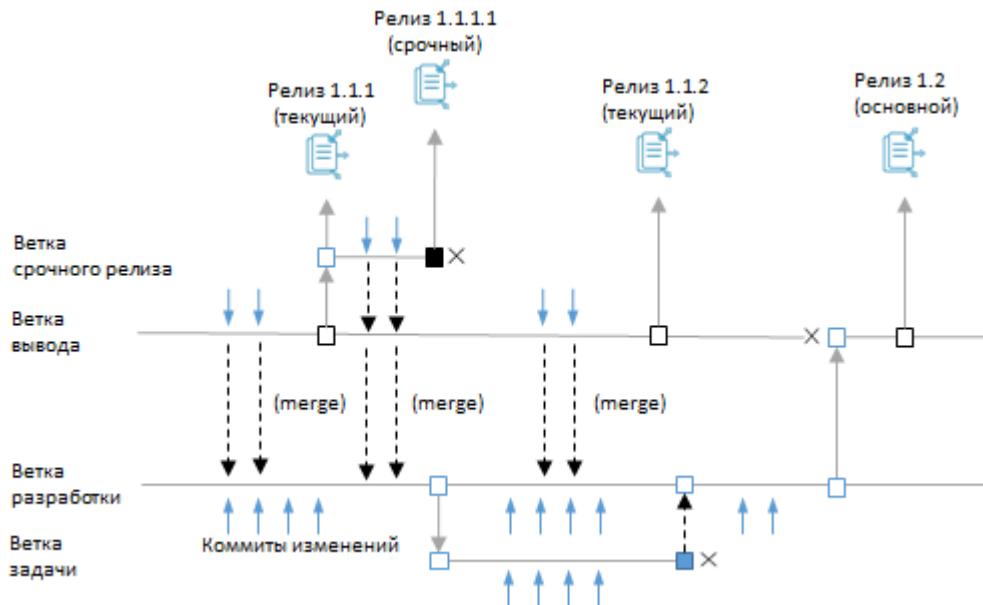


Рис. 57. Поддержка основных, текущих и срочных релизов

Для процесса с двумя видами релизов - текущими плановыми (например, раз в неделю) и срочными (по необходимости):

- Ветка основного потока разработки: master в git, trunk в svn, из этой ветки никогда релиз не делается. Изменение текущего планового релиза копируется в первую очередь сюда и лишь затем переносится (нетривиальный merge) в основной поток релизов.
- Ветка основного потока релизов: служит выпуска релизов, в нее копируется (merge) нужная часть из основного потока разработки перед выпуском соответствующего релиза.
- Ветка специальной разработки: при необходимости вести разработку, от которой надо иметь возможность отказаться
- Тег выпуска релиза: состояние кода ветки основного потока релизов, соответствующее сборке выпущенного релиза. Если данный репозиторий не менялся в релизе, то - копия тега предыдущего релиза
- Ветка внепланового релиза: при необходимости выпуска внепланового релиза, копируется из тега нужного выпущенного релиза. После этого изменение копируется в основной поток разработки и затем в основной поток релизов.

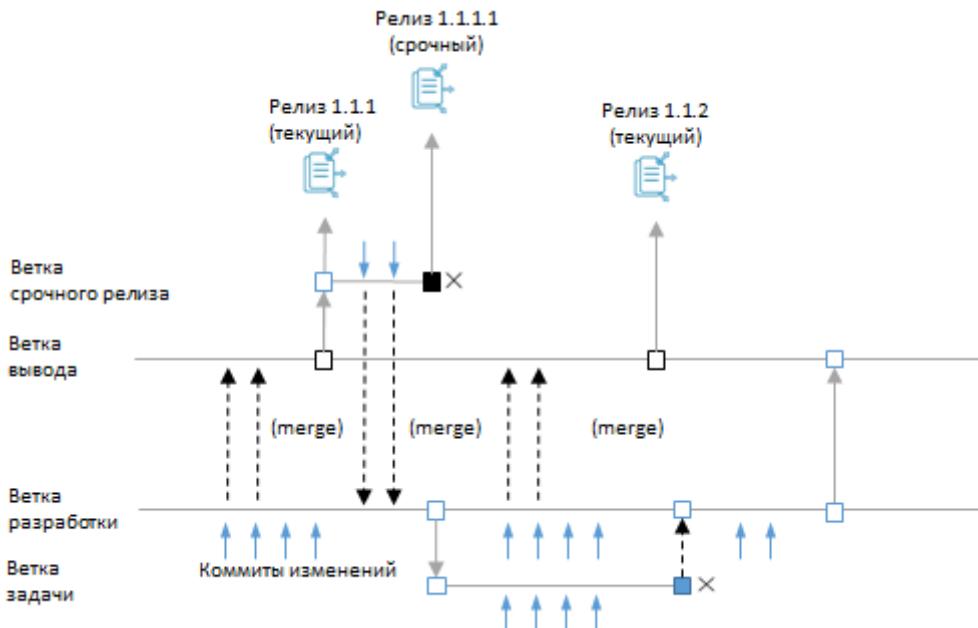


Рис. 58. Поддержка текущих и срочных релизов

Компилируемый и некомпилируемый код

Код делится на исполняемый и неисполнляемый (например, статические картинки или конфигурационные файлы). Исполняемый код делится на машинный, пи-код, компилируемый и интерпретируемый код. Машинный код выполняется процессором и является результатом компиляции компилируемого кода (например, C++), пи-код - также является результатом компиляции компилируемого кода (например, java), но выполняется специальной программой-интерпретатором. Программа-интерпретатор также может исполнять исходный программный код без предварительной компиляции (например, perl, bash).

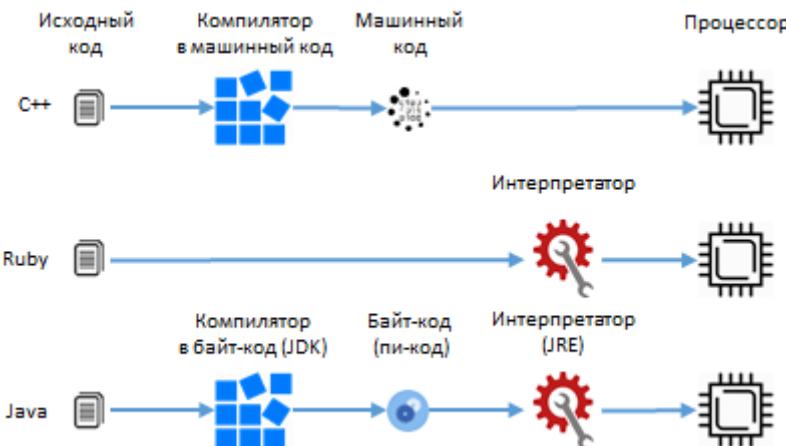


Рис. 59. Варианты исходного кода и его исполнения

Как следствие, есть два варианта:

- **компилируемый код**: код из репозитория исходного кода подвергается компиляции и в дистрибутив не помещается

- **некомпилируемый код:** код из репозитория исходного кода является частью дистрибутива и попадает в среду промышленной эксплуатации

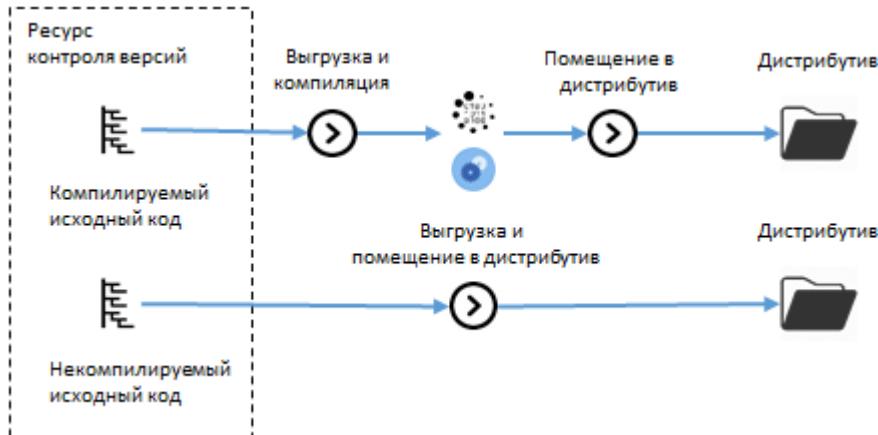


Рис. 60. Компилируемый и некомпилируемый код

Тем не менее даже для некомпилируемого кода как правило есть некоторое преобразование:

- в инкрементальном релизе не весь проект исходного кода может быть взят в дистрибутив
- поскольку это может быть не единственная часть релиза, код помещается в специальную папку дистрибутива
- для упрощения переноса большого количества файлов некомпилируемый код упаковывается в один или несколько архивов
- в случае конфигурационных файлов перед попаданием в среду файл корректируется - в него подставляются параметры конфигурации



Рис. 61. Возможные модификации некомпилируемого кода

Код и библиотеки из сторонних продуктов

Сторонние продукты и их части могут участвовать либо на стадии компиляции, либо на стадии установки дистрибутива в среду промышленной эксплуатации, в виде исходного или уже скомпилированного кода (библиотек). И то и другое для данного программного продукта является исходным кодом (но не собственным). В любом случае, соответствующий исходный код должен быть зафиксирован так, что при возникновении проблемы с релизом было понятно:

- Какой конкретно сторонний исходный код был включен в состав релиза
- Откуда был взят данный исходный код, кем и когда



Рис. 62. Контроль происхождения элементов дистрибутива

Это позволит и оперативно разобраться в возникшей технической проблеме и учесть вопросы безопасности.

Код баз данных

Объекты баз данных делятся на исполняемый код (структурные объекты - пользовательские типы данных, таблицы, хранимые в БД алгоритмы - триггеры, процедуры) и данные. Данные делятся на данные продукта, параметры конфигурации и операционные данные. Данные продукта создаются разработчиками продукта и поставляются в релизах.

Конфигурационные параметры определяются разработчиками, но их значения определяются в рамках администрирования конкретной среды промышленной эксплуатации. Операционные данные являются следствием операций программного продукта, ради которых он существует, определяются пользователями и администраторами.

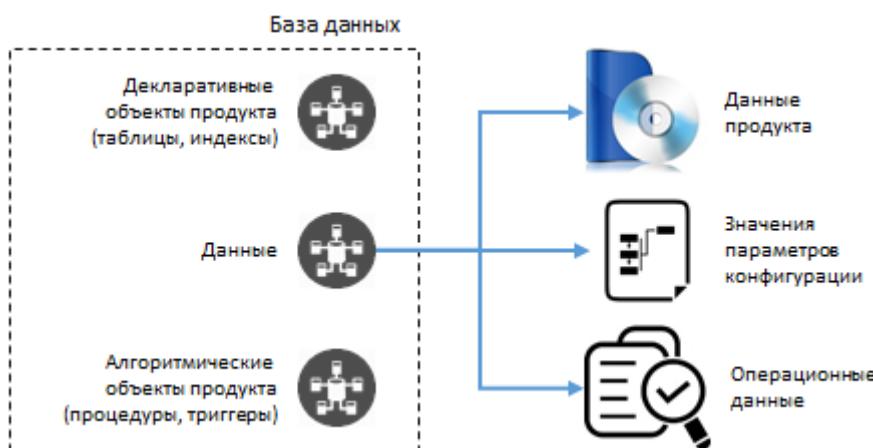


Рис. 63. Содержание баз данных

Базы данных относятся к особому типу элементов прикладных систем - они содержат не только объекты, определяемые разработчиком, но и данные, создаваемые эксплуатируемой программной системой. При этом хотя отделение объектов от данных теоретически и возможно, тем не менее, практически для больших баз данных, находящихся в промышленной эксплуатации, необходимость поменять объект (например, изменение структуры таблицы) может привести к очень сложным и продолжительным действиям со связанными данными. Это называется миграцией.

Другим вариантом миграции является принятие каких-либо решений, относящихся к уже существующим данным - их могут специальным образом поменять, удалить или создать. И это операции изменения, которые применимы исключительно к данному состоянию базы данных, возможно только к одной конкретной версии продукта.



Рис. 64. Особенности изменения баз данных

По этим причинам, изменения базы данных идут именно как операции изменения объектов - на нижнем уровне дистрибутива есть абстрактный скрипт, который как-то изменяет состояние базы данных и в общем случае применим только к той версии продукта, на которую рассчитан. После случайного применения к неверной версии скрипт может создать ситуацию необратимого разрушения данных. С учетом того, что в промышленной базе данных постоянно идут операции, то даже наличие процедуры бэкапов в этом случае не спасает.

Также в релизе зачастую несколько изменений базы данных, рассчитанных на последовательное применение и при внезапных проблемах с одним скриптом, применение всех остальных уже де-факто выполняется в нештатном режиме. Все это порождает необходимость особой строгости при операциях с базой данных, с контролем последовательности и статуса применения.

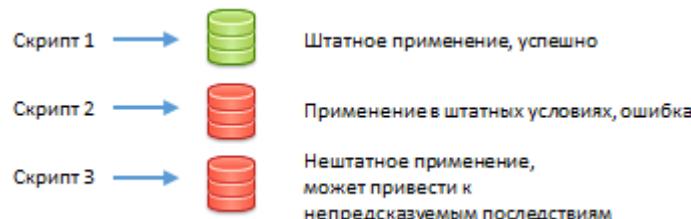


Рис. 65. Особенности применения скриптов в БД

К видам операций с базой данных и как следствие к видам файлов, поставляемых в дистрибутиве относят:

- Применение DML-скриптов - модификация данных без изменения метаданных (структурь объектов и исполняемого кода)
- Применение DDL-скриптов - модификация метаданных и связанное изменение данных
- Загрузка больших объемов данных при помощи специальных операций СУБД

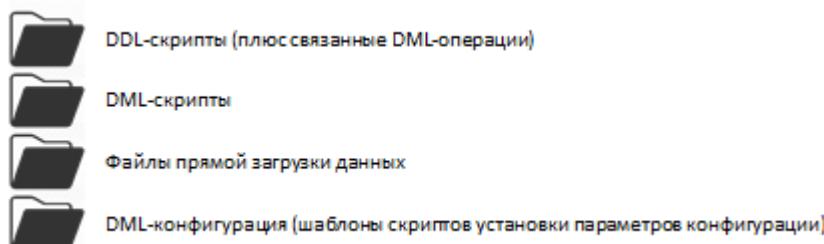


Рис. 66. Разделение изменений в дистрибутиве

Согласно методологии UM, код баз данных должен идти в явном виде отдельно от кода приложений. Код с параметрами конфигурации должен быть в репозитории исходного кода и в дистрибутиве без подставленных значений. Код изменения параметров конфигурации должен идти отдельно от кода изменения настроек, не являющихся параметрами конфигурации.

Конфигурируемый программный код

К конфигурируемому программному коду относятся следующие случаи:

- Программный код перед сборкой требует особой операции конфигурирования под текущее окружение (операционная система, ее настройки) и параметры сборки. К сожалению, параметры сборки как правило означают их скрытие внутри скомпилированного кода и невозможность поменять впоследствии, например, такие свойства как путь к лог-файлам. Такой подход методологией UM не рекомендуется.
- Конфигурационные файлы. Файлы, которые предназначены для хранения настроек и, в том числе, параметров конфигурации.
- Программный код в процессе работы или при установке, не являющийся конфигурационными файлами. Процессорные архитектуры специально проектировали, чтобы код не мог порождать код. Это плохая практика, свойственная вирусам и в общем случае мешающая отладке. Такой подход методологией UM не рекомендуется.
- Настройки баз данных. Особый случай, когда изменение базы данных, которое является DML-операцией (Data Modification Language), содержит параметры конфигурации. В этом смысле данный код по своей роли подобен конфигурационным файлам. Однако по методологии UM необходимо исключить из содержания таких файлом какие-либо иные действия, кроме изменения настроек.

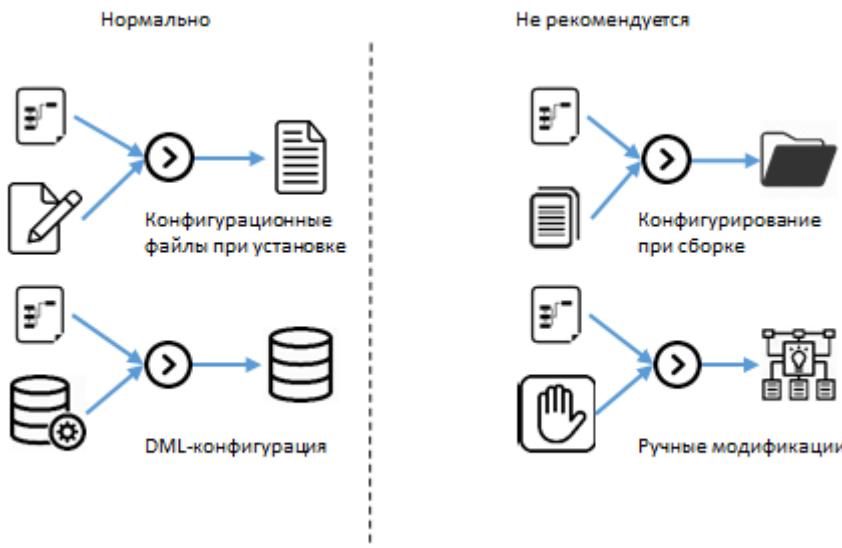


Рис. 67. Рекомендованное и нерекомендованное конфигурирование

Согласно методологии UM, код конфигурационных файлов должен идти в явном виде отдельно от кода приложений. Код с параметрами конфигурации должен быть в репозитории исходного кода и в дистрибутиве без подставленных значений

Сборка программного кода

ЧТО ТАКОЕ СБОРКА И КАКОВА ПРИЧИНА СБОРКИ

ИСХОДНАЯ ИНФОРМАЦИЯ ДЛЯ СБОРКИ

РЕЖИМЫ СБОРКИ

УЧЕТ ЗАВИСИМОСТЕЙ

СРЕДСТВА СБОРКИ

ФРЭЙМВОРК СБОРКИ

НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ

В рамках данной главы описывается что из себя представляет сборка программного кода в программных продуктах, что получается в результате сборки и какие важные особенности приходится учитывать при организации процесса сборки.

Что такое сборка и какова причина сборки

Что такое сборка? Начинающий программист без труда ответит на этот вопрос - это компиляция программы, перевод компилируемого исходного кода в исполняемый программный код. И будет не прав. Это лишь часть ответа, и для интерпретируемого программного кода, работающего без компиляции, понятие сборки тем не менее также существует. Правильные вопросы - "Сборка чего?", "Зачем мы делаем сборку?", "Какие результаты сборки?".



Рис. 68. Принципиальный порядок операций от кода до среды

В инженерном смысле в общем случае цепочка событий примерно ясна - программист пишет код, сохраняет его в репозиторий (или добавляет в репозиторий объекты, которые не являются текстом на языке программирования), этот код от одного или нескольких программистов каким-то образом попадает в процесс сборки, код согласно какой-то инструкции обрабатывается компилятором, затем специальной программой (линковщик и другие), собирается в более сложные объекты (например, библиотеки, исполняемые программы, веб-архивы и т.п.), затем он как-то попадает в дистрибутив, затем оттуда в среду, где этот код работает. Если это конфигурационные файлы с параметрами конфигурации, в среде такой код еще возможно и как-то преобразуется.

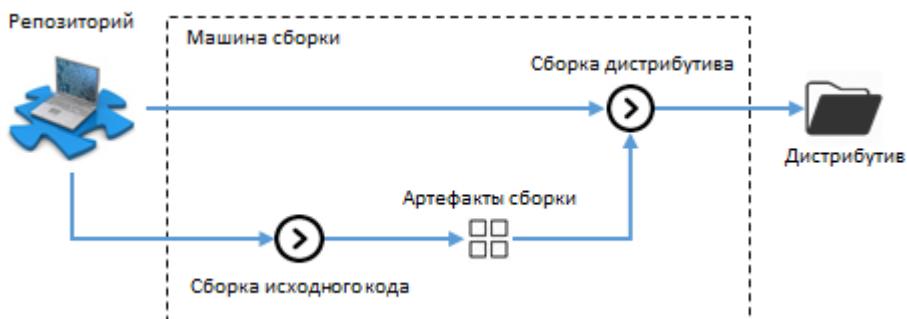


Рис. 69. Сборка исходного кода и сборка дистрибутива

Методология УМ разделяет сборку исходного кода и сборку дистрибутива. И то, и другое могут быть простыми операциями, но могут быть и более сложными организованными действиями - процессами. Сложность процесса сборки определяется тем, что как правило, программистов больше чем один, они используют результаты работы других команд, одновременно для одного и того же участка программного кода может быть несколько состояний - у одного программиста, у другого, в среде промышленной эксплуатации, в текущей плановой разработке, в срочно выпускаемом релизе. Поэтому, чтобы в этих конфигурационных сложностях не запутаться, необходимо правильно организовать процесс сборки.

Чтобы сделать процесс устойчивым и менее зависящим от человеческих ошибок, профессиональная сборка автоматизируется и производится в специальной инфраструктуре - выполняется на "машине сборке".



Рис. 70. Итеративность сборки дистрибутива

С точки зрения формирования дистрибутива - это как правило всегда процесс, поскольку дистрибутив должен быть отчуждаемым элементом, который позволит установить программный продукт (выполнить обновление) силами людей, не являющихся частью команды разработки, и этот продукт должен не содержать критических ошибок. Для первого приходится проверять, не забыли ли что-то положить и описать, а для последнего приходится тестировать и исправлять ошибки. Все это требует итеративного процесса, а не одной операции. Дистрибутив существует в контексте релиза - см. следующую главу.

Сборка делается по одной из следующих причин:

Для того, чтобы получить артефакты для дистрибутива конкретного релиза. Далее дистрибутив может быть установлен:

- в среду тестирования, т.е. это будет промежуточная сборка релиза
- в среду промышленной эксплуатации, т.е. это часть финальной сборки релиза

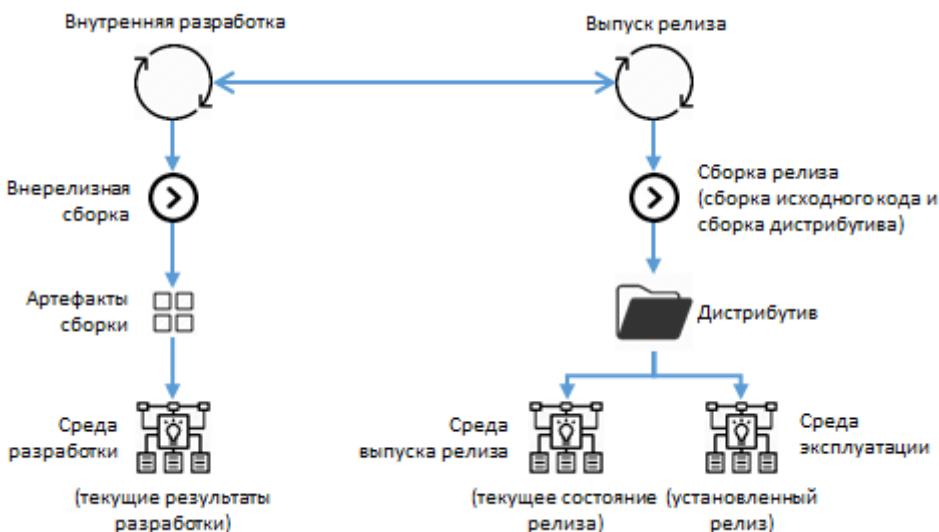


Рис. 71. Причина сборки

Для того, чтобы во внутреннем процессе разработки получить текущее состояние продукта и увидеть все текущие результаты разработки (или какой-то ее части) - текущая сборка программного кода, которая не связана с каким-либо релизом.

Если ограничить сборку операциями до дистрибутива, сборка есть совокупность операций следующих видов:

- установка тегов сборки - обязательная операция по методологии UM, чтобы результат сборки всегда можно было соотнести с точным состоянием кода
- получение исходного кода проекта сборки - выгрузка кода из тега репозитория исходного кода на машину сборки

- получение объектов зависимостей проекта сборки, используемых в сборке, но не являющихся частью исходного кода данного проекта - например, результатов сборки другого проекта исходного кода
- компиляция проекта сборки с созданием промежуточных артефактов при помощи какого-то инструмента разработки
- формирование итоговых артефактов проекта сборки, которые могут быть использованы для сборки другого проекта данного или другого продукта, или как часть дистрибутива данного продукта для установки в среду промышленной эксплуатации



Рис. 72. Сборка зависимых проектов исходного кода

Исходная информация для сборки

В продукте, для которого существует несколько проектов сборки и несколько параллельных потоков разработки, необходимо четко определить, что должно быть передано на вход процессу сборки:

- План сборки.** Это совокупность элементов, которые определяются для сборки до ее выполнения и которые служат для указания целей сборки и оценки того, соответствуют ли текущие фактические результаты сборки плану.
- Режим сборки.** Этот параметр связан с причиной сборки. С учетом нескольких потоков разработки необходимо соблюсти режим песочницы - так, чтобы данная песочница относилась к цели данной сборки и на нее не влияли другие сборки. Песочница в методологии УМ определяется понятием режима сборки, так что все необходимые песочницы можно создать заранее, однократно.
- Исходный код.** Минимальным элементом сборки является сборка одного проекта исходного кода. Данная сборка выполняется соответствующим инструментом, который так или иначе выполняет свою задачу, являясь для методологии УМ черным ящиком. В соответствии с политикой режима сборки код выгружается из некоторой ветки соответствующего репозитория. Инструкция по сборке одного проекта обычно тесно связана с самим исходным кодом и является частью исходного кода проекта сборки.
- Объекты внешних зависимостей.** Объекты могут быть получены в результате сборки других проектов, являться частью исходного программного кода, быть готовыми элементами внешнего по отношению к организации происхождения. Данные объекты могут где-то располагаться на машине сборки или может быть использован специальный процесс (например, nexus), для динамического копирования объектов на машину сборки в процессе сборки - как извне, так и из репозиториев артефактов фреймворка сборки.

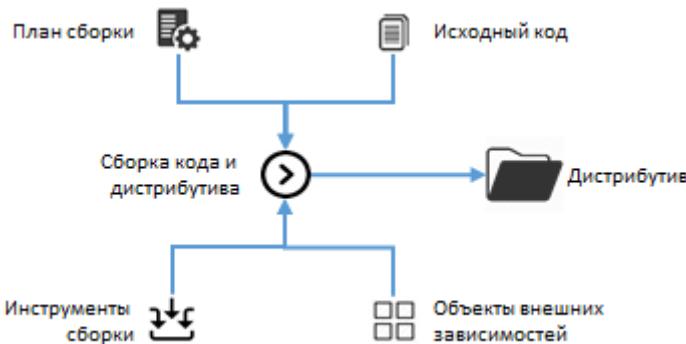


Рис. 73. Что нужно для сборки

План сборки (build scope) необходим в случае инкрементальных релизов для ясного понимания, что и зачем собирается, чтобы обеспечить связь с задачами, ради которых данный релиз запланирован. План должен определить следующие элементы:

- **Состав изменений.** Тикеты из системы управления задачами, изменения для которых и должны быть включены в сборку. Методология УМ требует обязательности указания тикета системы управления задачами в комментарии к коммиту (изменение в системе контроля версий). Комментарий такого вида должен быть обязательным. Сверив список фактических коммитов с планом сборки, можно будет обнаружить нежелательные коммиты и избежать проблем. Это оптимистическая стратегия, альтернативой для которой является контролируемый процесс коммитов и выделенный процесс интеграции, который требует больших издержек и является, соответственно, пессимистической стратегией.
- **Проекты сборки.** Если в продукте несколько проектов, то лучше не производить сборку тех проектов, где изменений быть не должно. Это в том числе скономит время и исключит те изменения, которые не были запланированы. Если проектов много, а изменение для данного релиза требуется только в одном проекте, то нежелательно блокировать другие потоки процесса разработки. Лучше просто не включать другие проекты в сборку и в них можно будет вносить изменения без опасения повлиять на данные результаты сборки. Альтернативой этому подходу являются релизные ветки репозиториев, которые создаются для каждого релиза и таким образом все коммиты относятся именно к релизу. Однако при большом количестве релизов и при множестве репозиториев возникает избыточное количество веток, постоянная необходимость переноса между ветками и возникает риск потерять изменение предыдущего релиза в следующем релизе. Впрочем, для внеплановых релизов стоит применять именно такой подход.
- **Артефакты сборки.** Если заранее известно, что надо обновить, то возможно некоторые артефакты сборки не имеет смысла устанавливать в среду промышленной эксплуатации - и для экономии времени, и для снижения рисков. Т.е. не все результаты сборки входят в план сборки. Например, код сборки одного проекта может порождать два артефакта, но только один из них может быть включен в дистрибутив для установки. Это особенно важно, если артефакты связаны с критическими элементами среды промышленной эксплуатации и их обновление порождает остановку сервиса. Ненужных обновлений лучше избегать.



Рис. 74. План сборки

Исходный код состоит из следующих веток:

- ветка trunk (master) - ветка альфа, место, где задачи постепенно реализуются и формируется отложенный код
- ветка prod - ветка бета, место, где находится код, предназначенный к ближайшему выводу в промышленную эксплуатацию, который подвергается финальному тестированию перед выпуском релиза
- специальная ветка под релиз - аналогично ветке prod, но для случая внепланового релиза

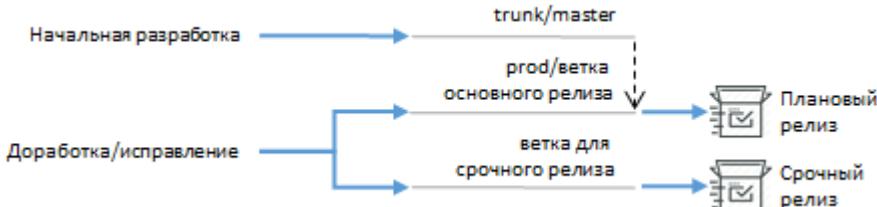


Рис. 75. Ветки исходного кода для сборки

Режимы сборки

Режим сборки является средством эффективного обеспечения состояния песочницы в вопросах сборки при реализации стандартных ситуаций разработки и поддержки программных продуктов. Альтернативами режиму сборки является организация песочницы для каждого из релизов, неконтролируемый выпуск релизов или блокирование всех параллельных операций с существенным снижением пропускной способности системы выпуска релизов. Все альтернативы имеют свои издержки, и в целом получаются менее выгодны. Методология UM определяет следующие режимы сборки:

Разработка альфа. Режим неконтролируемой сборки продукта для целей разработки и отладки, доступна всем разработчикам

- Производится в неконтролируемой среде, позволяющей разработчикам экспериментировать и выполнять минимум операций учета.
- Соответствующая сборка не имеет определенного плана, и как следствие, является либо полной, либо выполняемой вручную, как с машины сборки, так и с рабочего места разработчика.
- Результаты такой сборки не должны быть использованы для каких-либо релизов. При использовании nexus необходимо использовать версию с постфиксом SNAPSHOT и репозиторий артефактов с условным названием "разработка".
- Исходным кодом сборки является ветка "trunk", которая сопоставлена с внутренней разработкой и с которой релиз никогда не выпускается.

- В качестве зависимостей используются объекты этого же режима сборки из данного продукта и объекты "разработка альфа" и "разработка бета" из других продуктов



Рис. 76. Разработка альфа

Разработка бета. Режим неконтролируемой сборки конкретного релиза продукта для целей отладки, доступна всем разработчикам

- Производится в неконтролируемой среде.
- Сборка выполняется согласно плану определенного релиза.
- Результаты такой сборки не должны быть использованы для каких-либо релизов. При использовании nexus необходимо использовать версию без постфикса SNAPSHOT и репозиторий артефактов с условным названием "разработка".
- Исходным кодом сборки является ветка prod или специальная ветка под релиз
- В качестве зависимостей используются объекты этого же режима сборки из данного продукта и объекты репозитория "основной релиз" других продуктов



Рис. 77. Разработка бета

Релиз альфа. Режим предварительной контролируемой сборки всего продукта, доступен релиз-инженеру

- Выполняется в контролируемой среде.
- Сборка производится для всего продукта, без определенного плана.
- Результаты такой сборки не должны быть использованы для каких-либо финализированных релизов. При использовании nexus необходимо использовать версию с постфиксом SNAPSHOT и репозиторий артефактов с условным названием "основные релизы".
- Исходным кодом сборки является ветка "trunk"
- В качестве зависимостей используются объекты этого же режима сборки из данного продукта и объекты репозитория "основной релиз" других продуктов



Рис. 78. Релиз альфа

Релиз минор. Режим сборки минорных и внеплановых релизов, доступен релиз-инженеру

- Выполняется в контролируемой среде.
- Сборка выполняется согласно плану определенного релиза.
- Результаты сборки, после подтверждения тестирования, входят в состав финализированного релиза и устанавливаются в среду промышленной эксплуатации. При использовании nexus

необходимо использовать версию без постфикса SNAPSHOT и репозиторий артефактов с условным названием "текущие релизы".

- Исходным кодом сборки является ветка prod или специальная ветка под релиз
- В качестве зависимостей используются объекты этого же режима сборки из данного продукта и объекты репозитория "основной релиз" других продуктов



Рис. 79. Релиз минор - текущий или срочный релиз

Релиз основной. Режим сборки полного релиза, для выпуска основного релиза и для поддержания зависимостей, доступен релиз-инженеру

- Выполняется в контролируемой среде.
- Сборка производится для всего продукта, без определенного плана для последующего основного релиза или является финальной сборкой мажорных и внеплановых релизов
- Результаты сборки релиза, после подтверждения тестирования, входят в состав финализированного релиза и устанавливаются в среду промышленной эксплуатации. При использовании nexus необходимо использовать версию без постфикса SNAPSHOT и репозиторий артефактов с условным названием "основные релизы".
- Исходным кодом сборки является ветка prod или специальная ветка под релиз
- В качестве зависимостей используются объекты этого же режима сборки из данного продукта и объекты репозитория "основной релиз" других продуктов

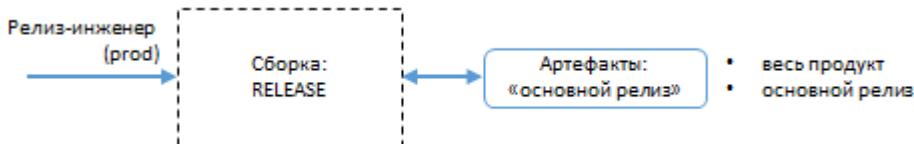


Рис. 80. Релиз основной

Учет зависимостей

В процессе сборки средних и больших продуктов, разросшихся до нескольких проектов, неизбежно возникает вопрос зависимостей. Зависимости бывают внешние и внутренние. Внутренние - на артефакты своего портфеля продуктов, внешние - на элементы продуктов, неподконтрольных текущей организационной единице.

Самый простой способ учить внешние зависимости - явно включить их в состав исходного кода проекта. Это имеет плюсы и минусы:

- Все, что нужно для сборки, находится в исходном коде в репозитории. Ничего нигде больше искать не нужно.
- В отличие от автоматизированных средств учета зависимостей, вы сохраняете в проекте только то, что вам на самом деле нужно. Де-факто размер дистрибутивов при этом существенно меньше.
- Однако, при смене версии продукта, от которого вы зависите, состав требуемых артефактов может существенно измениться, о чем вы узнаете только при полноценном тестировании, или при сбое в промышленной эксплуатации. Т.е. такой вариант возможен, но менять версии зависимых элементов можно только в основном релизе, где полноценное тестирование является обязательной частью. И наоборот, если вам пришлось поменять версию элемента зависимостей, необходимо полноценное (ретрессионное) тестирование.

Внутренние зависимости также можно учитывать путем явного хранения в исходном коде, однако в данном случае это как правило уже становится неприемлемым, поскольку внутренние продукты связаны с конечной функцией и требуют максимальной актуальности.

Если зависимости не хранить в составе кода и не разбираться с ними вручную, что становится важным в современных проектах, где внешних зависимостей очень много и вручную их обрабатывать команды уже не хотят, используются средства автоматизации учета зависимостей, хранящие артефакты, разложенные по версиям - такие инструменты как nexus или nuget, интегрированные со средствами сборки. В инструкции для инструмента сборки (например, maven), идет ссылка на уникальный код артефакта и его версию. В этой же инструкции явно указана версия, под которой результаты сборки данного проекта будут помещены в общий репозиторий артефактов.

Такой репозиторий артефактов имеет следующие плюсы:

- Использование публичных репозиториев позволяет абстрагироваться от команды или компании, которая разрабатывала данный артефакт и получать его единым способом.
- Существует единообразный метод указания версии и привязки к версии артефакта.
- Существует автоматически ведущаяся информация о том, кто и когда этот артефакт выложил, и даже какие зависимости были использованы при сборке данного артефакта.
- Исходный код продукта будет компактен, лишен десятков мегабайтов бинарных файлов.
- При смене версии, подчиненные зависимости более низкого порядка автоматически подтягиваются в проект сборки, и об этом не нужно явно заботиться при сборке данного проекта.



Рис. 81. Варианты учета зависимостей

Тем не менее, как связана версия в репозитории артефактов с версией релиза и номером сборки (каждая сборка уникальна и ей может быть присвоен уникальный порядковый номер)?

Методология UМ определяет версию в репозитории артефактов как версию соответствующего основного релиза, что приводит к следующим свойствам:

- Минорная версия 1.2.1 имеет версию основного релиза 1.2. Поэтому при сборке 1.2.1 артефакты будут выложены под версией 1.2.
- Как следствие, артефакты версии являются обновляемыми. Сборка с зависимостью на 1.2 в один момент подхватит версию 1.2.1, а в другой - версию 1.2.2
- Это означает необходимость обратной совместимости в рамках текущего основного релиза.
- Промежуточные сборки минорных и внеплановых релизов должны собираться в репозиторий "текущие релизы", а зависимости - браться через репозиторий "основные релизы", что позволяет избежать использования нефинализированных зависимостей.
- При минорных и внеплановых релизах изменение версий зависимостей в инструкциях по сборке не требуется - будет использован последний выпущенный релиз.

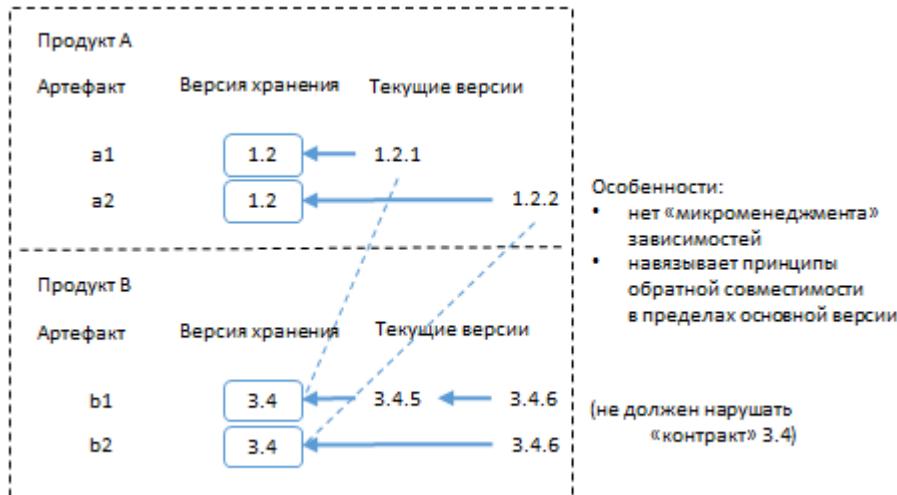


Рис. 82. Механизм реализации зависимостей по основным версиям

Средства сборки

Кроме собственно компиляторов (javac, cpp и т.п.), в типичных процедурах сборки используются и другие специализированные средства. Все средства сборки перечислить невозможно, но можно понять основные подходы на примере C-программ и java-программ:

- **make.** Один из наиболее старых методов сборки. Является оркестратором команд, использует метаязык описания зависимостей и вызывает произвольные команды для формирования элементов. Основной особенностью является сокращение сборки за счет автоматического вычисления необходимой части сборки на основании изменений таймстемпов файлов и хранения рабочих файлов. Однако этот механизм требует множества деталей описания. При выгрузке кода по тегам механизм таймстемпов работать не будет и логика оптимизации оказывается бесполезной.
- **ant.** Являлся и является весьма популярным, поскольку вместо использования явных команд в нем создан метаязык крупных полезных функций, что позволяет компактно описывать почти все практические задачи. Однако управление зависимостями не было реализовано, и чрезмерно разросшийся метаязык также превратился в проблему. Т.е. инструмент не стал способом стандартизации, а является средством удобной автоматизации всевозможных нестандартных решений.
- **maven.** Стал наиболее существенным событием в мире сборки, поскольку стандартизовала как вход, так и выход сборки. Впрочем, он не отвечает на многие вопросы управления кодом и релизным циклом, поэтому удобен как элементарный кирпичик сборки для java-проектов, но не подходит для других языков и как следствие недостаточен для организации сборки продуктов, включающих разнородный гетерогенный код.
- **gradle.** Не очень существенный шаг в сторону от maven, позволяющий инструменту сборки скачиваться в момент сборки. Это решает проблему зависимости от версии инструмента сборки и ее обновления, но это не часто нужно. Методология UM опирается на подход, при котором весь инструментарий сборки явно наблюдаем и контролируем на машине сборке.
- **msbuild:** При развитости графического интерфейса интерфейс командной строки у Microsoft всегда был несколько неудобным и msbuild - не исключение. Неизбежно приходится автоматизировать сборку проектов для технологий Microsoft, и данный инструмент позволяет это сделать. Однако никакими замечательными качествами он не обладает - поэтому довольствуемся тем, что автоматическая сборка в принципе возможна.

В рамках сборки также необходимо отметить дополнительные функции и инструменты, которые являются менее распространенными и в проектах могут отсутствовать:

- **lint:** выполнение анализа кода на соответствие стандартам. В эту же категорию попадают и другие анализаторы, которые могут изучать уязвимости, покрытие unit-тестами, и т.п.

- стилистическое форматирование - принудительное выполнение форматирования кода в соответствии с каким-то стандартом оформления
- автоматическое выполнение unit-тестов, присутствующих в составе кода
- генерация кода - формирование стандартизованных интерфейсов по спецификациям (например, wsdl)
- формирование временной среды продукта в процессе сборки для выполнения более сложных тестов
- **nexus/nuget**: хранение промежуточных и результирующих артефактов сборки

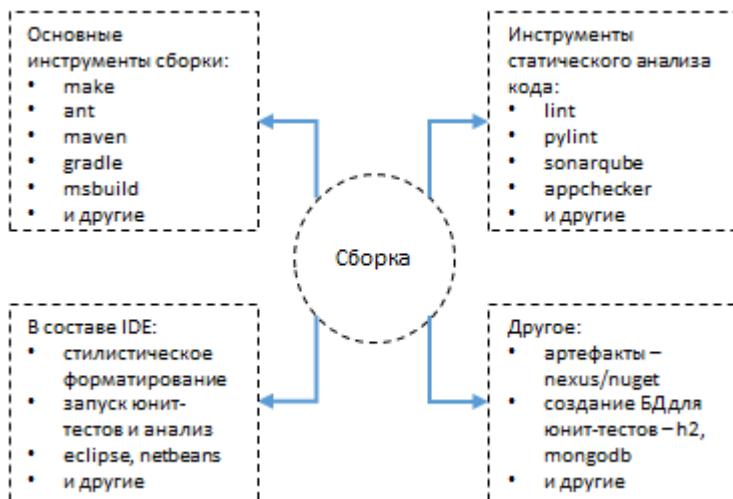


Рис. 83. Инструменты сборки

Фреймворк сборки

Фреймворк сборки - комплекс настроенных элементов инфраструктуры, инструментов сборки, механизмов доступа и правил сборки, который позволяет выполнять необходимые операции сборки нескольким людям без предварительной установки или перенастройки инструментов, т.е. чтобы одна сборка не делала невозможной выполнение другой сборки.

В фреймворк сборки входят:

- **Среда выполнения операций сборки** - то место, где сборка инициируется заинтересованным лицом. В том числе место, где заинтересованное лицо инициирует автоматическую регулярную сборку.
- **Машины сборки** - одна или несколько машин сборки, где выполняется сборка одного проекта исходного кода и где соответственно установлены и настроены инструменты для данной процедуры сборки (jdk, ant, maven и т.п.). Машина сборки может быть выделена для определенной процедуры сборки или может располагаться там же где и среда выполнения операций. На этой машине выполняется выгрузка кода из репозитория, поэтому кроме инструментов сборки, необходимы и клиентские модули нужных систем контроля версий.
- **Серверы контроля версий** - место, где хранится исходный код, и где поддерживается стандартная логика систем контроля версий с использованием веток, тегов и коммитов.
- **Репозитории артефактов** - место, где хранятся артефакты - результаты сборки и внешние готовые элементы. Возможно, это сервер, который обеспечивает доступ к хранящимся где-то еще артефактам. Репозиториев с учетом режимов сборки должно быть несколько.
- **Репозиторий дистрибутивов** - место, где артефакты хранятся в составе дистрибутивов определенных релизов. Предполагается, что планы релизов также находятся в составе дистрибутивов.
- **Логи сборки** - место хранения лог-файлов сборки, которые позволяют понять в чем проблема при неудачной сборке.

- **Средства планирования сборки.** Стандартным способом в развитых проектах является регулярная ежедневная сборка. Для этого нужен инструмент, который позволяет такие операции автоматически запускать.
- **Средства информирования о результатах сборки.** Если сборка происходит автоматически, то важно каким-то образом проинформировать заинтересованных лиц о том, что она произошла и о ее результатах.
- **Средства доступа.** Поскольку сборка может носить критический или закрытый характер, важно наличие механизма разграничение доступа, который позволит только авторизованным людям выполнять операции сборки или смотреть их результаты.



Рис. 84. Фреймворк сборки

Непрерывная интеграция

Под непрерывной интеграцией (continuous integration) понимается организация процесса разработки таким образом, чтобы ошибки, вносимые разработчиком, обнаруживались как можно ранее. Ошибки могут быть следствием неверной логики или неверной координации работ между несколькими разработчиками. Альтернативой непрерывной интеграции является ситуация, когда каждый разработчик работает в своей собственной песочнице и только изредка интегрирует свои результаты в основной код продукта. Непрерывная интеграция подразумевает следующее:

- автоматический перехват коммита в репозиторий исходного кода и выполнение стандартной для данного проекта процедуры:
- автоматическая проверка кода на соответствие стилистическому оформлению
- автоматическая сборка продукта
- автоматическая выполнение некоторого комплекта обязательных юнит-тестов, которые проверяют, не было ли сломано что-то в функциях продукта
- автоматическая переустановка среды разработки из новой сборки
- автоматическая рассылка команде разработчиков сообщений об ошибках при той или иной операции



Рис. 85. Реализация для небольших проектов

Разумеется, все это легко организуется только для небольших продуктов и проектов исходного кода.

Если сборка занимает минуты или часы, а коммиты происходят чаще, если полноценной транзакцией является несколько коммитов, если на среде идет тестирование и не хочется, чтобы она вдруг поломалась, то непрерывная интеграция в вышеприведенном виде нежелательна. Однако это не меняет задач и смысла непрерывной интеграции, которая в данном случае может быть трансформирована в следующий подход:

- каждый разработчик как минимум в конце дня сохраняет результаты своей работы за день в общей ветке репозитория, даже если вся задача не сделана, после завершения отладки сохраняемой части и обеспечив неухудшение свойств продукта (то, что работало, должно продолжать работать, новые функции могут быть недоделаны)
- каждую ночь выполняется сборка и обновляется среда разработки

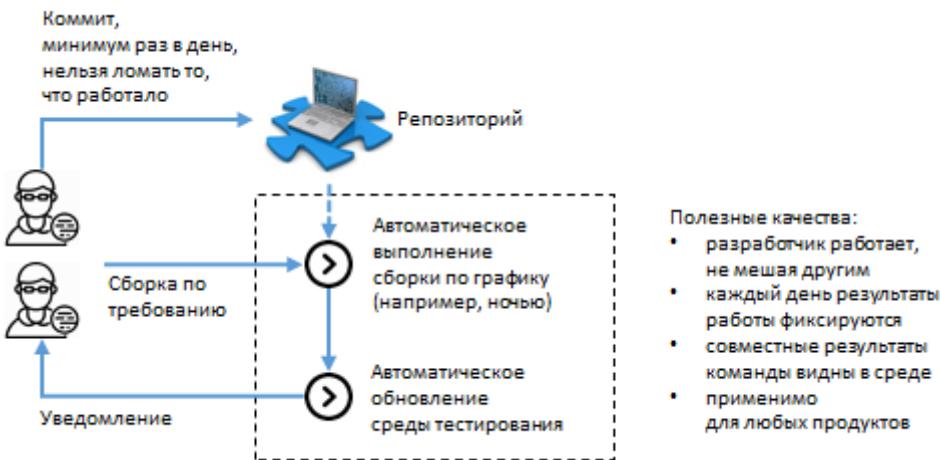


Рис. 86. Практическая реализация для средних и больших проектов

Таким образом ключевое в непрерывной интеграции - работа в команде, постоянное улучшение продукта, возможность постоянно наблюдать общий текущий результат, хотя бы с точностью до дня.

Выпуск релизов

ИЗМЕНЕНИЯ

ПОТОКИ ИЗМЕНЕНИЙ

ПОТОКИ РЕЛИЗОВ

ВИДЫ РЕЛИЗОВ И СЕРИАЛИЗАЦИЯ

ДИСТРИБУТИВЫ РЕЛИЗОВ

СПОСОБЫ ДИСТРИБУЦИИ РЕЛИЗОВ

МОДЕЛИ ФИКСИРОВАННОГО ОБЪЕМА И ФИКСИРОВАННОГО ОКНА

РЕЛИЗНЫЙ ПРОЦЕСС И ПРОЦЕСС УПРАВЛЕНИЯ ЗАДАЧАМИ

РЕЛИЗНЫЙ ПРОЦЕСС И РЕЛИЗНЫЙ ЦИКЛ

ФАЗОВЫЙ КОНТРОЛЬ

ЖИЗНЕННЫЙ ЦИКЛ РЕЛИЗА

УПРАВЛЕНИЕ СОСТАВОМ РЕЛИЗА

ОТМЕНА РЕЛИЗА

НЕПРЕРЫВНАЯ ПОСТАВКА

Откуда берутся релизы? Как они связаны с изменениями в программном продукте и с изменениями в развернутой системе? В рамках данной главы описывается каким образом осуществляется выпуск релизов с технической и организационной точки зрения. Что такое сборка и какова причина сборки

Изменения

Настоящая жизнь программного продукта начинается только с появлением реальных пользователей, работающих в среде промышленной эксплуатации. Для коробочных персональных продуктов процесс появления такой среды прост - человек покупает или получает дистрибутив продукта, устанавливает его у себя на компьютере и начинает пользоваться. Правда, о факте установки зачастую ничего не известно поставщику программного продукта, пока пользователь не обратился с конкретным вопросом. Поэтому понятие эксплуатации здесь неявное и основным признаком является именно продажа. Для бесплатных продуктов понять факт использования еще сложнее - при проблемах с установкой дистрибутива или с началом практического использования пользователь зачастую просто отказывается от продукта.

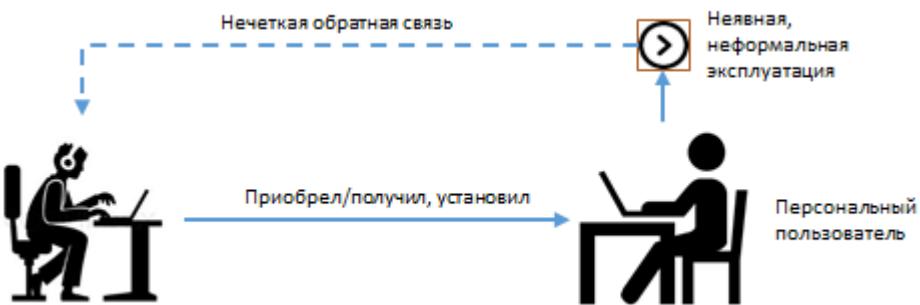


Рис. 87. Эксплуатация персональных продуктов

Для серверных продуктов, о которых в первую очередь идет речь, есть фаза опытной эксплуатации для начального развертывания и ввода в эксплуатацию, которая как правило определена рамочно и содержит итерации доведения до состояния, требуемого для начала полноценной эксплуатации. Но, как уже было сказано выше, это лишь начало и основные события происходят с программным продуктом после начала промышленной эксплуатации. Именно в период промышленной эксплуатации продукт воспринимается не как игрушка, а как реальный и нужный объект. Именно здесь начинается развитие продукта и сопровождение продукта, которые требуют изменений уже полноценно функционирующей среды промышленной эксплуатации. Такие изменения не могут выполняться неконтролируемым способом и в той или иной степени формализуются. Если использование продукта связано с критическими функциями или предоставлением бизнес-услуг, изменения как правило начинают учитывать в какой-то автоматизированной системе. Формализация изменений в фазе опытной эксплуатации имеет смысл только для настройки процесса управления изменениями.

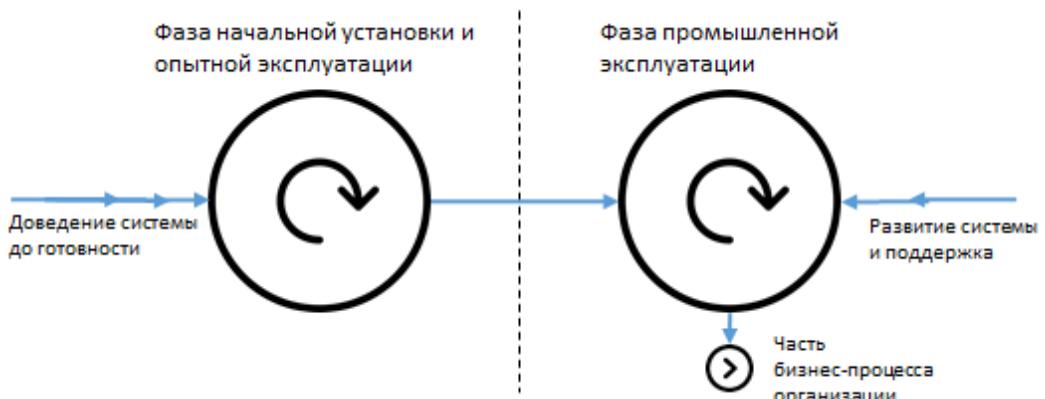


Рис. 88. Эксплуатация корпоративных серверных продуктов

У каждого изменения есть источник происхождения и конечная точка. Все вместе это образует жизненный цикл изменения.

Необходимо различать изменение продукта и изменение среды эксплуатации. Последняя, кроме продукта, еще состоит из параметров конфигурации среды эксплуатации, из элементов окружения (операционная система, системное ПО, базовое ПО) и инфраструктуры (сеть, вычислительные ресурсы, используемое оборудование). Источником изменения продукта может быть:

- **пользователь** - обратившись в службу поддержки, инициирует изменение
- **представитель заказчика** - предъявляет новое требование или заявляет претензию о невыполнении уже существующего требования в рамках договорных отношений
- **разработчик** - изменение для планового развития продукта в соответствии с техническим заданием или техническое изменение, связанное с изменением архитектуры и реализации (рефакторинг)
- **сотрудник эксплуатации** - для изменения эксплуатационных свойств продукта



Рис. 89. Инициаторы изменений продукта

Причиной изменения среды эксплуатации может быть:

- **техническое администрирование:**
 - изменение продукта - см. выше
 - изменение окружения или инфраструктуры как часть основного процесса эксплуатации (управление надежностью, производительностью, безопасностью, мониторинг) или по внешним причинам, не связанным с продуктом
 - изменение параметров конфигурации продукта - вследствие изменений в окружении или инфраструктуре
- **прикладное администрирование** - изменение прикладных данных как часть основных операций, обеспечиваемых интерфейсом продукта для администраторов
- **операции пользователей** - изменение прикладных данных как следствие операций конечных пользователей



Рис. 90. Причины изменений среды эксплуатации

Потоки изменений

Предметом изменения является один или несколько конфигурационных элементов продукта или среды эксплуатации. Например, если продукт в дистрибутиве и среде является одним монолитным исполняемым файлом, то конечным предметом любого изменения являются данный файл в дистрибутиве и соответствующий файл в среде. В реальных более сложных продуктах предметом изменения как правило является небольшая часть элементов конфигурации дистрибутива и среды.

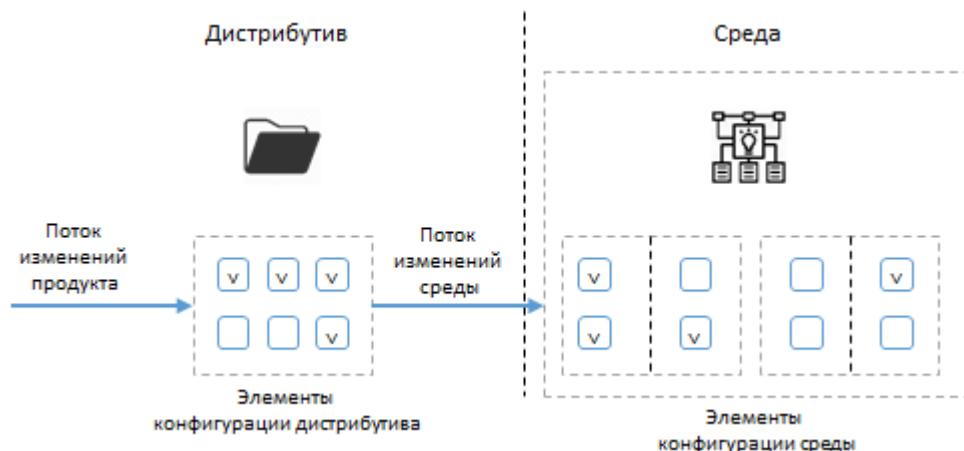


Рис. 91. Частичные изменения конфигурации продукта и среды

Элементы конфигурации могут быть отнесены к разным видам, которые определяют особенности технического администрирования. Как следствие, выделенные виды источников изменений и виды конфигурационных элементов порождают виды изменений. Определенные виды изменений могут выполняться независимо от других видов. Некоторые изменения достигают конечной точки одновременно в виде неразделимой группы, например, в составе одного бинарного файла. В результате изменения или группы изменений, следующие друг за другом, образуют поток изменений. Зависимые виды изменений идут в одном потоке, но в общем случае существует несколько потоков изменений.

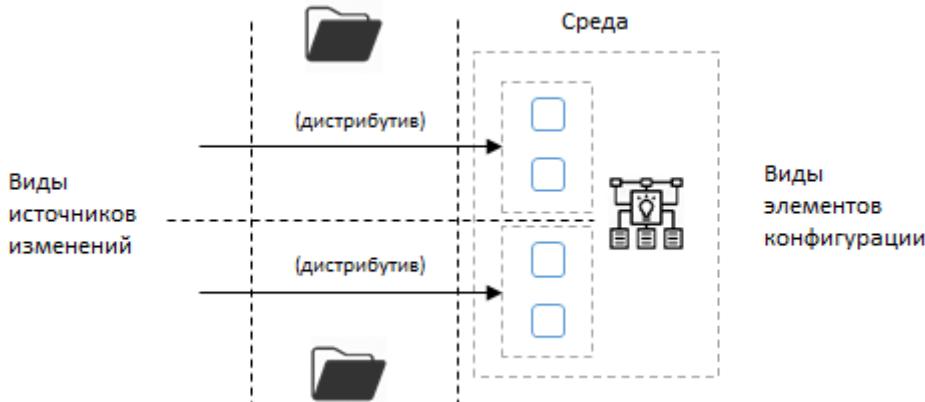


Рис. 92. Виды изменений и раздельные потоки изменений

Если продукт тиражируемый, то жизненный цикл изменения продукта заканчивается официальным выпуском дистрибутива, включающего соответствующее изменение. Потоки изменений тиражируемого продукта заканчиваются в хранилище дистрибутивов. Жизненный цикл изменения среды эксплуатации, вызванного изменением продукта, и соответствующий поток изменений (поток установки обновлений программного продукта) начинается с факта выпуска дистрибутива. Поток изменений прикладного администрирования (контент-менеджмент) никак не связан с изменениями продукта, может начинаться с запросов пользователей и заканчиваться подтверждением решения проблемы со стороны пользователя. Таким образом существует группа потоков изменений продукта и группа потоков изменений каждой конкретной среды промышленной эксплуатации.

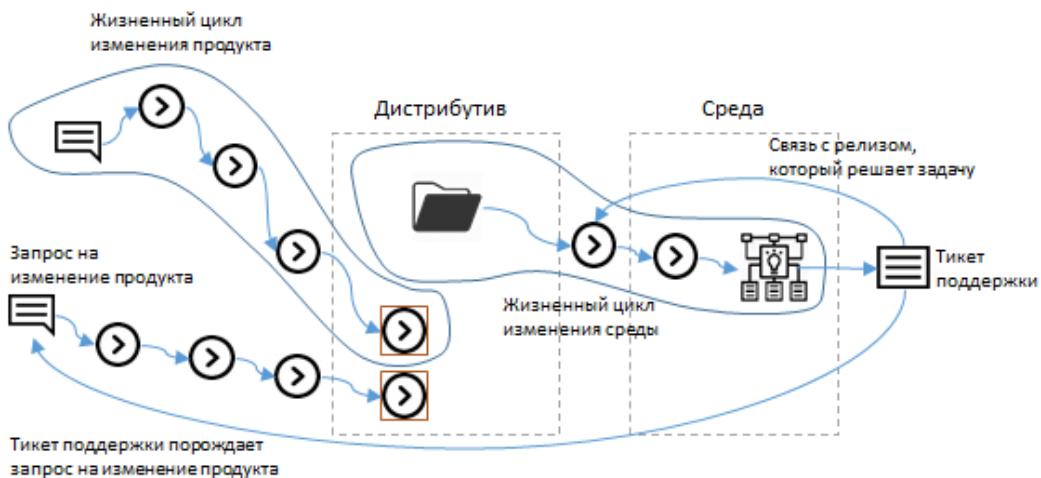


Рис. 93. Жизненные циклы изменений продукта и среды

Если же продукт существует в виде только одной среды промышленной эксплуатации, то жизненный цикл изменения продукта включает жизненный цикл изменения среды промышленной эксплуатации и можно говорить о группе потоков изменений продукта, которая заканчивается в среде эксплуатации и группе потоков изменений среды эксплуатации, не связанных с изменениями продукта.

Весьма существенным фактом в жизни изменения является то, что работы, связанные с изменением, могут оказаться сложнее чем изначально казалось, и изменения в потоке могут меняться местами или вообще отменяться. Тестирование может быть частичным, поскольку полное регрессионное тестирование является весьма затратным, и изменение может оказаться нерабочим, так что изменение решают отбросить, выполненные действия отменить - выполнить другие действия, которые "откатывают" ту часть изменения, которая применилась. Сразу скажем, такой алгоритм "отката" может быть неизвестен из-за непредвиденного состояния или просто не был подготовлен, его приходится изобретать на ходу и

иногда инженеры объективно оказываются в тупике и система остается недоступна весьма продолжительное время.



Рис. 94. Корректировка потока изменений

Потоки релизов

Изменения отражают смысл и суть происходящего в продукте и среде эксплуатации, однако они не являются самостоятельными по следующим причинам:

- Для компилируемого программного кода сборка производится сразу для всего репозитория исходного кода, захватывая области кода, связанные с несколькими изменениями, и порождая общие бинарные файлы. В среде же нельзя обновить только часть бинарного файла.
- Между изменениями может быть зависимость, и тогда они должны быть упорядочены. Задержка или отмена одного изменения таким образом повлечет задержку или отмену других изменений.
- Изменение может быть проработано исходя из определенного состояния среды, к которому оно должно быть применено. Однако к моменту попадания данного изменения в среду промышленной эксплуатации другие изменения могут привести среду к совершенно другому состоянию, и при применении данного изменения результат может быть непредсказуем и отличаться от требуемого.
- Некоторые изменения являются взаимно зависимыми - например, два изменения могут быть допустимы только если сделаны одновременно, и среда будет в нерабочем состоянии после любого одного из этих двух изменений.

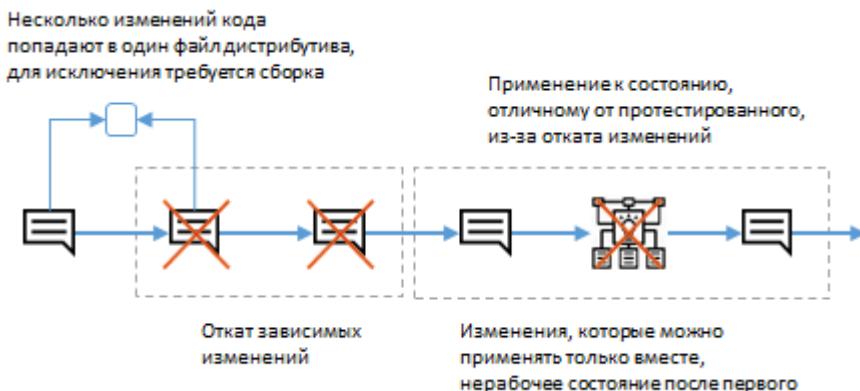


Рис. 95. Влияние зависимостей между изменениями

Также есть соображения эффективности, по которым выполнять даже независимые изменения по отдельности нежелательно:

- Каждое обновление среды промышленной эксплуатации порождает риск неработоспособности, а иногда и требует процедуры прекращения и повторного открытия операций, с перезапуском процессов, их проверкой и другими регламентными действиями. Это порождает накладные расходы на изменение и некоторое время неработоспособности. Несколько изменений,

выполненных как одна операция, могут потребовать практически те же затраты и тот же период недоступности (downtime), что и одно изменение. Таким образом, группировка изменений в общем случае резко снижает затраты и повышает качество сервиса.

- Выполнение обновлений может быть в какое-то время суток нежелательным (рабочие дни, дневные часы) и для проведения обновлений выделяется специальное окно, в которое должны попасть все накопившиеся обновления.
- Изменения зачастую затрагивают код, повторно используемый в разных функциях продукта, и кроме достижения своей цели они могут иметь побочный эффект и сломать те функции, которые не являлись предметом изменения. Это означает, что при любом обновлении системы требуется так называемое регрессионное тестирование - т.е. проверка не только самого изменения, а также функций, которые исходя из затрагиваемого кода могут перестать работать. Предсказать точно область тестирования сложно, особенно для тестировщиков, которые не являются авторами кода и не знают его структуры и особенностей. Все это приводит к относительно обширному регрессионному тестированию. Если изменения сгруппированы, то количество разных целевых конфигурационных состояний в среде эксплуатации становится в разы меньше и затраты на тестирование также резко падают.

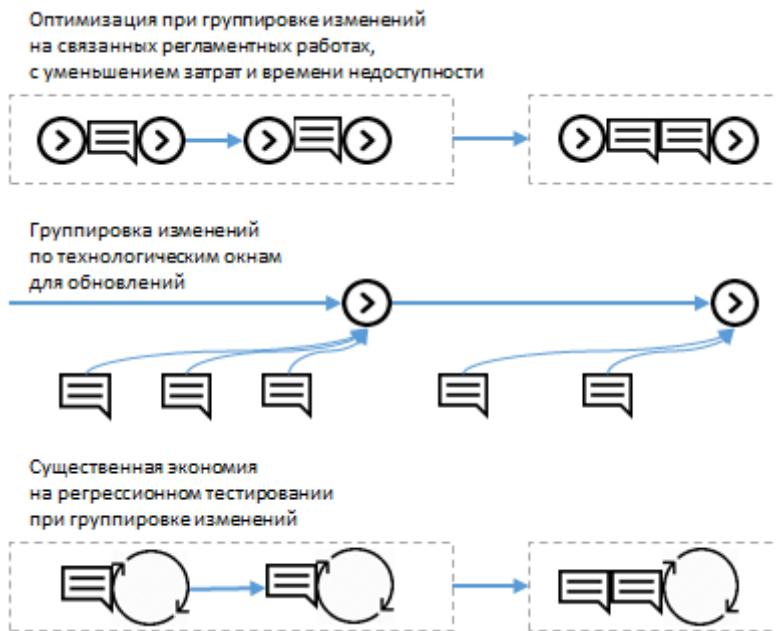


Рис. 96. Оптимизация процесса при группировке изменений

Таким образом и возникает понятие релиза - группы изменений продукта, для которой совместно проводится тестирование и обновление среды промышленной эксплуатации. Как следствие, эти изменения также совместно планируются, реализуются и документируются.

В принципе, если не рассматривать вопросы тестирования и обновления, каждое изменение по отдельности существенно проще и с точки зрения рисков выгоднее делать их каждое отдельно. Именно по этой причине команды, которые не имеют поставленного процесса тестирования, и не очень серьезно подходят к качеству продукта и сервиса, предпочитают не использовать больших релизов.

Понятие релизов относится к изменению продукта, поскольку именно для программного кода характерны все те причины, которые делают релизы важными. Применяя технологию релизов, мы сводим потоки изменений продукта к потокам релизов. В результате изменение среды промышленной эксплуатации производится группой потоков релизов и группой потоков внедрительных изменений.

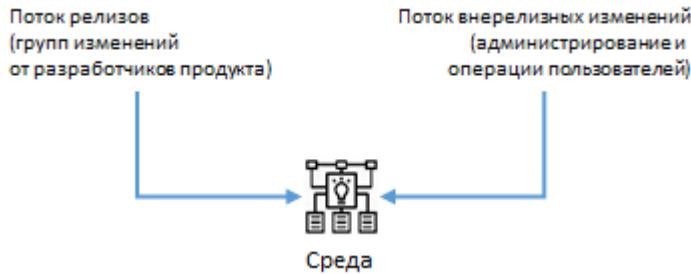


Рис. 97. Потоки релизных и внорелизных изменений

Виды релизов и сериализация

Если группировка так выгодна, что же мешает объединять изменения в очень большие группы? Есть несколько причин, по которым релизы зачастую не настолько большие, насколько могли бы быть:

- Изменения появляются постоянно и рано или поздно надо прекратить копить изменения для текущего релиза и все-таки выпустить его.
- Изменение приобретает свою ценность только будучи выполненным в среде промышленной эксплуатации и чем дольше мы держим его, не выпуская, тем меньшее время оно доступно в работе (time-to-market)
- Изменение имеет такой атрибут как срочность, которая может быть высокой или из-за явного указания заказчика или из-за того, что изменение должно исправить существенную ошибку в среде промышленной эксплуатации. Такое изменение не может ждать.
- Группа изменений может соответствовать некоторой архитектурной переделке продукта и чтобы не запутаться, их производят отдельно от обычных, неархитектурных изменений

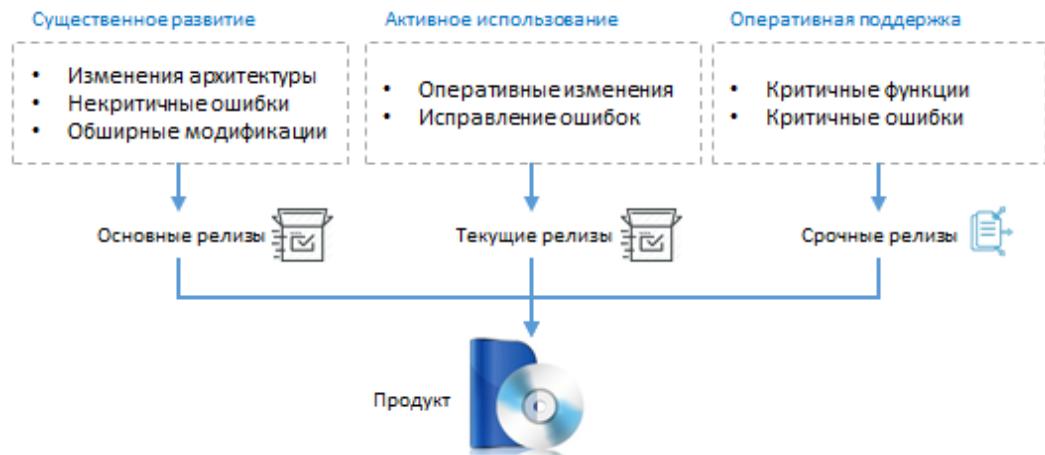


Рис. 98. Виды релизов

По этим причинам появляются следующие виды релизов:

- Основные (мажорные) релизы - отражают развитие продукта, соответствуют архитектурным изменениям, могут также включать и другие виды изменений, в основном исправления не очень существенных ошибок.
- Текущие (минорные) релизы - отражают необходимость более менее оперативно выпускать изменения, которые не являются рискованными и из-за природы которых регрессионное тестирование не нужно или может быть существенно ограничено.
- Срочные релизы - для срочных изменений, связанных с выпуском критически важных функций или исправлением критических ошибок в среде промышленной эксплуатации

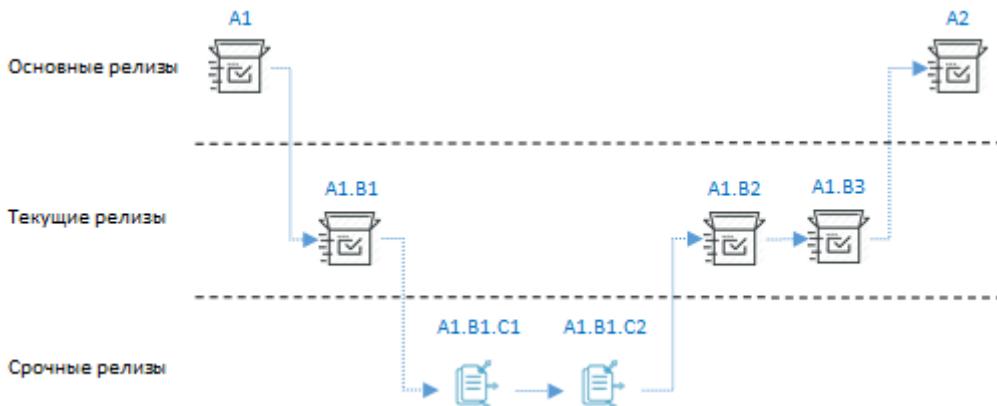


Рис. 99. Нумерация релизов для контроля сериализации

Все указанные выше виды релизов являются присущими любому normally развивающемуся и используемому продукту. Отсутствие основных релизов отражает тот факт, что развитие продукта практически остановилось. Отсутствие текущих релизов означает, что продукт практически не используется. Отсутствие срочных релизов может быть связано как с очень высоким уровнем выходного контроля (что означает в среднем повышенные затраты на разработку и тестирование), так и со спецификой продукта, если его обновление не может быть выполнено оперативно или использование не носит постоянного характера.

Указанные выше виды релизов формируют разные потоки релизов - в то же время, когда разрабатывается и выпускается основной релиз, выпускаются и текущие релизы. До выпуска текущего релиза, может возникнуть необходимость выпустить срочный релиз. Чтобы учесть зависимости между изменениями в составе релизов и дать основу для тестирования релизов, выпуск релизов в разных потоках сериализуется. Для этого применяется система версий:

- Основной релиз выпускается с версией А.
- Текущие релизы после выпуска основного релиза и до выпуска следующего основного релиза выпускаются с версией А.В
- Срочные релизы после выпуска текущего релиза и до выпуска следующего текущего релиза выпускаются с версией А.В.С

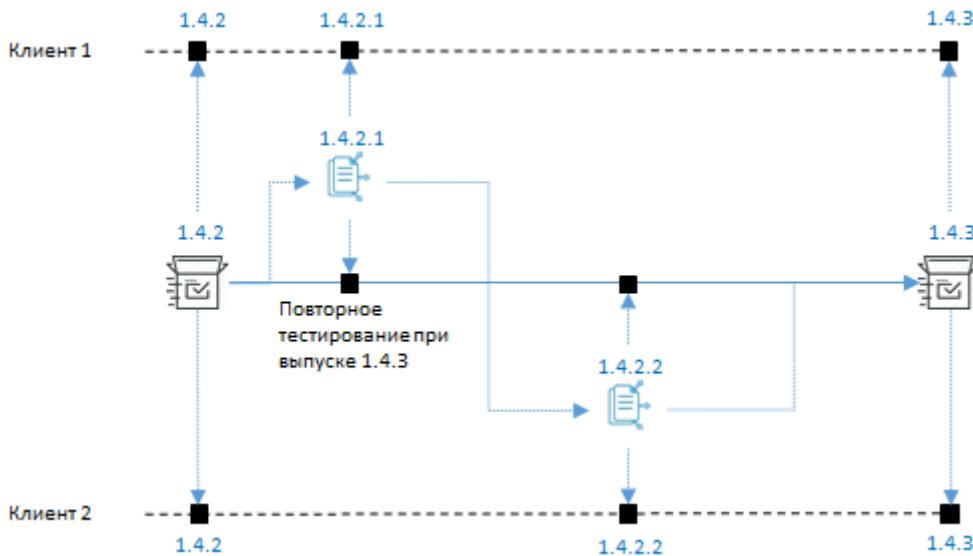


Рис. 100. Обеспечение сериализации при выпуске тиражируемых продуктов

Выпуск и установка релизов производится в соответствующем порядке. Таким образом, несмотря на параллельность трех потоков релизов, существует строгая последовательность состояния кода и состояний среды промышленной эксплуатации, в соответствии с номерами версий, и известно, какое исходное состояние будет при установке данного релиза и какая версия находится в среде промышленной эксплуатации в каждый момент времени. Чтобы изменения не терялись, выполняется процедура слияния (merge) кода из срочного релиза в текущий и из текущего в основной.

Точно такой же метод необходимо применять для тиражируемых продуктов. Т.е. если есть два клиента установили версию 1.4.2, потом пока разработчики делали версию 1.4.3, понадобилось выпустить внеплановое изменение для клиента 1. Это изменение должно быть выпущено на основании тегов версии 1.4.2 в дополнительной ветке и иметь версию 1.4.2.1. После выпуска 1.4.2.1 данное изменение должно быть внесено в релиз 1.4.3. Поскольку по сути это уже другой код из другой ветки, то данное изменение из 1.4.2.1 должно быть внесено в список изменений 1.4.3 и повторно протестировано. Это плата за срочность релиза. Нельзя пытаться сэкономить и, например, не включать изменение в 1.4.3, планируя внести его когда-нибудь, поскольку это означает непригодность 1.4.3 для клиента 2. Также если возникнет внеплановый релиз для клиента 2, то его надо делать с версией 1.4.2.2 на основе тега 1.4.2.1 также с внесением изменений в релиз 1.4.3. Нельзя в общем случае делать релиз 1.4.2.2 на основе релиза 1.4.2. Это поломает сериализацию релизов и создаст возможность ошибки из-за того, что важные изменения 1.4.2.1 не попадут клиенту 2 и исправление будет строиться не на основе 1.4.2.1, а также при слиянии кода 1.4.2.2 в 1.4.3 будет потенциальный конфликт, поскольку в 1.4.3 код базируется как раз на 1.4.2.1.

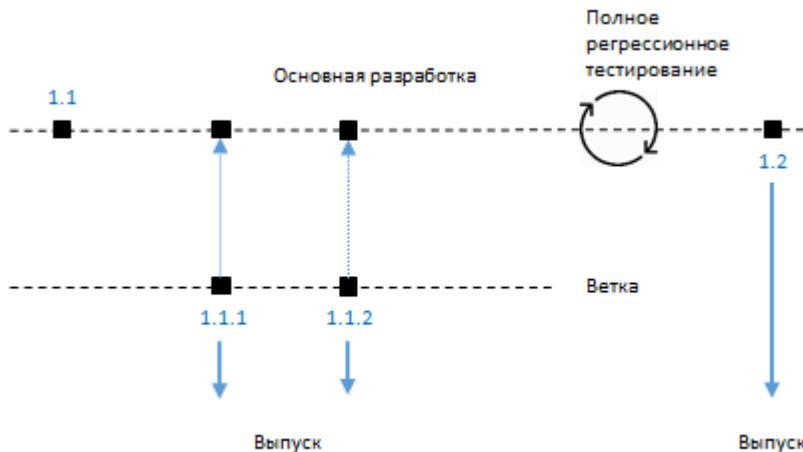


Рис. 101. Обеспечение сериализации при параллельной разработке релизов

При выпуске текущих релизов аналогичная ситуация возникает при внесении всех изменений текущих релизов в основной релиз. Однако основной релиз по определению предполагает полноценное регрессионное тестирование, в рамках которого тестируется уже не изменение как таковое, а состояние - полный комплект функций продукта. Т.е. если функцию несколько раз меняли в текущих релизах, то при регрессионном тестировании функция будет протестирована на соответствие последнему требуемому состоянию. Поскольку при архитектурных изменениях все-равно нужно проводить полноценное тестирование, необходимый объем тестирования остается минимально возможным, оптимальным.

Дистрибутивы релизов

Релизы - это группы изменений продукта, которые необходимо запланировать, протестировать и выполнить в среде промышленной эксплуатации. Но что фактически представляют собой эти изменения?



Рис. 102. Объекты, затрагиваемые релизом

Продукт в среде представлен исполняемыми файлами, конфигурационными файлами, базой данных и операционными данными в файлах. У среды промышленной эксплуатации есть определенная конфигурация, которая в данном релизе может поменяться (например, может добавиться новый процесс обработки). Также нельзя забывать, что продукт требует определенного окружения и эти требования могут меняться от релиза к релизу. Исполняемыми файлами считаются файлы любого формата, не подлежащими изменению по сравнению с дистрибутивом. Операционные данные в файлах - всевозможные лог-файлы, файлы с созданными в процессе работы данными. База данных - содержит программный код (хранимые процедуры и т.п.), данные продукта (например, предопределенные справочники), параметры конфигурации (например, хранящиеся в таблице настроек) и операционные данные, которые появляются в процессе выполнения действий администраторами и пользователями. Изменения релиза таким образом состоят из:

- установки новых версий всех или части исполняемых файлов
- изменений конфигурационных файлов в фиксированной их части
- изменений параметров конфигурации в файлах и базах данных
- изменений окружения - смены версии базового ПО, установка и переустановка системного ПО
- изменений конфигурации среды - установки или остановки процессов, перемещения файлов (приложений) из одних процессов в другие, добавления или удаления исполняемых и конфигурационных файлов
- изменений программного кода базы данных и данных продукта
- модификации уже накопленных операционных данных, чтобы они соответствовали новым форматам и логике обработки, которые привносятся данным релизом



Рис. 103. Идентичность изменений в разных средах

Чтобы выполнить эти изменения, нужно точно определить эти изменения. Ведь их нужно протестировать в какой-то тестовой среде. Если изменения будут недоопределены, тогда в среде тестирования будут выполняться одни изменения, а в среде промышленной эксплуатации - другие. В результате тестироваться будут не те изменения, которые будут делаться в конечном счете и результат тестирования не будет иметь смысла, т.е. релиз фактически не будет протестирован.



Рис. 104. Повышение декларативности дистрибутива

Видно, что в списке только часть изменений носят характер конечных файлов, которые можно просто взять и положить в нужное место. Остальные – это модификации той или иной сложности. Часть этих модификаций – описывается как программа на каком-то языке. Например, скрипт PL-SQL для модификации данных или shell-скрипт для исправления конфигурационного файла. Но часть – зависит от окружения и не может полностью определяться в продукте. Например, смена версии базового ПО – соответствующий дистрибутив новой версии базового ПО поставляется не в релизе продукта и далеко не всегда он может быть автоматически найден в процессе установки. В релизе в таком случае идет инструкция для человека и от того насколько одинаково выполнены такие операции в тестовой среде и среде промышленной эксплуатации, также зависит корректность тестирования и успешность при выполнении изменений. Кроме того, даже в простых случаях, несмотря на возможность автоматизации, вместо программы часто предоставляется инструкция для человека. Это связано с тем, что такая программа оиять же зачастую не проходит тестирования, а выполняемая на уровне системного администрирования, она может принести очень много проблем. Правильным решением этой дилеммы является использование универсального алгоритма для конфигурационных изменений, качество которого не зависит от релиза.

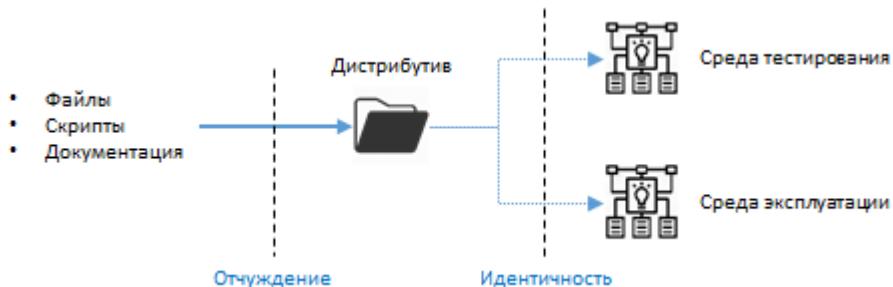


Рис. 105. Ключевые задачи операций с дистрибутивами

Файлы, скрипты, инструкции, документация, описание изменений полностью описывают конкретный пакет изменений. Будучи сложенными в выделенном месте, они образуют то, что называется дистрибутивом. Дистрибутив обеспечивает отчуждение пакета изменений от авторов – то, что делает возможным упаковать пакет в один архив и так или иначе передать сотрудникам эксплуатации той или иной среды промышленной эксплуатации. Также именно дистрибутив является тем объектом, который применяется то в тестовую среду, то в конечную среду и это делает процесс понятным и прозрачной с инженерной точки зрения. Полностью автоматизированный дистрибутив устраняет ошибки, связанные с конфигурационным управлением. А по статистике половина ошибок связана именно с конфигурационными несоответствиями.

Способы дистрибуции релизов

При всем том, что для персональных продуктов и односерверных продуктов дистрибутив является весьма распространенной и понятной всем формой передачи продукта в эксплуатацию, для продуктов на основе открытого кода и более сложных продуктов это уже совсем не такой уж типичный способ, поскольку отчуждение для сложной конфигурации требует специальных решений по управлению конфигурацией. Известными вариантами передачи релиза в среду эксплуатации являются:

- дистрибутив в форме папки с файлами, без средств установки, но с инструкцией (например, один war-файл, который является архивом и содержит все нужные файлы в составе приложения)
- дистрибутив в форме папки с файлами, включающий средства установки (например, исполняемый файл, полученный при помощи InstallShield)
- дистрибутив в форме папки с файлами, в формате определенного средства установки (например, пакет в формате msi, который будет выполнен имеющимся в составе Windows сервисом установки программ)
- инструкция по сборке и установке из кода на основе доступного репозитория кода (make configure, make install)
- дистрибутив-репозиторий, содержащий разнообразные артефакты, скрипты для операций и метаописания целевого состояния для хостов определенного типа (мастер-репозиторий Puppet)
- дистрибутив-копия сервера, в котором файлы предустановлены и представляют собой готовую к исполнению конфигурацию (контейнер docker)
- без дистрибутива вообще - скрипт обновления является частью среды и по команде обновляет ее напрямую из репозитория программного кода

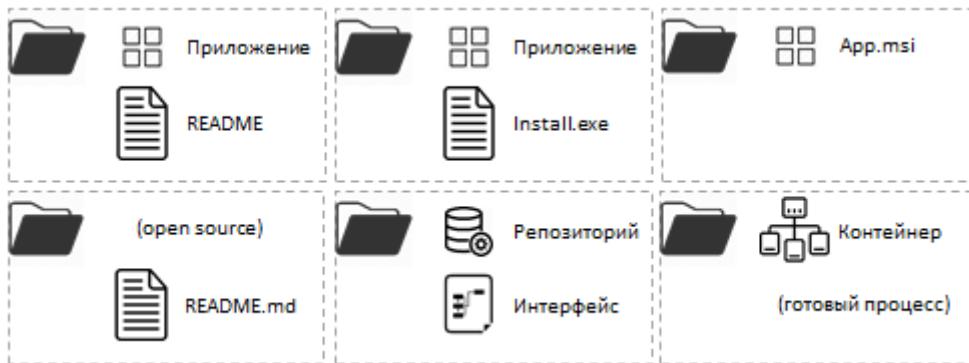


Рис. 106. Содержимое дистрибутива

В разных условиях все эти варианты имеют право на существование и де-факто все эти варианты имеют широкое распространение, но выбирая способ дистрибуции для более-менее сложных серверных продуктов, использующихся для выполнения важных функций в среде промышленной эксплуатации, необходимо учитывать следующее:

дистрибутив должен быть наблюдаем и отчуждаем:

- именно выпущенный дистрибутив отражает результаты работ по разработке и представляет собой определенную ценность, без него процесс выпуска релизов становится неконтролируемым как с точки зрения управления разработкой, так и с точки зрения управления изменениями в среде промышленной эксплуатации
- без дистрибутива среда промышленной эксплуатации фактически является частью инфраструктуры разработки, что может противоречить границам ответственности и требованиям безопасности

Явный дистрибутив:

- идентифицированная версия определенного продукта
- наблюдаемый и отчужденный



Рис. 107. Явный дистрибутив

принцип устройства дистрибутива должен позволять как начальное развертывание конкретного сервера, так и инкрементальное обновление

- например, мастер-репозиторий Puppet - умеет как устанавливать, так и обновлять сервер, однако сам репозиторий не допускает инкрементального содержания и не может распространяться как инкрементальный релиз
- при отсутствии поддержки инкрементальных дистрибутивов неизбежные текущие и срочные релизы будут слишком тяжеловесны, чтобы их выпускать явно и процесс обновления будет неконтролируемым

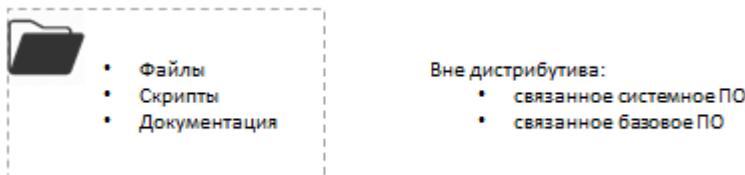


Рис. 108. Объекты вне дистрибутива продукта

дистрибутив должен решать вопросы всех элементов продукта, которые регулярно меняются

- например, конфигурационные файлы и структура базы данных являются частым объектом изменений и в формате дистрибутива для этих типов файлов должно быть отведено свое место
- однако системное и базовое ПО и его параметры либо устанавливаются однократно, или меняются сравнительно редко - поэтому данный пункт не распространяется на них

дистрибутив должен решать вопросы конфигурирования сред - одни и те же файлы из дистрибутива должны работать и в тестовой среде при выпуске дистрибутива, и в среде промышленной эксплуатации

- встречались случаи, когда параметры конфигурации (тестовая/промышленная) определялись при компиляции и было два разных дистрибутива одного релиза - для тестирования и для конечной установки. Как следствие, файлы из последнего ни разу не проверялись, что разумеется, делало невозможным вывод об их статусе
- встречались случаи, когда конфигурационные файлы для конкретных тестовой и промышленной сред эксплуатации были по-отдельности в дистрибутиве - что делает невозможным развертывание такого дистрибутива на дополнительной тестовой среде (например, для демонстрации) и также порождает риск того, что изменения проведены и протестированы в тестовом комплекте, но были упущены в комплекте для среды промышленной эксплуатации



Рис. 109. Конфигурирование объектов релиза для сред

дистрибутив по возможности не должен содержать каких-либо скриптов развертывания, созданных специально для данного релиза

- скрипты такого рода выполняются с полномочиями, способными привести к необратимым последствиям в среде промышленной эксплуатации, а их полноценное тестирование сложно выполнить, поскольку конфигурация тестовой среды и среды промышленной эксплуатации может совершенно неожиданно для автора скрипта отличаться, поскольку автор просто может не иметь доступа к данной среде и опираться на неверные предположения, а тестирование в работающей среде невозможно

Отдельно стоит отметить, что для персональных продуктов целесообразно применять концепцию zero-install, которая состоит в том, чтобы установка выполнялась в одно действие и продукт сам автоматически обновлялся по мере необходимости. Однако механизм обновления при этом все-равно

должен использовать дистрибутивы, размещенные в публичном хранилище. Лучше, если это хранилище доступно для внешнего анализа и не является закрытым форматом.

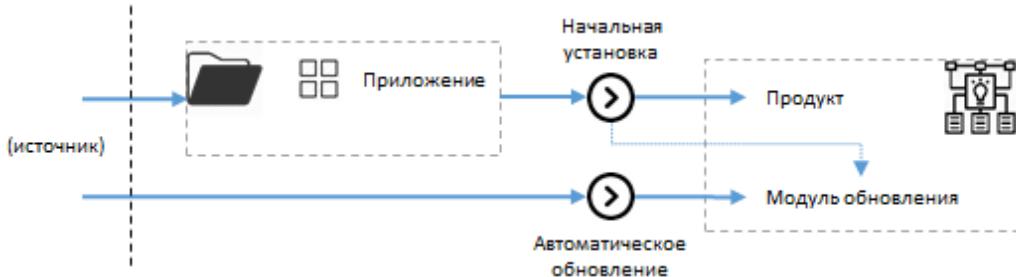


Рис. 110. Встроенное обновление

Модели фиксированного объема и фиксированного окна

В проектном менеджменте хорошо известен треугольник компромиссов - из объема задач, времени и качества выбери только две цели. Третья выполнена не будет. Треугольник компромиссов существует, поскольку всегда происходит что-то, что усложняет выполнение задач и происходит это обычно в ходе выполнения задач. Поэтому увеличить объем четвертого компонента, который отвечает за площадь треугольника - количество людей, которые делают дело, будет сложно - откуда они вдруг возьмутся? А если даже и появятся, то им нужно будет время на то, чтобы войти в тематику проекта и сделать задачу правильно. Хорошо известен полушуточный-полусерьезный закон Мерфи как раз об этом - "добавление людей к проекту только увеличивает срок его выполнения".

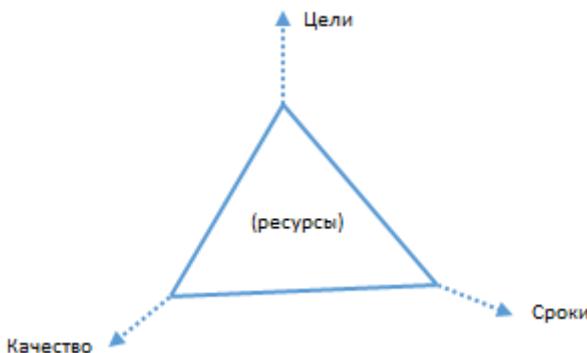


Рис. 111. Треугольник компромиссов

Качество - очень важное свойство, но непростое. Затраты на повышение качества с какого-то момента начинают расти экспоненциально, поэтому качество нельзя делать объектом безусловной максимизации. Качество должно быть разумным. Тем не менее в треугольнике компромиссов качеством релиза жертвовать нельзя, т.е. можно и нужно заранее установить некоторые ограниченные цели по качеству (иногда совсем не амбициозные) как ограничение снизу, но нельзя пускать качество на самотек - как получится. Неработающий продукт и отсутствующий - это почти одно и то же.

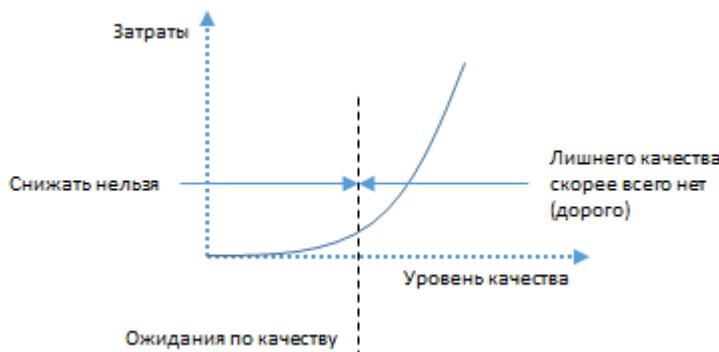


Рис. 112. Уровень качества

Таким образом у нас остается один единственный выбор, чем мы жертвуем - объемом или сроками:

Модель фиксированного объема.

Здесь мы жертвуем сроками ради того, чтобы весь запланированный объем задач в данном релизе выпустить. Это хорошо всем известный метод, который при практическом применении оборачивается множеством минусов:

- Несмотря на желание исходно сделать в релизе конкретный список задач, никому это не удается - задачи появляются, изменяются и это не зависит полностью от менеджера. Даже самого идеологически убежденного менеджера заставляют менять планы его руководители и заказчики. Поэтому выстраивание процесса на отрицании изменений, которые все-равно происходят - приводит лишь к снижению общей эффективности.
- Сроки де-факто сдвигаются по абсолютно всем задачам, поскольку каждый релиз задерживается. При этом это следствие проблем только с отдельными задачами, из-за которых страдают все остальные.
- Поскольку ожидания постоянно нарушаются, ни разработчики, ни эксплуатация, ни заказчик не могут приспособиться к этому де-факто беспорядочному выводу изменений.



Рис. 113. Модель фиксированного объема

Модель фиксированного окна (time window).

В этой модели мы жертвуем объемом ради того, чтобы те задачи, которые готовы, дошли до пользователя в срок, независимо от проблем с другими задачами. Это имеет следующие свойства:

- Хотя большинство задач выходит точно в срок, каждая конкретная задача может быть задержана, причем на больший срок, чем это было бы при модели фиксированного объема
- Как только становится понятно, что задача не успевает в срок релиза, необходимо обеспечить ее вывод из состава релиза. А ведь часть кода уже написана и на него возможно завязаны другие задачи. Это порождает в общем случае не просто исключение задачи из списка, но также и операции по откату кода и исключении из релиза зависимых задач, независимо от степени их готовности. Для исключения рисков изменения должны вноситься в ветку выпуска только после достижения статуса готовности к тестированию - т.е. после решения всех архитектурных проблем и после проверки функции самим разработчиком (отладки)

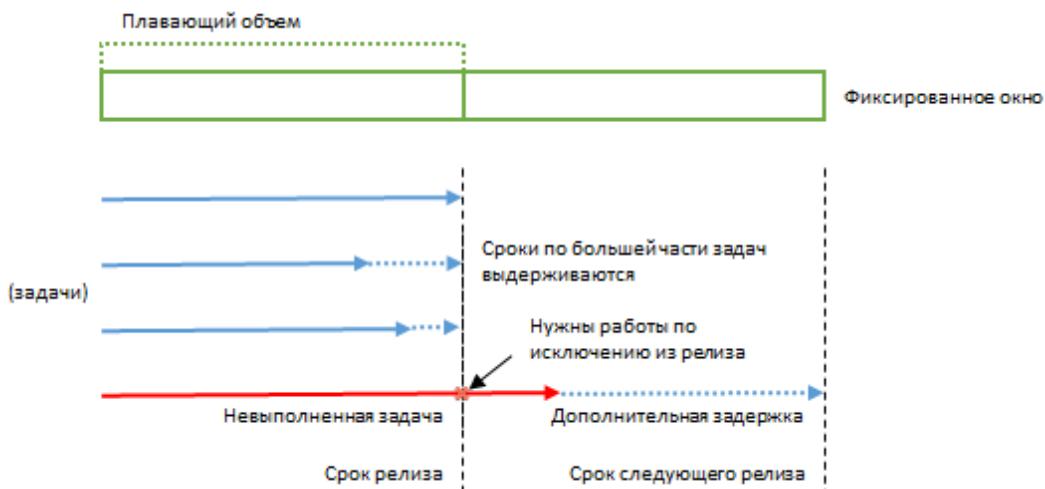


Рис. 114. Модель фиксированного окна

Релизный процесс и процесс управления задачами

Процессы - это то, как четко, рационально и последовательно выполняются наши действия. Процессы служат для того, чтобы одинаковые с какой-то точки зрения вещи выполнять одинаково, если данный порядок выполнения работ (технология) является оптимальным. Такой порядок возникает не сразу и именно в рамках процесса осуществляется типизация задач, поиск и оптимизация алгоритмов выполнения задач, анализ и контроль оптимальности. Целью является выполнение запросов, которые даются на вход процессу, с нужным качеством и с минимальными затратами, быстро и с гарантированным результатом. В зависимости от тематики и областей ответственности, типы задач распределяются по нескольким процессам.

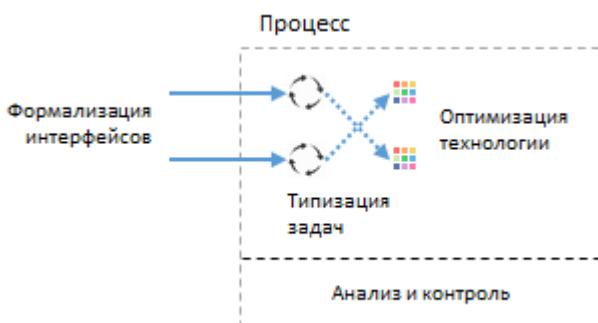


Рис. 115. Назначение процесса

Процесс не сразу возникает - ведь обычно в процессе участвует несколько человек и им надо понять, что есть одинаковые вещи и привыкнуть к тому, чтобы всем одинаково выполнять свою работу. Чтобы говорить о действующем процессе, нужен функционирующий механизм контроля - объективной

оценки текущего состояния и корректировки процесса. Такой контроль можно выполнять вручную, но в современном мире объем информации в работе даже трех-четырех человек такой, что один человек с ним не справится, не сможет свести множество операций в единую статистику, проанализировать и сделать правильные выводы. Кроме того, иногда требуется реагировать в оперативном режиме, и контролер, действующий в ручном режиме, просто не успеет этого сделать. Все это приводит к тому, что внедрение современных процессов немыслимо без автоматизации.

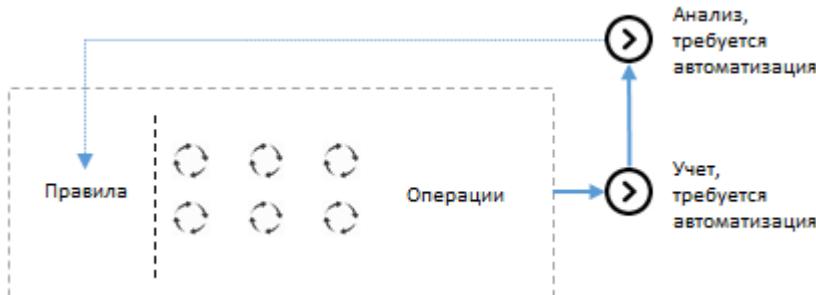


Рис. 116. То, что делает процесс существующим не на бумаге

Релизный процесс и процесс управления задачами - это два разных процесса, которые взаимодействуют, но относятся к разным типовым объектам, и отражают разные операции. Если рассмотреть создание программных продуктов, то задачи разработки в них относятся к типовым элементам - для каждой из них надо сформулировать постановку задачи, проработать реализацию, написать код, отладить его, протестировать и отразить в документации. Однако, как правило, мы не можем сделать сборку задачи, выпустить релиз задачи, и даже протестировать задачу как таковую, без учета других выполненных или выполняемых в данный момент задач. Когда мы говорим о релизе, мы говорим об объекте, отличном от задачи и у него есть своя схема выполнения, отличная от схемы выполнения задачи. Процесс управления задачами относится к области менеджмента, для которого технические подробности собственно разработки не важны. Результат процесса - абстракция под названием "выполненная задача". Релизный процесс, в противоположность процессу управления задачами, отражает технологию создания конечного результата - доставки изменений пользователям, так что то, что было сделано в задачах, начинает приносить пользу.

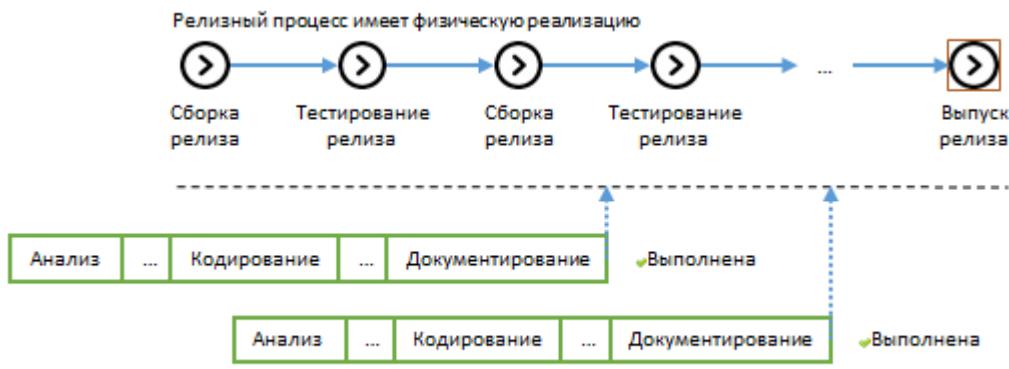


Рис. 117. Взаимодействие логических и физических операций

Автоматизация процесса управления задачами состоит в поддержке формальной схемы процесса, не требующей технических подробностей. Для постановки релизного процесса необходимо решать технологические вопросы управления кодом, сборки, формирования дистрибутивов, поддержки и обновления сред. Именно поэтому все менеджеры умеют ставить процесс управления задачами, но плохо себе представляют что такое релизный процесс. Как следствие, релизный процесс и процесс управления находятся в разных сферах ответственности, в соответствии с областями компетенции. Процесс управления задачами - в руках менеджера проекта, а релизный процесс - в руках релиз-менеджера. Автоматизация процесса управления задачами хорошо известна - это инструменты типа Atlassian JIRA,

MS Project Server и т.п. Однако релизный процесс для своей постановки также требует автоматизации, инструменты для которой гораздо менее известны и множество проблем с релизами программных продуктов вызваны именно отсутствием такой автоматизации.

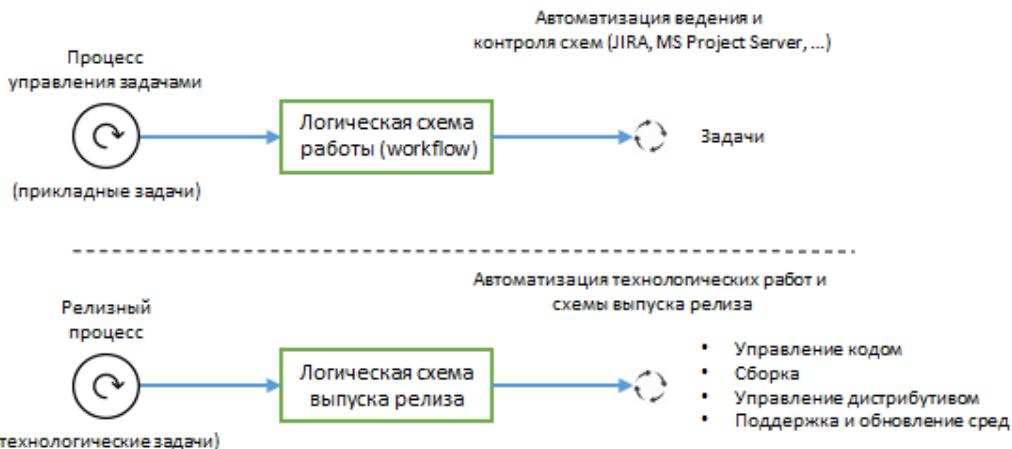


Рис. 118. Аналогичные элементы процессов

Релизный процесс и релизный цикл

Любой реальный и живой продукт требует множества релизов, от нескольких в месяц до нескольких в день, и выпустив несколько релизов, неизбежно приходишь к мысли о том, что от релиза к релизу повторяются одни и те же организационные и технологические проблемы, которые требуют одних и тех же подходов к решению. Т.е. к выпуску релизов надо подходить как к процессу, в котором релиз - одна типовая операция. Типовая - означает, что несмотря на разные функции продукта, которые выводятся в релизе, с точки зрения действий по выпуску релиза многое остается одинаковым.

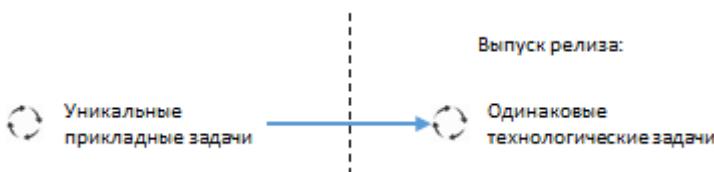


Рис. 119. Единая технология для разных задач

Хотя общий состав задач по продукту на период обычно явно планируется, тем не менее как эти работы распределены по релизам, вопрос неоднозначный. Поскольку релиз требует специфической организации и координации работ, то с большой вероятностью, если релизы в проекте явно не планируются, и выпускаются по "ходу дела", то организация работ страдает беспорядочностью, у разработчиков, эксплуатации и заказчика нет понятных всем ожиданий. Эффективность такого процесса существенно ниже, чем могла бы быть. Поэтому релизы обычно заранее планируют и делают это одним из двух способов:

- Исходный перечень задач на период разбивается на группы и для каждой группы планируется отдельный релиз. Очень удобный для бумажного планирования способ, который однако работает только в том случае, если все задачи можно точно просчитать, или точные сроки выполнения задач настолько важны, что по мере необходимости выделяются дополнительные ресурсы, которые позволяют решить все непредвиденные проблемы. Отметим, что зачастую точные сроки как раз не важны и такой подход отражает неумение менеджера планировать и управлять рисками. Поскольку если сроки не важны, то дополнительных ресурсов менеджеру не дают и непредвиденные обстоятельства приводят к тому, что каждый релиз задерживается, и десять человек ждут двух. В сумме проект выполняется существенно дольше и затраты на него тоже получаются больше.
- Планируется выпуск релизов каждые сколько-то дней, недель или месяцев, без попыток зафиксировать состав задач для каждого. Это то, что называется регулярным циклом выпуска

релизов. Такой вариант удобен тем, что к нему постепенно приспосабливаются все участники - разработчики, эксплуатация, заказчик и процесс становится максимально прозрачным и эффективным.

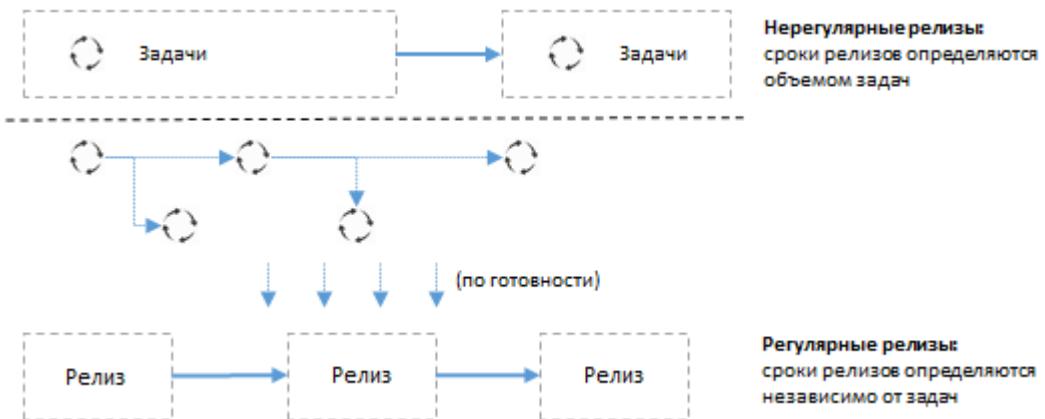


Рис. 120. Сроки регулярных и нерегулярных релизов

Каждый релиз требует каких-то действий, которые когда-то начинаются и когда-то заканчиваются. Поэтому можно говорить о начале работы над релизом и об окончании этой работы, о периоде выпуска релиза. Однако выпуская один релиз, нам приходится уже делать что-то для следующего релиза. Для релизов в одном релизном цикле есть два варианта:

Периоды выпуска следующих друг за другом релизов не пересекаются. В этом случае команда разработки одновременно занимается только одним релизом из потока. Это кажется удобным, но если включить в состав работ и планирование, и анализ, и согласование с заказчиком, и тестирование, и подготовку документации - то, например, для текущих релизов это нереалистично. Аналитики обсуждают задачи с заказчиком, готовят постановку, а разработка получается сидит и ждет. Так не бывает. Поэтому непересекающиеся релизы означают следующее:

- Это могут быть основные релизы, которые похожи на мини-проекты, где на начальной стадии программисты вовлечены в проектирование, затем они выполняют основной объем программирования, затем отслеживают подготовку к установке и обновление. Т.е. спектр задач программистов существенно шире кодирования, а длинный цикл позволяет им оставаться все-таки программистами, а не мастерами широкого профиля. Это может быть психологически важно - человек ведь не машина и программист не может однообразно писать программы изо дня в день, ему необходимо переключаться и на другие виды деятельности.
- Из работ, относящихся именно к релизу, исключаются некоторые виды работ - например, планирование (которое в этом случае происходит неконтролируемым образом в неопределенное время), анализ и даже сама разработка (например, первичная разработка альфа-версии может выполняться в рамках задачи, будучи непривязанной ни к какому конкретному релизу). Оставшиеся работы в этом случае могут быть разложены по непересекающимся периодам.



Рис. 121. Занятость разработчиков в основных релизах

Пересекающиеся периоды выпуска релизов. В этом случае, например, пока программисты и тестировщики выпускают один релиз, менеджеры и аналитики планируют, согласовывают требования и делают постановку задач для следующего релиза. Релизный цикл выглядит как лесенка - каждый релиз образует ступеньку, ступеньки накладываются друг на друга и смешены друг

относительно друга. Тем не менее, для пересекающихся релизов тоже не все работы могут относиться на сам релиз. Если работы невозможны ограничить по срокам или сроки выходят за рамки принятого релизного цикла, они ведутся только в рамках процесса выполнения задач, но вне релизного процесса.

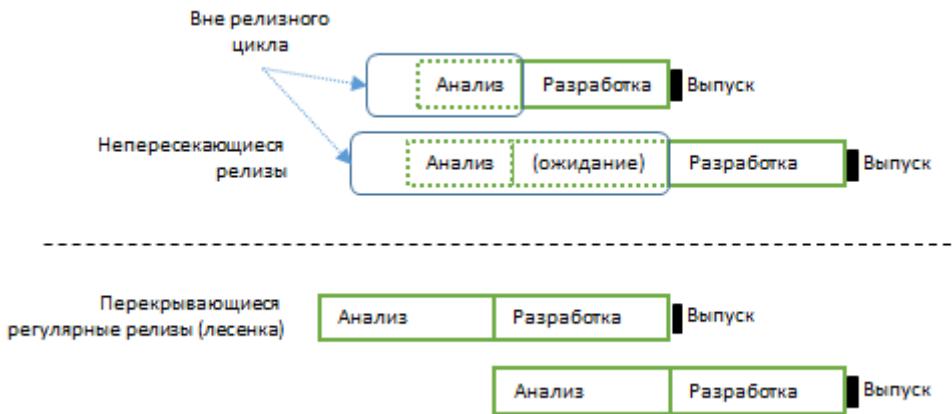


Рис. 122. Включение и исключение работ из релизного цикла

Фазовый контроль

Релизы, которые выпускаются не ночью из последних сил, без нервных срывов и боязни, что что-то вдруг пойдет не так, которые не требуют двух менеджеров на трех программистов - такие релизы как правило случаются не сами по себе, а являются следствием правильной организации релизного процесса.

Сложности релиза с процессной точки зрения заключаются в следующем:

- Выпуск релиза подобен взрыву - очень долго готовятся, закладывают взрывчатку, рассчитывают последствия, всех предупреждают, а потом в несколько секунд все происходит и если кто-то не отошел в сторону или что-то неправильно рассчитали - сразу же получаем негативные последствия. Т.е. сама по себе длительность выпуска не гарантирует успеха, и даже в чем-то расслабляет - пока релиз в стадии выпуска, менеджер может позволить команде все - это внутренность черного ящика, лежащего в кладовке. А в момент, когда релиз выпущен, оказывается, что кто-то забрал дистрибутив, установил, у него все поломалось и нет ни понимания, почему это произошло, ни что делать. Все правильные вещи, которые надо было сделать и которые не были сделаны, дают о себе знать. Это перестает быть внутренним делом команды разработки, но уже поздно, все уже произошло и впереди может быть только следующий релиз, в котором может произойти все то же самое.
- С другой стороны, выпуск релиза может быть подобен детскому стишку "У попа была собака, он ее любил, она съела кусок мяса - он ее убил. В землю закопал и на камне написал - У попа была собака...". Т.е. процесс выпуска релиза входит в непрерывные действия по выпуску, которые повторяются, повторяются и никак не заканчиваются. Программист написал программу, тестировщик нашел ошибку, и программист начал ее исправлять. В это время заказчик позвонил менеджеру и сказал, что через месяц запускают новый техпроцесс и поэтому функция должна выглядеть по-другому, надо кое-что дописать, что-то переписать. Программист опять пишет, тестировщик опять тестирует, опять ошибки, опять исправления. Эту задачу закончили, а другой программист уже начал, но еще не закончил другую задачу. В этом случае говорят, что "процесс не сходится". И все вроде бы не виноваты, все добросовестно выполняют свою функцию. А релиза все нет и нет. Т.е. результат работы команды - ноль. Пользователь не получил новых функций, страдает от неисправленных ошибок. А ведь задачи-то некоторые давно сделаны. Но нельзя отправить в релиз сами задачи, только весь код вместе, поэтому все мы слышали про случаи, когда выпуск релиза задерживается на месяцы и даже годы.

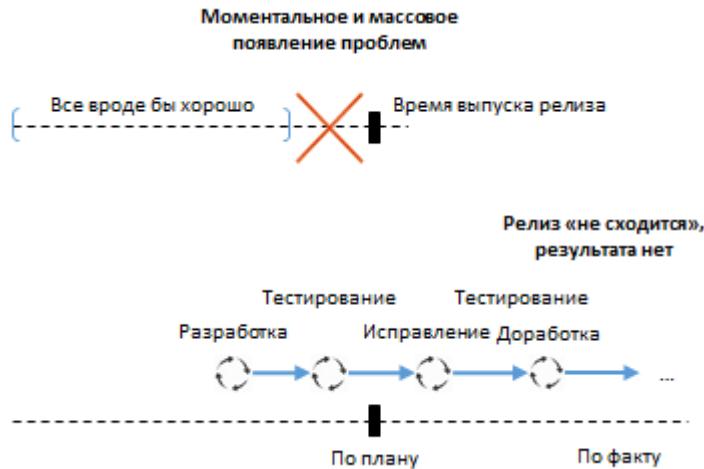


Рис. 123. Типичные проблемы при выпуске релизов

Правильный процесс по выпуску релиза организован так, чтобы были учтены обе эти сложности - релиз должен быть предсказуемым и по качеству и по срокам. Типовым способом выпуска релиза, обеспечивающим сходимость процесса, является механизм фазового контроля, состоящий в следующем:

- Общую длительность периода выпуска релиза делят на непересекающиеся части, которые называются фазами. Фаза определяется как период, к концу которого все задачи релиза должны достигнуть определенного состояния. Смысл фаз можно определить по-разному, но важно, чтобы повторение фазы не имело смысла. По этой причине фаза не совпадает с видом работ, которые производятся по задаче, т.е. с участком схемы процесса выполнения задач. Фазы релиза невозвратны, в то время как схема процесса выполнения задач (workflow) является обычно возвратной - т.е. задача может передаваться то разработчику, то тестировщику или аналитику, и обратно.
- Единственный способ продвижения задач по фазам - только вперед. Если какие-то задачи релиза при достижении срока окончания фазы не могут перейти на следующую фазу из-за нарушения критерия фазы, скорее всего именно они будут причиной того, что релиз не выйдет в срок. Поэтому уже в первой фазе с нарушителями необходимо разобраться:
 - Первым способом является отказ от выпуска этой задачи в данном релизе. Если текущие релизы выходят регулярно, то обычно нет ничего страшного, если задача выйдет чуть позже. К этому варианту решения проблем необходимо еще на стадии организации релизного цикла подготовить команду разработки и заказчика. Получится, что точечно по отдельным задачам возникают проблемы и это приводит к сдвигу решения этих задач, но влияния на остальные обязательства это не оказывает. При таком подходе команда разработки берет изначально обязательство выпустить некоторый объем задач за определенный период, но точный релиз выпуска конкретной задачи фиксировать с заказчиком не нужно - это приведет к обязательствам, которые вам придется нарушать.
 - Вторым способом является взятие менеджером именно этих задач на контроль. Менеджер в сбалансированной команде обычно не успевает вникать во все задачи и все их отслеживать. Фазовый контроль дает менеджеру способ понять, на что необходимо обратить внимание. При этом задача остается в релизе и на последующих фазах ее возвращают в график. В том числе и при помощи дополнительных ресурсов.
- Добившись определенного состояния задач на одной фазе, важно сохранить это качество и на следующей фазе. Например, если в фазе планирования задачи не было в плане и с учетом рисков ее все-таки включили в релиз, то на фазе готовности задач к тестированию данная задача должна также быть готова. Задержка на одной фазе не дает ей оправдание для других фаз, иначе именно эта задача и задержит весь релиз. Представьте себе, что в метро на станции сердобольный машинист поезда долго ждет, пока сядут пассажиры, бегущие к поезду от входа. И он совсем не думает о том, что на следующей станции скопились пассажиры, а в тоннеле скапливаются поезда. Весь ритмичный порядок движения нарушается и достаточно задуматься

о том, что чувствуют пассажиры поезда, стоящего в тоннеле, чтобы понять, что надо закрыть двери и поехать, несмотря на все уважение к спешащим сесть людям.

Правильно определив фазы - так, чтобы продвижение соответствовало объективному приближению к конечному результату и разбираясь с проблемными задачами не в конце выпуска, а по мере обнаружения, мы получим процесс, в котором наш релиз постепенно и устойчиво подходит к успешному завершению.



Рис. 124. Критерии завершения фаз

Жизненный цикл релиза

Совокупность фаз с фиксированной длительностью (фазовый график) составляет жизненный цикл релиза. В общем случае жизненный цикл делится на две части - до выпуска релиза и после выпуска релиза, обозначая работы по выпуску релиза и работы по доведению релиза до конечного использования (поставке релиза). В каждой из двух частей определяется одна или несколько фаз. В конкретном релизе есть определенная дата выпуска, от которой строится план-график релиза, где фазы получают определенные даты начала и окончания. Фазы выпуска релиза строятся от даты выпуска назад, а фазы поставки релиза строятся от даты выпуска вперед.



Рис. 125. Календарное планирование выпуска и поставки релиза от даты выпуска

Фазовый график определяется заранее и является планом релиза. По умолчанию фазовый график не предназначен для автоматической корректировки по факту обнаружения проблем в задачах. Тем не менее, если дата выпуска релиза по каким-то причинам сдвинулась, фазовый график поменяется. Фазовый график также позволяет разделить происходящее в релизе на хорошо и плохо. Если плохого становится слишком много и указанные выше действия не помогли, есть два варианта:

- сдвинуть релиз - изменить дату выпуска на более позднюю, что с точки зрения качества предпочтительно, поскольку при обоснованном исходном графике нет никаких причин считать, что дальше все будет лучше, чем планировалось
- изменить структуру фаз - сдвинуть окончание текущей фазы, продлив ее за счет уменьшения последующих фаз

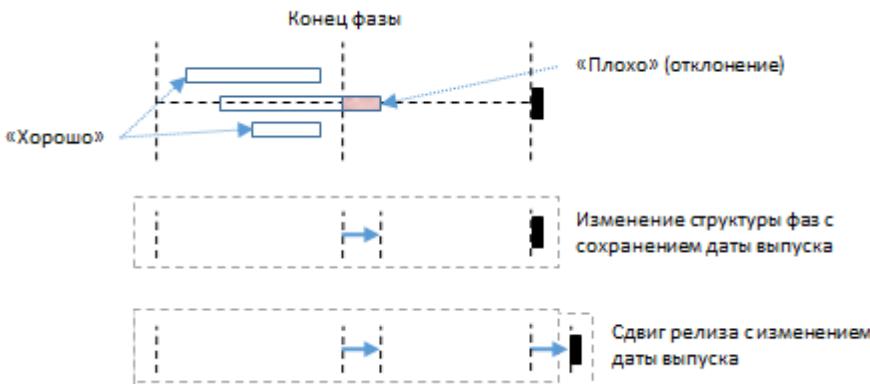


Рис. 126. Фиксация отклонения и принятие решения о сдвиге релиза

Таким образом плановый график превращается в фактический. В конце необходимо провести работу над ошибками, сравнив плановый и фактический графики. Это позволит понять, насколько фазовая структура релиза была хороша и насколько правильно были выбраны пропорции длительности фаз.

Для нерегулярных релизов жизненный цикл задается в начале. Чтобы его определить, необходимо указать структуру и длительность фаз. Это можно сделать одним из следующих образов:

- на основе графика существующих регулярных релизов или на основе графика предыдущего аналогичного релиза:
 - оценить время выпуска на основании сравнения объема задач с типовым объемом регулярного релиза (объем можно оценить обычными методами проектного управления) или просто как требуемая дата выпуска (возможно, стоит вычесть некоторый риск-буфер) минус текущая дата
 - взять фазы из регулярного цикла (или фактического графика предыдущего релиза) и пропорционально времени выпуска изменить длительность фаз
 - добавить фазы, отражающие специфику данного релиза
 - приложить дату выпуска к фазовому графику и получить план-график данного релиза

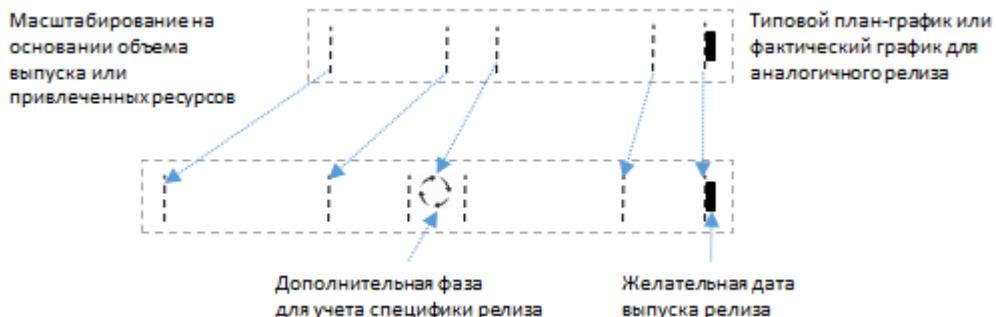


Рис. 127. Построение плана-графика релиза «по аналогу»

- на основе оценки работ, которые будут выполняться на соответствующих фазах
 - вывести заведомо непредсказуемые события за рамки релиза, управляя ими методами проектного управления в рамках процесса выполнения задач
 - определить типичные фазы - начальное планирование, разработка альфа-версии (все функции реализованы, готово к тестированию), разработка бета-версии (подтверждение готовности от независимого тестирования), добавить технологические и специфические фазы (например, нагружочное тестирование, финализация релиза)

- определить длину фаз, пользуясь классическим подходом о выделении 30% на программирование и 30% на тестирование

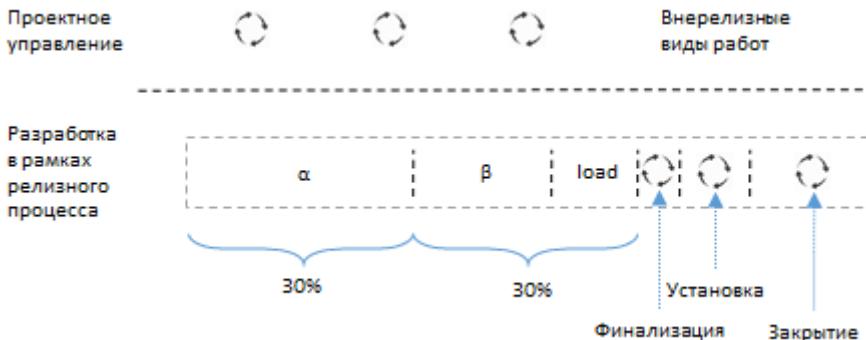


Рис. 128. Эффективное распределение работ по графику релиза

Для регулярных релизов жизненный цикл является стандартным. Исходно он определяется так же, как и для нерегулярного релиза, используется в последующих релизах и время от времени корректируется на основании работы над ошибками. Необходимо очень тщательно подходить к изменениям релизного цикла - изменения должны быть редкими и обоснованными, поскольку множество участников операций подстраиваются под конкретный цикл и теоретически верная перестройка может на практике нанести больше вреда чем пользы.

Примеры циклов:

Ежеквартальный релиз:

- Длительность - 90 дней, регулярный, без перекрытия, 80 (календарных) дней на выпуск, 10 на поставку, резерв 1 день на выпуск и 1 день на поставку
- Фазы выпуска:
 - Планирование релиза - 7 дней
 - Анализ и проектирование - 20 дней
 - Начальная реализация - 30 дней
 - Стабилизация - 14 дней
 - Регрессионное и нагружочное тестирование - 7 дней
 - Финализация - 1 день
- Фазы поставки:
 - Установка - 2 дня
 - Закрытие релиза - 7 дней



Рис. 129. Пример типового релизного цикла еженедельного релиза

Еженедельный релиз:

- Длительность - 21 день, регулярный со сдвигом 7 дней, с перекрытием, 14 (календарных) дней на выпуск, 7 на поставку, без резерва
- Фазы выпуска:
 - Планирование и анализ - 7 дней
 - Разработка - 6 дней
 - Финализация - 1 день
- Фазы поставки:
 - Установка - 1 день
 - Закрытие релиза - 6 дней

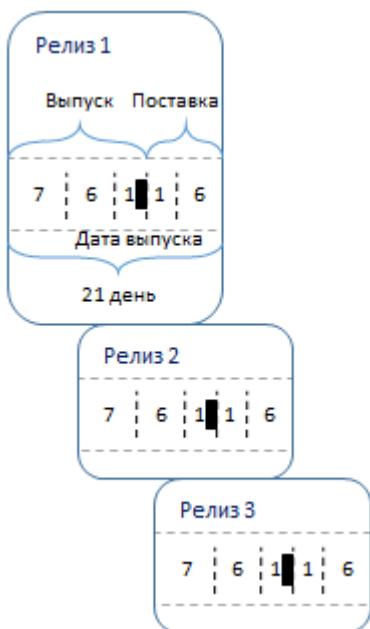


Рис. 130. Пример типового релизного цикла еженедельного релиза

Управление составом релиза

Состав релиза (scope) - это состав задач, результаты которых должны войти в данный релиз. Входной информацией для планирования релиза является информация процесса управления задачами.

Процесс управления задачами обычно использует какое-нибудь средство автоматизации для учета задач (например, Atlassian JIRA или RedMine). Также обычно в этом инструменте заведен классификатор с номерами релизов, который позволяет приписать номер релиза конкретной задаче. Однако вести учет состава релиза в данном инструменте будет недостаточно по следующим причинам:

- не каждая задача относится к какому-нибудь релизу, и не каждая задача при заведении сразу относится к определенному релизу, а отнесение задачи к релизу не означает, что релиз на самом деле фактически содержит что-то по этой задаче на физическом уровне
- поменять номер релиза в задаче легко, даже слишком легко, чтобы на это полагаться - включение или исключение задачи из релиза связано с технологическим процессом переноса кода в ветку для вывода или отката, с установкой в среду тестирования, с контролем связанных изменений в коде продукта, файлов изменения базы данных и конфигурационных изменений
- система фазового контроля не является эквивалентом схемы выполнения задачи - она устроена как набор состояний, постепенно приобретаемых задачей

- к релизу полагается структурированный отчет, который содержит информацию о составе и статусе релиза, но структуру и данные, которые нужны в нем, как правило невозможно получить автоматически из инструмента управления задачами (если это все-таки пытаются делать, то отчет становится нечитаемым и де-факто не используется)

Состав релиза рекомендуется учитывать отдельно в специальном листе контроля (check-list), который является артефактом релизного процесса. В листе контроля есть один или несколько списков изменений (change list). В нем можно отразить весь релиз в нужной структуре и с нужной детализацией.

Задача	Критерий фазы 1	Критерий фазы 2	Критерий фазы 3
DEV-341	✓	✓	TBD
DEV-342	✓	✓	✓
DEV-381	✓	DESCOPED	-
DEV-400	✓	✓	✓
DEV-401	✓	TBD	TBD

Состояние кода в ветке вывода релиза

Рис. 131. Лист контроля задач

Лист контроля устроен как таблица с группировками и колонками, соответствующими фазам. Прохождение задачи через фазу отмечается в листе контроля отметкой, которая может ставиться, но не может сниматься. Записи в этом списке не удаляются. Если задача снята с релиза, в колонке, соответствующей последней фазе, ставится отметка DESCOPED.

Записи, находящиеся в списке контроля, соответствуют состоянию ветки, с которой производится вывод релиза. Работы, выполняемые в рабочей, внутренней ветке кода, учитываются в инструменте управления задачами.

Когда появляется новая задача, необходимо провести анализ и определить в инструменте управления задачами, куда она попадает с точки зрения релизов. Здесь есть три характерных варианта:

- задача относится к неопределенному будущему, т.е. ее неплохо бы сделать, но для ближайших планов есть более важные задачи - это так называемая "помойка", которую можно пометить релизом с условным названием "future"
- задача относится к текущему основному периоду, и должна быть выпущена в ближайшем основном релизе или в одном из ближайших текущих релизов до выпуска основного релиза, и помечается соответствующим релизом
- задача относится к ближайшему текущему релизу, фаза планирования которого уже завершена - в этом случае необходимо сначала обсудить возможность включения в состав релиза с лицами, контролирующими выпуск релиза

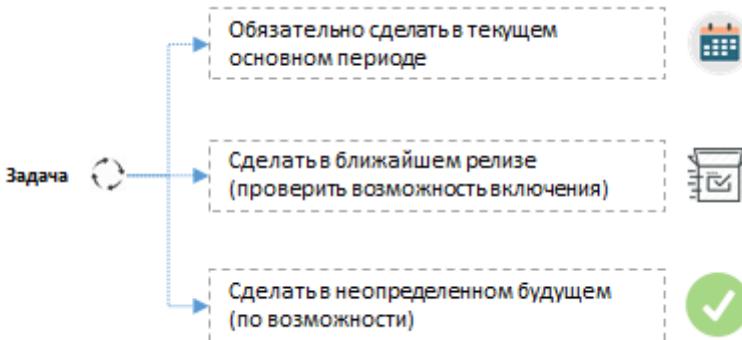


Рис. 132. Варианты при релизном планировании применительно к конкретной задаче

Фаза планирования релиза (если она явно определена) протекает следующим образом:

- задачи помечаются в инструменте управления задачами как связанные с данным релизом
- исполнитель по задаче оценивает задачу с точки зрения готовности к выпуску в данном релизе
 - если исполнитель считает, что сделать задачу в сроки данного релиза не получится, он сообщает об этом менеджеру и задачу переводят на последующий релиз
 - если исполнитель считает выпуск возможным, то он принимает ответственность за проведение задачи по всем фазам релиза и отражает этот факт, занося информацию о задаче в лист контроля
- в конце фазы планирования список задач в списке контроля считается начальным составом релиза - именно этот список определяет какой код необходимо собирать и какие артефакты должны попасть в дистрибутив

Если при завершении одной из фаз релиза задача не помечена как успешно прошедшая фазу, то эта задача является кандидатом на исключение. По всем таким задачам проводится совещание на предмет исключения из релиза. Исключенная задача помечается релиз-инженером как DESCOPED в колонке последней фазы выпуска.

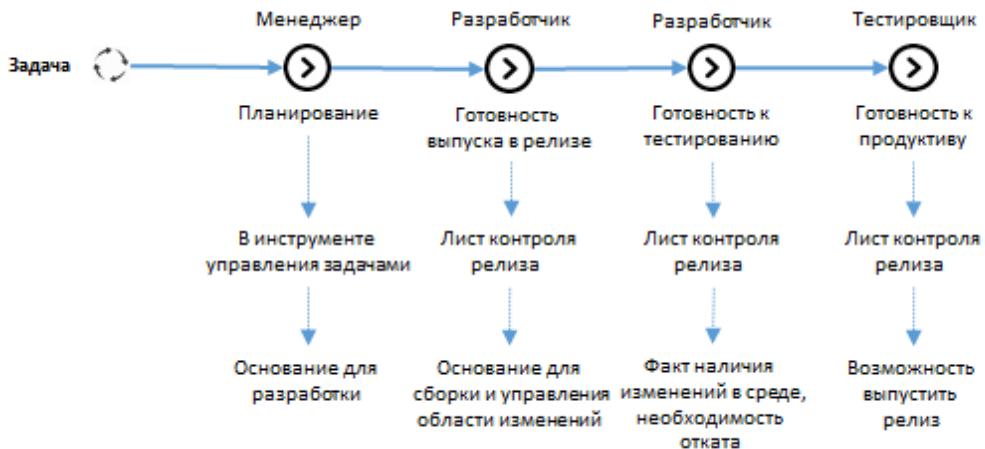


Рис. 133. Учет планирования и выполнения задачи

Если возникла необходимость включения новой задачи после завершения планирования, то это выполняется в следующем порядке:

- лица, отвечающие за выпуск релиза, обсуждают соответствующие риски и принимают решение о возможности включения задачи в релиз (чем на более поздней фазе выпуска это происходит, тем более критическим должен быть анализ и лица с более высоким статусом должны участвовать в принятии решения)
- задача помечается в инструменте управления задачами как относящаяся к данному релизу
- исполнитель по задаче берет ответственность за выпуск задачи в данном релизе и вносит запись на лист контроля без пометки о включении в план
- релиз-инженер корректирует план сборки, план установки релиза и помечает задачу как включенную в план

Если в процессе разработки выяснилось, что задача должна быть снята с релиза, то в инструменте управления задачами задача переносится на последующий релиз, а релиз-инженер выполняет и отслеживает соответствующие действия по физическому исключению задачи из сборки и артефактов релиза. При этом может потребоваться учесть зависимости и исключить другие задачи.

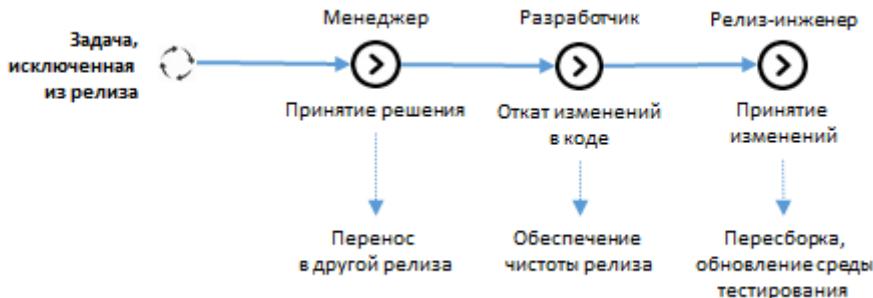


Рис. 134. Отказ от вывода задачи в текущем релизе

Необходимо отметить, что списки изменений могут не соответствовать фактическим изменениям в коде из-за откатов, внутреннего рефакторинга, не подлежащего тестированию, исправлению низкоуровневых ошибок разработчиком. Тем не менее, ни один артефакт не должен выпускаться в релизе с пустым списком изменений.

Отмена релиза

Несмотря на все выполненные действия, а возможно, как раз из-за того, что они не были выполнены, ситуация с релизом может зайдти в тупик. Например, определенная задача не готова и непонятно, когда будет готова, но исключить ее из релиза нельзя, поскольку все остальные задачи уже от этой зависят, откат изменений просто приведет к практическому пустому релизу. Если у вас регулярные релизы, то в этом случае имеет смысл признать релиз несостоявшимся и перейти к новому циклу. Для частых текущих релизов это с практической точки зрения не очень важно. Несмотря на негативную окраску такого факта, лучше не нарушать цикличности, задерживая данный релиз и пытаясь его доделать. Пользы от такого решения будет больше, поскольку иначе следующий релиз у вас также пойдет не по графику.



Рис. 135. Отображение изменений по задачам на изменения в среде

Отмененный релиз сохраняет за собой свой номер и следующий релиз будет иметь следующий номер. В выпущенных версиях появится пропуск.

Непрерывная поставка

Непрерывная поставка (Continuous Delivery) - это принцип организации работ по выпуску и поставке релизов, когда:

- Совокупность релизных циклов и потоков релизов такова, что при необходимости изменение попадает в среду эксплуатации тогда, когда нужно, по факту готовности изменения.
- Релизные циклы автоматизированы как в выпуске, так и в поставке до такой степени, что с минимальными задержками изменение попадает от разработчика в среду эксплуатации.

Идея непрерывной поставки опирается на более общую технологию JIT (Just In Time), которая используется, чтобы шить джинсы промышленным способом для одного покупателя по его размерам,

чтобы печатать книжку тогда, когда она понадобилась - т.е. делать вещи тогда, когда они понадобились. Система релизного процесса при технологии непрерывной поставки позволяет в любой момент выпустить новый релиз, без технологических задержек, излишних для данного изменения, и без влияния на другие изменения. Вторым вариантом реализации является система, при которой текущие регулярные релизы имеют настолько короткий цикл, что технологические задержки воспринимаются как несущественные.

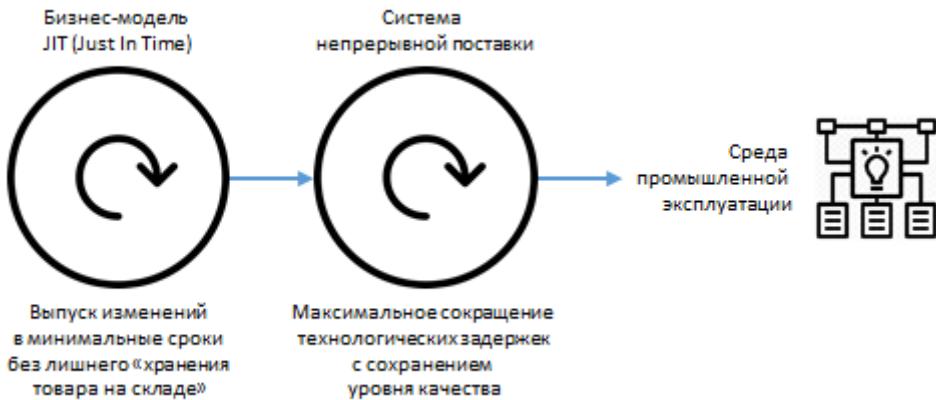


Рис. 136. Применение технологии JIT

Для системы непрерывной поставки необходимо решить следующие технологические вопросы:

- оперативное управление ветками кода для внесения изменений без риска повлиять на активные релизы с более длинным циклом
- формализация области изменений и единая метамодель кода-сборки-дистрибутива-среды для автоматического просчета всей инженерной цепочки
- автоматическое конфигурирование для того, чтобы множество мелких быстрых изменений, проводимых через разные среды, не привело к ошибкам из-за человеческого фактора
- автоматизация сборки, формирования дистрибутива и установки, чтобы существовал конвейер доставки изменения от кода до эксплуатации без необходимости руками перекладывать какие-то файлы
- отлаженный на организационном уровне процесс и организационная готовность, чтобы при сохранении необходимого уровня контроля выполнять изменения в среде эксплуатации без бюрократических задержек

С точки зрения релизных циклов, кроме просто короткого текущего цикла, может существовать и цикл для срочных релизов, в котором существуют фазы, но их длительность равна нулю. Т.е. цикл не налагает минимальную задержку в фазе. Для релизов под конкретное изменение это нормально, поскольку нет необходимости координировать работы с множеством изменений. По факту готовности разработки задача отправляется на тестирование, по факту завершения тестирования - на установку. Если успели сделать за 5 минут - значит через 5 минут. Такие релизы могут выпускаться в день по несколько десятков раз. Но, разумеется, это не может применяться для любого вида кода, иначе неизбежно зависимости повлекут за собой влияние на текущие релизы. Если в системе есть простые регулярные и независимые данные или микросервисы, их можно вполне менять через такие нулевые циклы микрорелизов.

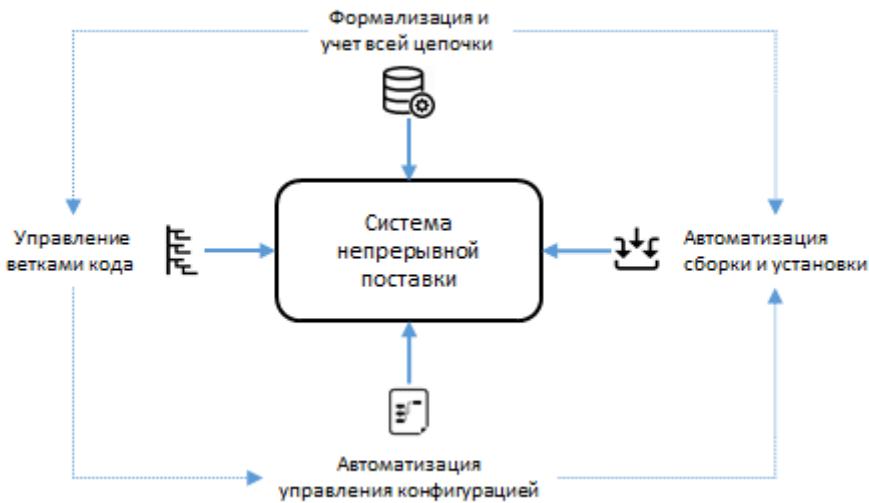


Рис. 137. Построение системы непрерывной поставки

Смежным термином является **непрерывная установка** (Continuous Deployment), когда коммит в репозиторий после автотеста сразу же попадает в среду. Данная технология может применяться в средах промышленной эксплуатации для некоторых выделенных функций, с заранее известным риском. Например, для управления контентом. Маловероятно, что непрерывная установка может применяться для обычного алгоритмического кода, где инженерных аспектов много и автотесты - лишь часть процесса обеспечения качества.

Другой, рассмотренный ранее термин **непрерывной интеграции** (Continuous Integration) - не является обязательным для непрерывной поставки, поскольку обращает внимание на другие аспекты разработки - взаимодействие разработчиков в команде и совместное владение кодом.

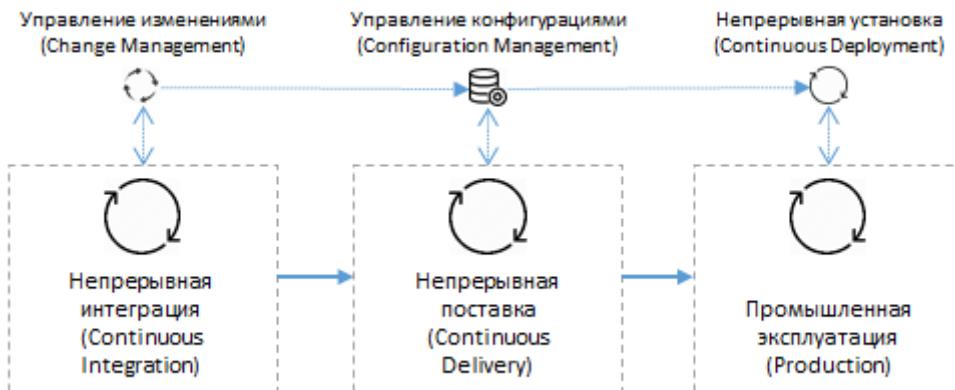


Рис. 138. Объединение технологий

Создание и поддержка сред

ЭКЗЕМПЛЯР ПРОДУКТА

СРЕДА ПРОДУКТА

КОНФИГУРАЦИЯ СРЕДЫ

ДАТАЦЕНТРЫ И ОБЛАЧНАЯ ИНФРАСТРУКТУРА

ОБЛАКА И СЕТЕВЫЕ РЕСУРСЫ

ТИП СРЕДЫ

СЕГМЕНТАЦИЯ СРЕДЫ

ХОСТЫ И АККАУНТЫ

ФУНКЦИОНАЛЬНЫЙ СЕРВЕР

УЗЛЫ СЕРВЕРА

РАЗМЕЩЕНИЕ ЭЛЕМЕНТОВ

ОПЕРАЦИИ В СРЕДЕ

ТЕСТИРОВАНИЕ РЕЛИЗОВ

СОЗДАНИЕ СРЕД ДЛЯ ТЕСТИРОВАНИЯ

ЗАКРЫТЫЕ ДАННЫЕ И ДАННЫЕ ДЛЯ ТЕСТИРОВАНИЯ

Что имеется в виду под средой? Что относится к ней, а что является смежным объектом? Чем отличаются среды друг от друга и сколько нужно сред при организации релизного процесса? Как связаны друг с другом среды разных продуктов?

Экземпляр продукта

Если есть продукт, то с точки зрения конечного использования он может быть тиражируемым, когда он устанавливается в нескольких местах или заказным, когда среда промышленной эксплуатации ровно одна. Понятно, что там, где есть несколько организаций-владельцев тиражируемого программного продукта (владельцев лицензий), следует говорить о нескольких экземплярах продукта. Если же владелец один, ситуация может быть неоднозначной - за экземпляр коммерческого программного продукта платят, но относится ли это к каждой установке?

Структура среды промышленной эксплуатации даже для заказного решения может быть такова, что де-факто продукт просто несколько раз устанавливается в разной инфраструктуре, возможно с разными настройками. Если эти установки совершенно не зависят друг от друга и не взаимодействуют друг с другом, то мы считаем их разными экземплярами продукта. Иначе, если продукт предусматривает такое взаимодействие, то в рамках одной компании-владельца эти несколько установок образуют лишь более сложную конфигурацию продукта.



Рис. 139. Инсталляции в организации

Кроме промышленной эксплуатации, продукт также может быть установлен для целей тестирования разного вида, демонстрации, для воспроизведения инцидентов и т.п. Такие задачи являются естественными и обычно не приводят к необходимости дополнительного лицензирования, хотя разумеется, это определяется конкретным лицензионным соглашением. Т.е. обычно экземпляром продукта мы считаем одну конфигурацию продукта, предусмотренную документацией продукта и находящуюся в одной организации для целей промышленной эксплуатации, допуская при этом одну или несколько установок для целей, отличных от промышленной эксплуатации.

Среда продукта

При описании функционирующего экземпляра продукта мы используем понятия приложений и процессов. Для промышленного использования с процессами тесно связано понятие кластеризации. Попробуем определить, как все эти понятия соотносятся друг с другом.

Приложение - это набор файлов дистрибутива, которые обеспечивают автоматизацию некоторой функции, внутренней или конечной. Приложение содержит программный код или результат его компиляции в р-код (псевдо-код для интерпретатора) или в машинный код. Код программного продукта и процедура сборки в целом определяют список приложений, которые относятся к продукту. Данные приложения установлены в одном или нескольких экземплярах для целей тестирования или конечного использования.



Рис. 140. Приложение

Для промышленного использования приложения устанавливаются в кластерной конфигурации (множество однотипных экземпляров, каждый из которых берет на себя какую-то часть входного потока данного типа), что обеспечивает масштабируемость и отказоустойчивость. Для целей тестирования поведения приложения в кластерной конфигурации тестовую установку также делают кластерной, хотя, как правило, количество узлов в кластере существенно меньше чем в промышленной конфигурации. Для оценки производительности в тестовой зоне могут временно создать подобие промышленной конфигурации, но постоянно держать такие ресурсы неэффективно.

Разные приложения в составе продукта могут быть установлены каждое само по себе, а могут быть объединены в группы, которые работают под управлением так называемых серверов приложений. Эти группы (или одиночные приложения) образуют процессы, которые посредством дублирования формируют кластеры. Если состав приложений определяется кодом продукта, то группировка приложений в процессы (конфигурация процессов) и количество процессов на каждую группу (конфигурация кластеризации) обычно допускают вариации.

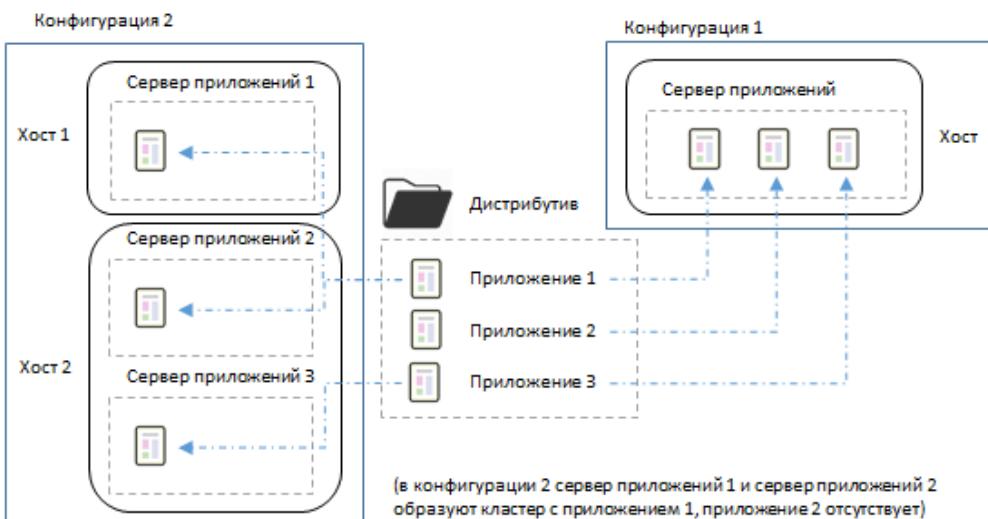


Рис. 141. Процесс сервера приложения

Приложения взаимодействуют друг с другом и зависят друг от друга, поэтому как правило есть некоторые ограничения на то, как приложения можно объединять и какая минимальная кластеризация необходима для соответствия целям использования. В рамках существующих ограничений в продукте может быть оценены и разработаны несколько конечных конфигураций процессов и кластеризации либо выработаны рекомендации, как эти конфигурации формировать в зависимости от условий использования.

Средой называется определенная конфигурация процессов и кластеризации, соответствующая ограничениям продукта, развернутая на определенных инфраструктурных ресурсах (виртуальные машины, дисковые массивы, сетевое оборудование и т.п.).



Рис. 142. Инфраструктура и приложения

Среда - это установленный экземпляр продукта в предусмотренной документацией продукта конфигурации и в состав среды не входят другие продукты (несколько продуктов могут использовать одни и те же ресурсы инфраструктуры). Таким образом, каждый продукт имеет какое-то количество сред, созданных либо для целей того или иного тестирования (включая показ заказчику, отладку разработчиком, нагружочное тестирование и т.д.) - тестовые среды, либо для целей конечного использования - среды промышленной эксплуатации.

Конфигурация среды

К конфигурации среды относятся упомянутые ранее конфигурация процессов и конфигурация кластеризации.

Кроме того, в конкретной среде могут быть развернуты не все приложения, входящие в состав продукта, что так же относится к конфигурации процессов. Причинами для такой ситуации могут быть:

- В данной среде промышленной эксплуатации некоторые функции продукта не нужны и как следствие ненужные приложения не развертывались. Это характерно для тиражируемых продуктов с опциями конфигурации, но может быть и для заказного продукта, если в продукте приложения были разработаны и выпущены в релиз, но в реальное использование так и не были запущены по каким-то причинам.
- Данная версия продукта может содержать какое-то приложение, в то время как в другой версии (более старой или более новой), которая установлена в среде, данного приложения может и не быть.
- Некоторые приложения не предназначены для конечного использования, а являются средством для разработки или тестирования продукта, разработанными как часть продукта. Поэтому они в принципе отсутствуют в среде промышленной эксплуатации.
- Некоторые приложения являются частью инструментария эксплуатации промышленных сред и могут отсутствовать в тестовых средах.
- По инфраструктурным или иным причинам вместо одной среды промышленной эксплуатации, содержащей все нужные приложения продукта, могут использовать несколько сред, содержащих отдельные части продукта. С точки зрения структуры и состояния установленных объектов разницы при этом нет, однако понятие среды используется на организационном уровне и с точки зрения операций, и это разбиение начинает играть свою роль.

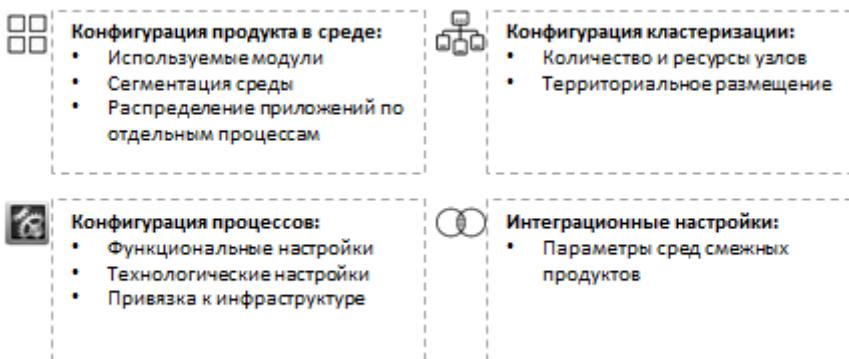


Рис. 143. Параметры конфигурации среды

Данная среда также может иметь уникальные для данной среды значения параметров конфигурации, связанные с:

- Уникальными названиями хостов и IP-адресов, другими параметрами, которые зависят от выбранной конфигурации инфраструктуры (инфраструктурные параметры)
- Функциями продукта, которые должны быть доступны в данной среде (опции конфигурации продукта)
- Тестовыми настройками, связанными с задачами тестирования (например, порт подключения для отладки)
- Настройками, имеющими значение в режиме промышленной эксплуатации (например, учетная запись для операций мониторинга)

Продукт, и как следствие его среды, зачастую существует не сам по себе, а в зависимости от других продуктов, которые необходимы для выполнения операций или даже просто для запуска процессов данного продукта. Это то, что называется системной интеграцией - продукты интегрированы в некоторую систему. Фактически, речь идет о том, что при такой зависимости, каждая среда требует подключения к средам смежных продуктов, от которых данный продукт зависит.

Идентификация смежных сред и параметры конфигурации, необходимые для установления связи с процессами этих сред, также входят в определение конфигурации данной среды.

Датацентры и облачная инфраструктура

В мире ИТ-систем уровня предприятия и выше серверы и сетевое оборудование, на которых работает система, располагается в оборудованных помещениях со специальными климатическими условиями, каналами связи и физической безопасностью. Такое помещение может быть внутренним помещением организации или находиться в выделенном только для этой цели здании. И то и другое мы называем датацентром, что может звучать и слишком громко и слишком обычно для фактического положения дел.

Для крупных национальных и глобальных систем точки доступа располагаются в датацентрах, физически приближенных к соответствующим географическим регионам. Например, датацентры в федеральных округах, датацентры по центрам активности (Европа, Америка, Азия и т.п.). Системы могут располагаться как в выделенных датацентрах, так и на арендуемых мощностях коммерческих датацентров общего назначения. Также глобальная или национальная система может предоставлять сервис, будучи расположенной в одном месте, что будет приводить к разному времени реакции в зависимости от расположения пользователя.

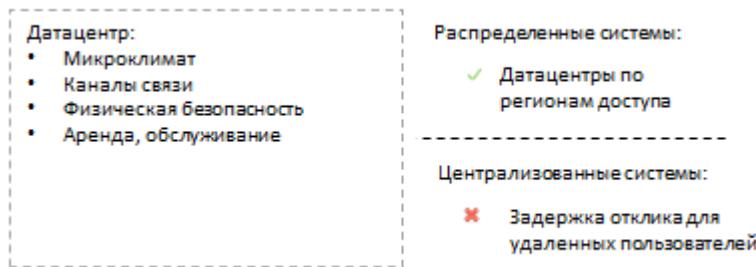


Рис. 144. Централизованные и распределенные системы

При организации релизного процесса, требующего достаточно большого объема обновления как по количеству операций, так и по объему переносимых данных, информация о датацентрах становится важной при больших расстояниях, или при использовании сетевых каналов низкой пропускной способности или низкого качества передачи, даже простая операция может стать невозможной (просто повторяя ее, можно каждый раз получать ошибку) и требовать специальной технологии при совершении операций. Например, передача большого файла может оборваться или файл будет скопирован с повреждениями.

Если ошибку не заметить, то обновление систем может привести к нарушению работоспособности среды, что также может выясниться далеко не сразу и анализ неисправности в системе с сотнями и тысячами файлов может быть совсем не простым делом, и привести к продолжительной неработоспособности.

Есть еще один опасный момент - при операциях системы, нарушение целостности файлов может быть не обнаружено и операции будут помечены как выполненные, но пройдут нештатно. Отмена таких операций и возможные последствия могут весьма проблемными и предсказать такие последствия достаточно сложно в силу практической невозможности предварительно смоделировать и протестировать такого рода повреждения.



Рис. 145. Отложенные проблемы в распределенной системе

Под облачной инфраструктурой понимают вычислительные ресурсы, которые предоставляются в виде виртуальных машин, которые существуют благодаря специальному программному обеспечению систем виртуализации, которое использует специальные массивы оборудования так, что несколько виртуальных хостов используют ресурсы одного физического массива, и эту структуру можно динамически менять, добавляя один хост без прекращения сервиса со стороны другого.

Такие системы повышают эффективность использования оборудования и существенно ускоряют выделение и освобождение ресурсов. Например, память, выделенная виртуальному хосту, но не используемая его процессорами, учитывается системой виртуализации и может быть выделена для использования другому хосту. Т.о. выделенная память в несколько раз может превышать используемую (oversubscription).

Это важно в тех случаях, когда точно оценить необходимые системе ресурсы не получается и ресурсы выделяются с запасом. Данный запас не лежит мертвым грузом, и общая эффективность существенно повышается. Это же касается дисковых и процессорных ресурсов.

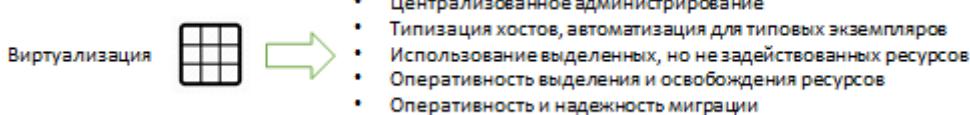


Рис. 146. Преимущества виртуализации

Системы виртуализации имеют механизмы квотирования, возможность создавать, удалять хосты и изменять выделенные им ресурсы через пользовательский или программный интерфейс. Данные функции позволяют довести процесс создания сред до полной автоматизации.

Облака и сетевые ресурсы

Сети предназначены для организации адресации хостов и контроля доступа между источником и получателем, включая фильтрацию пакетов. Источник и получатель могут располагаться на одном хосте, на разных хостах датацентра или на неизвестной или неконтролируемой локации в глобальной сети.

Специальное сетевое оборудование формирует физически существующие сети, но облачная инфраструктура, кроме виртуальных хостов, дает возможность создавать также и виртуальные сети. В итоге типичная сетевая структура в датацентрах имеет трехслойную организацию:

- **Глобальная сеть**, с глобальными адресами, как часть общей сети. Далеко не все хосты датацентра имеют собственные внешние адреса и возможность прямого обращения к ним извне в таком случае отсутствует. Это элемент безопасности. Глобальные адреса отражаются на локальные и наоборот через соответственно механизмы трансляции адресов DNAT и SNAT. Один и тот же хост может быть виден под разными внешними адресами на входящих и исходящих соединениях.
- **Общая локальная сеть датацентра** (возможно разбитая на насколько сегментов), в которой все хосты имеют собственный адрес. Это необходимо для обеспечения взаимодействия систем в датацентре без выхода в интернет.
- **Виртуальная подсеть**, в которую помещается группа виртуальных хостов. Если каждую среду каждого продукта создавать в собственной виртуальной подсети, то инфраструктурное размещение продуктов и систем становится наиболее прозрачным с точки зрения управления. Наличие подсети позволяет настроить сетевое взаимодействие, изолировать продукты друг от друга и разрешить серверам одной среды взаимодействовать произвольным способом. Точное описание сетевой связности внутри продукта порождает высокую степень детализации информации, и затраты на ее поддержку существенно возрастают. Поэтому лучше контролировать лишь межсетевые связи, соответствующие внешним интерфейсам продукта.

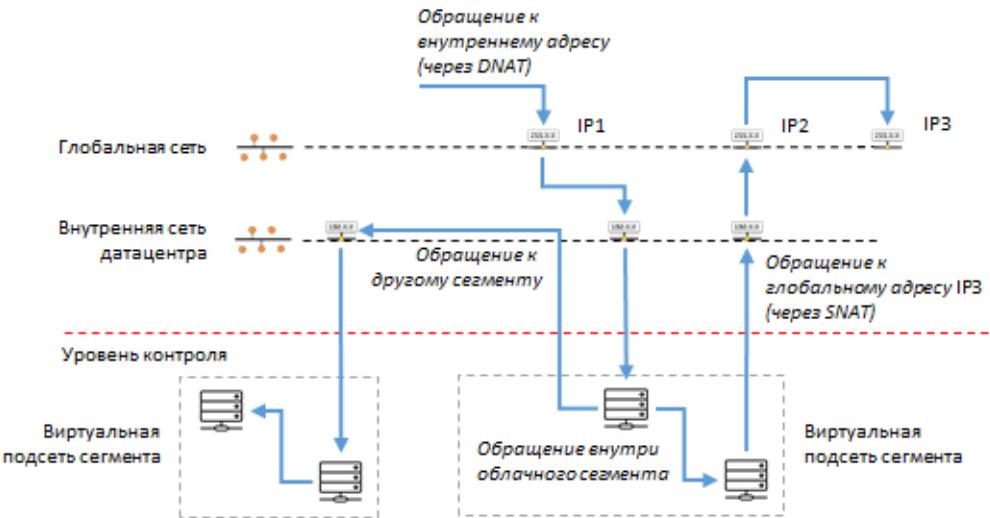


Рис. 147. Организация сетей датацентра

Релизный процесс, который приводит к процессу поддержки сред продукта, связан с датацентрами и сетевой структурой:

- Инфраструктура разработки - где находится код, где и как выполняется внутренняя сборка, где находятся среды разработки
- Инфраструктура выпуска релизов - где выполняется сборка для релизов, где находятся среды, после тестирования в которых выпускается релиз, где находятся дистрибутивы релизов
- Инфраструктура промышленной эксплуатации - где находятся среды промышленной эксплуатации, где находятся пользователи.

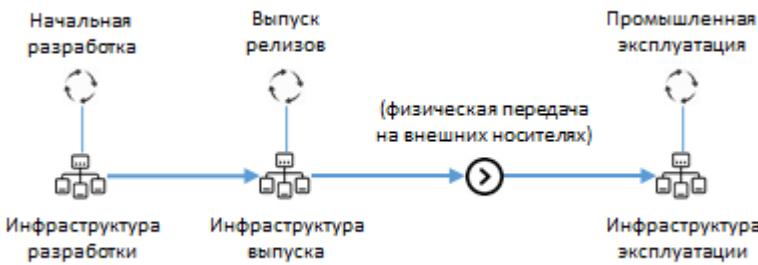


Рис. 148. Релизный процесс и инфраструктура

Все части инфраструктуры связаны общими процессами и требуют, чтобы на сетевом уровне были организованы связи. Может быть такое, что сети между этими инфраструктурными сегментами нет. В таком случае решаются вопросы физической передачи данных на внешних носителях, и это неизбежно усложняет и замедляет процессы. Связи имеют определенную надежность, пропускную способность и защищенность. При организации и настройке операций это также необходимо учесть.

Тип среды

Среда продукта является целостной системой программных элементов и в процессе использования приводит к такой же целостной системе операционных данных.

Выполнение разных по значению операций может привести к нарушению целостности данных. Например, тестирование, выполняемое в среде промышленной эксплуатации, порождает тестовые

операции, которые не имеют формального статуса и могут приводить к нарушению финансовой и юридической отчетности, сделанной на основании данных среды. Еще более странно будет выглядеть отладка разработчиком в среде промышленной эксплуатации.

Можно конечно тщательно отделить тестовые данные от остальных, но это усложняет и вопросы доступа и вносит лишние риски. Как правило, в среде промышленной эксплуатации для целей проверки работоспособности либо выполняют немодифицирующие операции, либо используют средства для отмены операций.

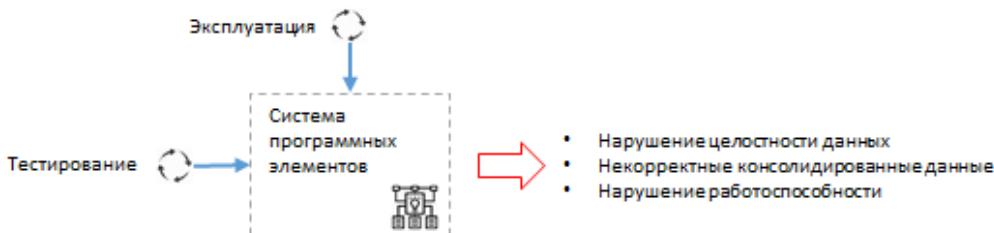


Рис. 149. Последствия использования одной среды в разных целях

Принципиальным делением сред на категории с точки статуса данных, которое в любом случае необходимо провести:

- **Среда промышленной эксплуатации.** Конечное использование. Операционные данные имеют формальный статус.
- **Среда тестирования.** Достаточно общая категория, в которую попадает и среда для показа заказчику, и среда разработки. Общее для них то, что любые операции могут быть при необходимости повторены, поскольку выполненные операции никакого формального значения не имеют.

Принципиальным делением сред на типы с точки зрения уровня контроля является:

- **Среда разработки.** Не контролируется вне команды разработки. Механизмы и правила обновления, выполняемые операции не определяются никем извне и ни на что не влияют, кроме внутреннего процесса. К этому типу относятся среда разработки, среды внутреннего тестирования и т.п.
- **Вспомогательная контролируемая среда.** Служит для какой-то определенной цели вне команды разработки и должна соответствовать некоторым внешним требованиям по своему состоянию - версиям элементов, настройкам и времени доступности. К этой категории относится среда для демонстрации, среда для интеграционного тестирования, среда для выпускающего тестирования, среда для приемочного тестирования и т.п.
- **Среда промышленной эксплуатации.** Это основная контролируемая среда, которая обеспечивает конечную цель существования продукта.

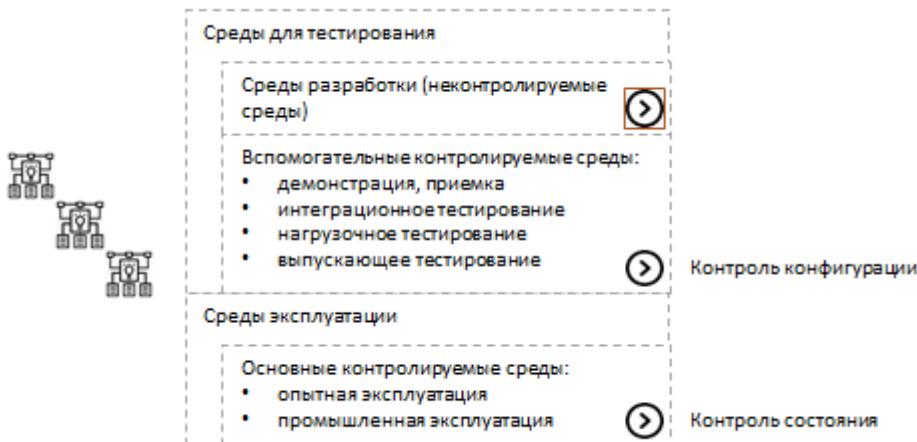


Рис. 150. Классификация сред

Деление сред по цели использования может носить весьма разный характер для разных продуктов и его уже нельзя так однозначно определить. Также хотелось бы отметить, что в принципе количество сред необходимо сводить к минимуму, объединяя несколько целей в рамках одного типа по уровню контроля, по следующим причинам:

- **Эффективность использования инфраструктуры.** Понятно, что каждая среда требует отдельных вычислительных ресурсов и чем больше сред, тем больше ресурсов задействовано. При объединении целей использования возрастают требования к среде, но сокращаются затраты на оборудование.
- **Затраты на поддержку сред.** Автоматизация развертывания может упростить задачу поддержки сред, однако для относительно сложных продуктов большое количество операций (включая своевременный запуск средства автоматизации) требует отдельных усилий на каждую отдельную среду и в целом объем затрат становится существенным. В условиях частичной автоматизации затраты становятся еще больше.
- **Качество сред.** Пользователь среды (связанный с данной целью использования) является лицом, наиболее заинтересованным в работоспособности среды и ее соответствия требованиям. Каждый факт использования по сути является фактом контроля среды на соответствие требованиям в той или иной мере. Поскольку именно для пользователя существует данная среда, то именно его сообщения о проблемах являются наиболее важными и как правило не игнорируются, что приводит к устранению неисправностей. Чем больше пользователей и операций, выполняемых ими, тем больше исправленных проблем и тем быстрее эти проблемы решаются. Таким образом, один пользователь фактически помогает всем остальным. Чем больше целей использования обеспечивается данной средой, тем выше ее качество.

При объединении нескольких целей в одной среде могут возникнуть конфликты интересов с точки зрения требуемого состояния - одним требуется одна версия продукта, а другим - другая, одним одна производительность, а другим - другая, одним надо обновить среду сейчас, а другим надо, чтобы среду сейчас не останавливали и не изменяли.

✖ Конфликт интересов: <ul style="list-style-type: none"> по версиям по производительности по функциям по времени недоступности 		✓ Выгоды: <ul style="list-style-type: none"> сокращение вычислительных ресурсов сокращение затрат на поддержку повышение качества сред
---	--	--

Рис. 151. Основные дилеммы, возникающие при объединении сред

Итого, типы сред определяют разграничение сред по уровням контроля, а выбор оптимального набора сред для каждого типа является результатом рассмотрения разных целей и соответствующих требований с целью определения минимально возможного комплекта, в котором разные интересы

взаимно согласованы, возможно в виде некоторого регламента использования, обеспечивающего оптимальный компромисс.

Сегментация среды

Чтобы описать среду, необходимо определить ее конфигурацию. Самая простая среда - это одна программа на одном компьютере и конфигурация проста и понятна каждому. Однако, когда программные продукты усложняются и среда размещается на десятках и сотнях компьютеров, работа с этим множеством как с единым списком становится неудобной.

В сложной системе неизбежно возникает своя структура, по географическому признаку, по функциям или по областям ответственности. Это деление, возникшее по объективным причинам и облегчающее управление системой, стоит зафиксировать явно на уровне формальных элементов и метаданных в учетной системе. В общем случае деление представляет из себя дерево, но для практических целей лучше ограничиться одним уровнем группировки - сегментами среды.



Структурирование среды на сегменты:

- по функциям
- по географическому расположению
- по областям ответственности

Рис. 152. Основные причины

Формально зафиксированная сегментация позволяет определять операции и правила через названия сегментов, что сделает действия и описания более четкими и компактными, а также гибкими, поскольку состав сегмента может меняться со временем.

Сегментацию также удобно применять, когда некоторая часть продукта устанавливается в нескольких экземплярах с разными настройками. Тогда сегмент сопоставляется с экземпляром, а его настройки привязываются к сегменту. Иначе пришлось бы указывать настройки для каждого процесса отдельно, с появлением дублирования и вызываемых им ошибок. Также можно экземпляр сопоставить с отдельной средой.

На двух сегментах среды одна и та же версия продукта, остальные части среды без изменений

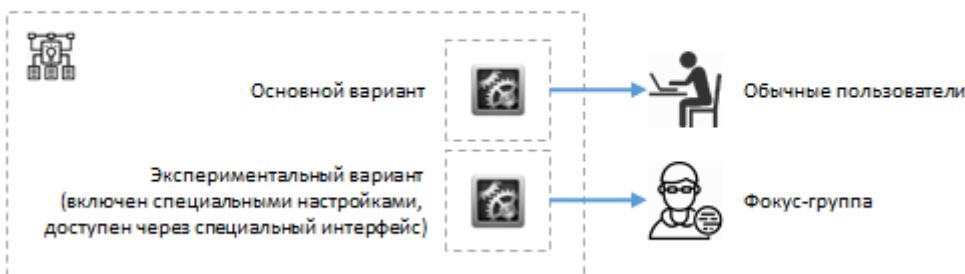


Рис. 153. Реализация ABC-тестирования

Хосты и аккаунты

Облачная инфраструктура позволяет быстро создавать виртуальные машины из общего пула ресурсов, и освобождать эти ресурсы, когда машина больше не нужна. Тем не менее, некоторые программные продукты или их части могут устанавливаться на физически отдельные компьютеры или на спецоборудование, которое с внешней точки зрения управляет как отдельный компьютер. Как виртуальный, так и физический компьютер имеет в сети идентификацию по имени (domain name) или IP-адресу.

Для упрощения для такого идентифицированного объекта будем использовать понятие "хост". При этом для различных операций в системе хост может иметь несколько вариантов адресации.

Для однозначности в качестве идентификации будем использовать ту адресацию, через которую выполняются операции системного администрирования. IP-адреса могут выделяться динамически при запуске хоста, но для серверного оборудования обычно адрес назначается статически при включении хоста в определенную подсеть, чтобы исключить ошибки передачи данных, связанные с различием времени определения адреса по имени и временем отправки пакета. IP-адрес хоста определяется на самом хосте, в то время как доменное имя - на хосте, где находится сервис доменных имен - dns.

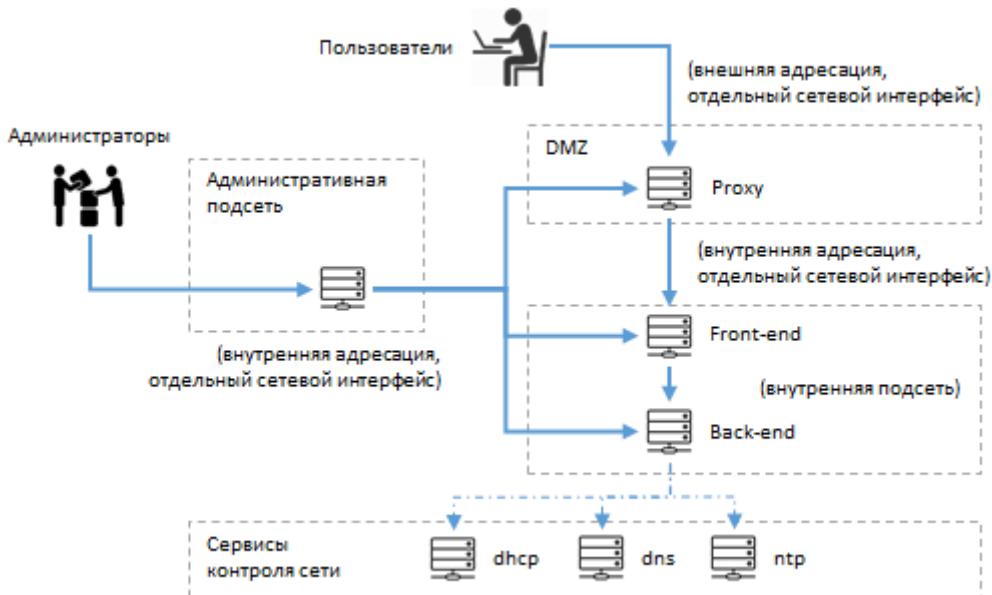


Рис. 154. Организация защищенной среды выполнения

Хост является частью иерархии инфраструктуры - он находится по основной идентификации в определенной подсети определенного dataцентра. Для замены хоста производят одно из:

- Старый хост останавливается и поднимается новый с тем же самым именем и IP-адресом. Обычно это происходит, если хост стал неработоспособен или если процессы хоста кластеризованы и другие части кластера автоматически возьмут на себя нагрузку данного хоста. Зависящие от хоста процессы других хостов должны автоматически повторно открыть соединения (reconnection) без явного перезапуска процессов администратором. Это поведение считается стандартной обязательной логикой для промышленного использования в распределенной системе.
- Старый хост обслуживает существующий поток, новый хост поднимается с новым IP-адресом, после чего в dns для имени хоста меняют целевой IP-адрес на новый. После изменения старый хост останавливают, что приводит к переподключению текущих соединений. Тем не менее, получение нового адреса может в отдельных случаях автоматически не выполниться и необходимо проверить статус операций зависимых процессов и возможно, перезапустить их. Разумеется, при использовании явных IP-адресов в конфигурационных файлах зависимых процессов этого будет недостаточно и необходимо сначала изменить значения, а затем перезапустить процессы.

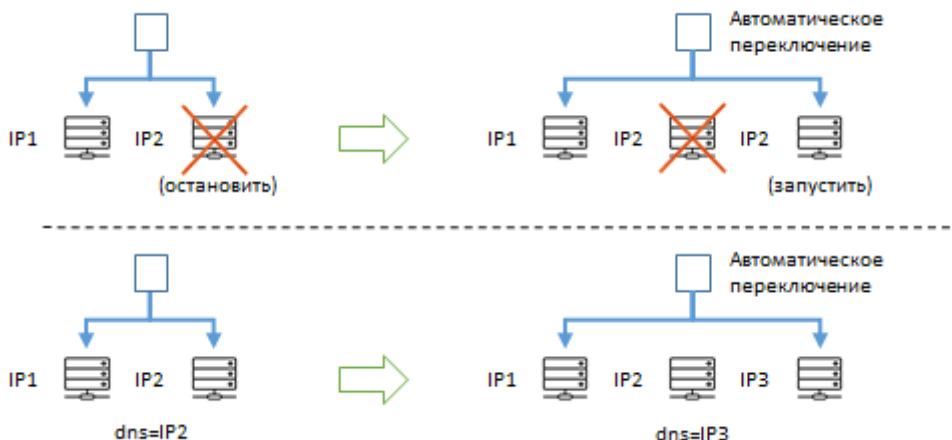


Рис. 155. Замена хоста с ограничением кластера и через dns-сервис

Иерархия инфраструктуры строится независимо от иерархии прикладных систем, так как один и тот же хост может содержать процессы разных программных продуктов. Для удобства стоит ограничить разнообразие и запретить конфигурацию, где один сегмент среды размещается более чем в одном датацентре, или где на одном хосте размещены процессы более чем одной информационной системы.

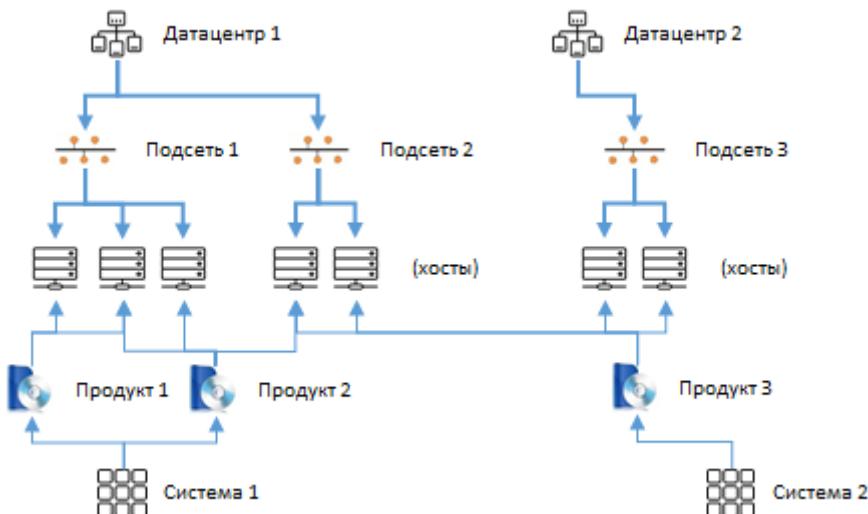


Рис. 156. Размещение программных продуктов на инфраструктуре

Аккаунт - это обобщенная учетная запись пользователя в операционной системе хоста. Пользователь проявляется в операционной системе как владелец объектов или субъект прав, который имеет доступ для выполнения каких-либо операций.

С точки зрения серверных систем, как правило, пользователи связаны не с личными "человеческими" учетными записями, а со служебными аккаунтами, под которыми процесс приложения был установлен или под которыми он был запущен. И здесь есть два варианта, которые фактически используются:

- Установка производится под суперпользователем - учетной записью, которая соответствует всему хосту (root, Administrator, Local System Account), а запуск - под аккаунтом с ограниченными правами. В этом случае предполагается, что только администратор хоста может что-то на него устанавливать, а сама программа не должна иметь лишних прав. Т.е.

администрирование программного продукта нельзя будет на уровне прав отделить от администрирования операционной системы, что тоже не очень хорошо, поскольку именно на уровне суперпользователя для хоста определяются политики компании. Еще одним минусом (проявляется часто в Linux) является то, что при обращении к собственным файлам и каталогам приложение может получить отказ в доступе, поскольку эти файлы принадлежат пользователю, под которым выполнялась установка.

- Суперпользователь создает учетную запись приложения, определяет для нее политику (группы, пароль и т.д.) и домашний каталог, а установка и запуск выполняются под отдельной учетной записью в домашнем каталоге. В этом случае все технические операции с программным продуктом могут выполняться отдельным специалистом и заведомо изолированы от системных прав. При выполнении программы гарантированно не сможет получить права суперпользователя, а также нет конфликта между правами на файлы и на выполнение. Такой вариант является наиболее предпочтительным и рекомендуемым к использованию.

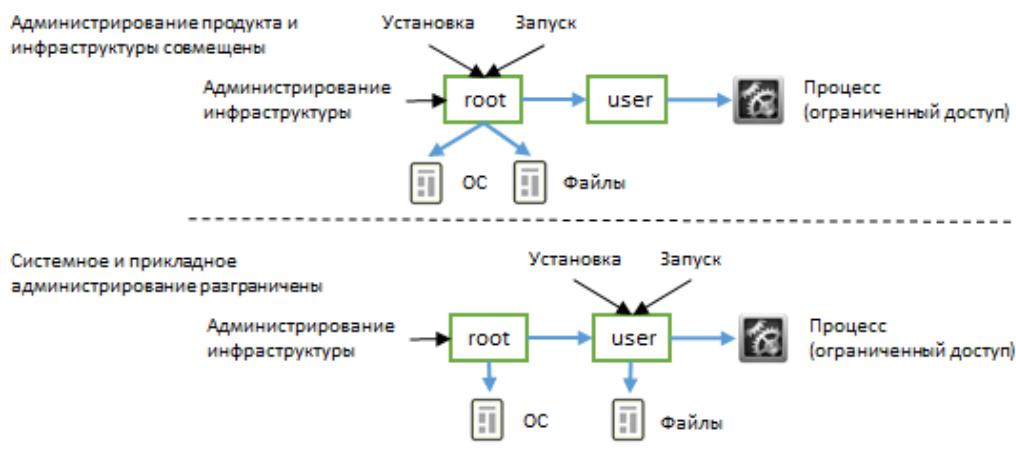


Рис. 157. Системное и прикладное администрирование

В итоге иерархия инфраструктуры расширяется еще одним уровнем аккаунтов, а конкретный прикладной процесс локализован внутри одного аккаунта, при помощи которого выполняется также и обновление программного продукта для данного процесса.

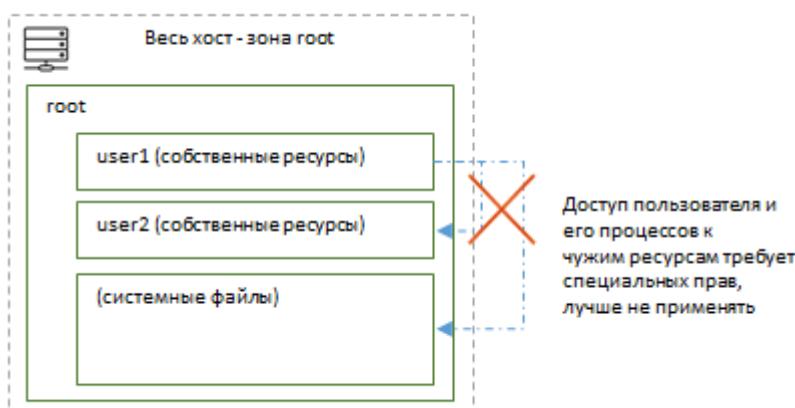


Рис. 158. Аккаунты хоста

Функциональный сервер

Итак, группа приложений (возможно из одного приложения), работающая в рамках одного процесса сервера приложений или одно приложение, работающее само по себе без сервера приложений, образует прикладной процесс (возможно с дочерними подпроцессами), привязанный к определенному аккаунту определенного хоста инфраструктуры.

Для целей масштабирования и отказоустойчивости, или для тестирования этих свойств данный процесс и такими же приложениями также запущен на других хостах или других аккаунтах данного хоста, образуя **кластер** (запуск двух процессов одного кластера под одним аккаунтом одного хоста не рекомендуется и не рассматривается). Для управления кластером может использоваться дополнительный административный процесс без приложений или с приложениями, связанными с управлением (например, Admin-процесс в WebLogic), который тоже является частью кластера.



Рис. 159. Функциональный сервер

Данный кластер связан с конкретным составом приложений и таким образом реализует некоторую функцию (набор функций). Обращение к кластеру происходит через один из следующих механизмов:

- явное указание списка узлов, с которыми можно соединяться (например, в конфигурационном файле клиента HornetQ)
- использование дополнительного административного процесса базового ПО для маршрутизации соединений (Oracle listener)
- использование дополнительного функционального сервера для маршрутизации конкретного технологического протокола (например, маршрутизация веб-запросов через nginx)
- использование дополнительного функционального сервера для маршрутизации общего плана по IP-адресам (например, keepalived+pacemaker)
- наличие только одного активного процесса в кластере, который обрабатывает запросы, и один или несколько пассивных процессов, которые решают задачу отказоустойчивости, но не масштабируемости (например, Postgres)



Рис. 160. Основные виды маршрутизации в кластере

Кластер можно назвать **функциональным сервером**, поскольку с точки зрения потоков данных именно он является узлом системы, который выполняет определенную функцию.

Такое представление удобно, поскольку позволяет описать сегмент среды как линейный список функциональных серверов (конфигурацию процессов) и отобразить диаграмму взаимодействия, которая останется верной вне зависимости от инфраструктуры и конфигурации кластеризации. Это позволяет задать систему идентификации функциональных серверов вне зависимости от среды.

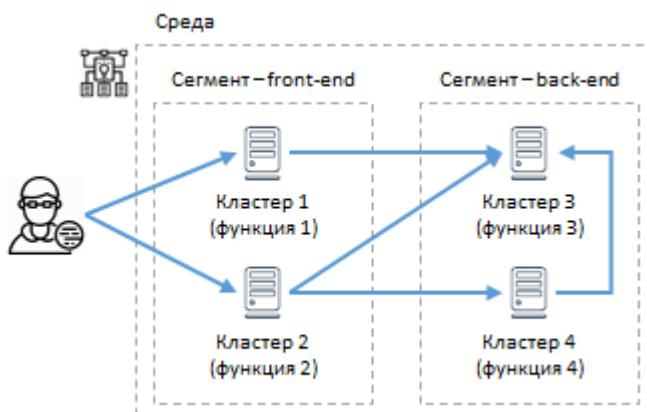


Рис. 161. Взаимодействие функциональных процессов в среде

Также это позволяет сравнивать две среды одного продукта и автоматически определить как элементы соответствия, так и элементы несоответствия. Несоответствие может иметь объективную причину, но в целом среды одного продукта должны быть подобны друг другу.

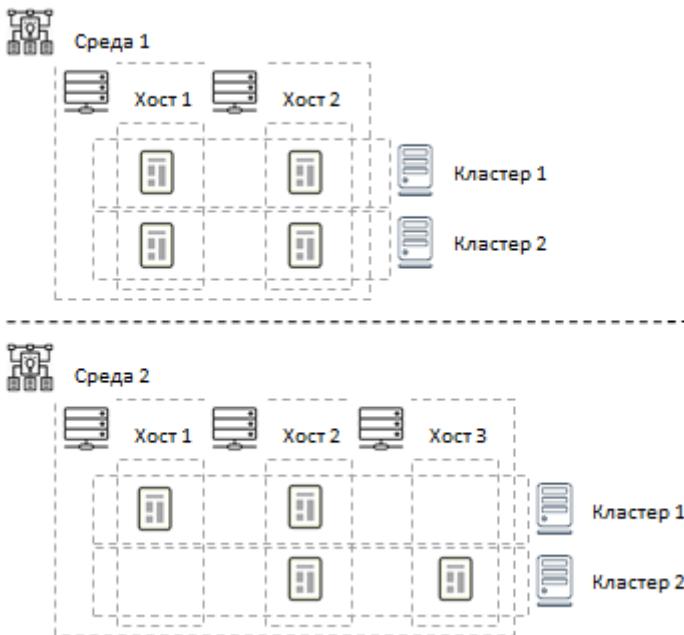


Рис. 162. Возможные варианты размещения на инфраструктуре

Варианты конфигурации процессов позволяют описать варианты развертывания среды вне зависимости от конфигурации кластеризации и конкретной инфраструктуры.

Для определения функционального сервера в среде нужно зафиксировать:

- параметры конфигурации сервера и их значения
- операционную систему и способ контейнеризации - механизм, который позволяет управлять сервером
- элементы дистрибутива, которые должны быть установлены на сервер (или применены к базе данных, если это сервер БД)
- идентификация процессов на инфраструктуре - список пар (хост, аккаунт)
- референтный сервер - одну из сред продукта можно использовать как эталонную и каждый функциональный сервер сопоставлять одному из серверов эталонной среды

Если указан **референтный сервер**, то предполагается, что параметры конфигурации, операционная система, способ контейнеризации и элементы дистрибутива совпадают в данной и эталонной средах, а значения параметров конфигурации и идентификация процессов на инфраструктуре различаются.



Рис. 163. Сравнение с эталонным сервером

Узлы сервера

Повторим, функциональный сервер соответствует кластеру, выполняющему определенную функцию, а не хосту. Кластер состоит в общем случае из нескольких функциональных узлов и одного управляющего узла. Практически встречаются следующие конфигурации кластера:

- **Вырожденный кластер** - из одного узла. Кластером мы его называем для общности, поскольку ни масштабируемостью, ни отказоустойчивостью такой сервер не обладает.
- **Вспомогательный кластер** - когда для каждого функционального узла некоторого кластера есть дополнительный процесс. Эти процессы образуют другой кластер, дополняющий первый, но запросы маршрутизируются строго от одного узла одного кластера к одному узлу другого кластера. Примером вспомогательного кластера является проксирование запросов к веб-серверу (например, WildFly) через nginx с отдачей статики на последнем. При этом на каждом хосте с процессом WildFly запущен процесс nginx.
- **Активно-пассивный кластер** - когда только один сервер принимает запросы, а второй находится в stand-by режиме, и готов заменить первый, если с ним что-то случится. Например, так работает Postgres.



Рис. 164. Роли в кластере

- **Готовый (native) кластер без управления** - когда кластеризация обеспечивается соответствующим базовым ПО и предусматривает взаимодействие узлов с передачей данных (например, открытых сессий), когда нет управляющего узла и функциональные узлы сами контролируют взаимодействие в кластере.

- **Готовый кластер с управлением** - когда в кластере есть управляющий узел, который, например, может выполнять установку новых версий приложений на весь кластер в "горячем" режиме, без остановки сервиса.
- **Фактический кластер** - когда каждый узел кластера работает сам по себе, но при помощи маршрутизации (например, через nginx) узлы входной поток распределяется между ними.



Рис. 165. Роли в кластере (продолжение)

Это порождает следующие типы узлов:

- Функциональный узел - **master**, случай активно-пассивного кластера, для активного узла.
- Функциональный узел - **slave**, случай активно-пассивного кластера, для пассивного узла.
- Функциональный узел - **self**, случай фактического кластера, для любого узла, готовый кластер с управлением или без для функционального узла, вырожденный кластер или вспомогательный кластер.
- Административный узел - **admin**, случай готового кластера с управлением для управляющего узла.

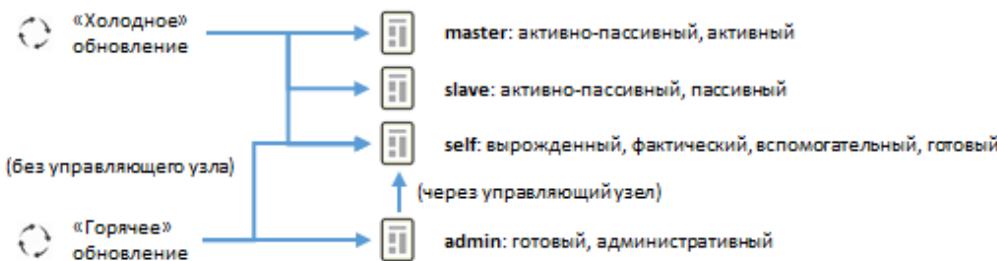


Рис. 166. Обновление узлов кластера

Вид узла и способ обновления влияет на то, какая у него конфигурация:

- Элементы для "холодного" обновления определяются для каждого узла, функционального и управляющего.
- Элементы для "горячего" обновления при наличии управляющего узла могут быть определены только для него, если их установка выполняется через управляющий узел, иначе - только для функциональных узлов.

Размещение элементов

С точки зрения границ программного продукта и его кода, сервер относится к одному из вариантов:

- **Самостоятельный процесс** - состоит из исполняемых файлов, скомпилированных в процессе сборки из собственного кода продукта и, возможно, кода из других продуктов, возможно с дополнительными библиотеками из других продуктов.
- **Процесс на основе базового ПО** - состоит из одного или нескольких продуктов, которые являются платформой для запуска собственного кода продукта. Например, базовое ПО JDK, Tomcat, а прикладное ПО (часть продукта) - war-приложение и конфигурационные файлы к нему.
- **Базовое ПО с определенной конфигурацией** - состоит из базового ПО, конфигурационные файлы которого содержат настройки, определяющие функцию этого сервера в продукте. Обычного кода продукта при этом нет, но в качестве него выступают конфигурационные файлы.

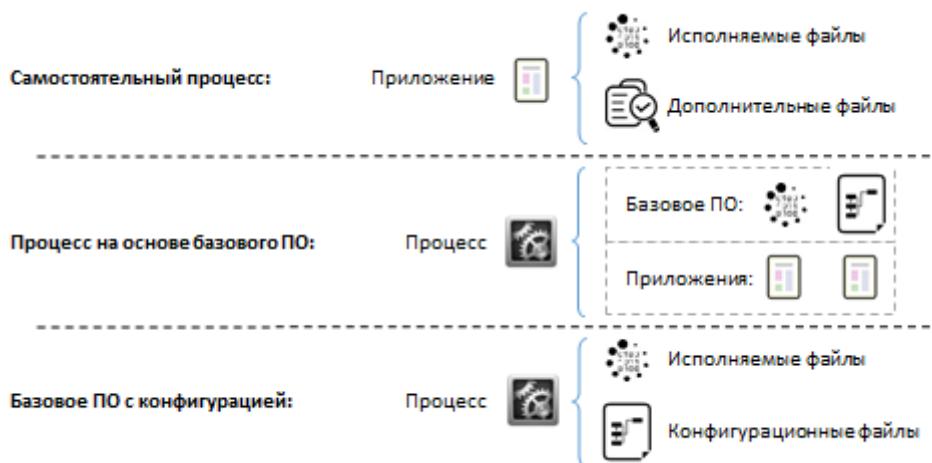


Рис. 167. Варианты процессов в системе

При использовании базового ПО есть некоторая тонкость. Файлы могут быть самостоятельные - полностью созданные как часть продукта, или это файлы базового ПО, которые идут в составе его дистрибутива, но в которые вставляются изменения, которые относятся к продукту. В последнем случае лучше считать, что эти файлы есть в коде продукта и при установке они затирают файлы базового ПО, которые идут по умолчанию.

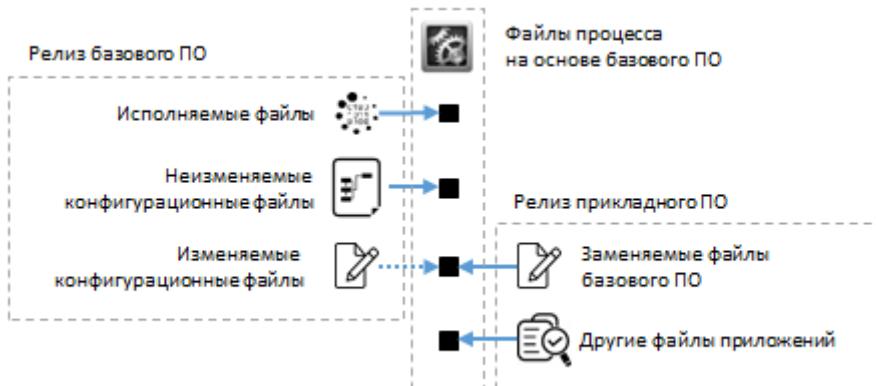


Рис. 168. Разграничение базового и прикладного ПО

Предположим, что на данном хосте есть только один узел одного функционального сервера. Если он установлен в индивидуальный аккаунт, а дистрибутивы у разных продуктов были своими, то достаточно легко осуществить деление содержимого хоста на уровни:

- **Системное ПО** - то, что установлено вне аккаунта сервера, плюс то, что появляется при создании аккаунта и определении политики по отношению к нему.
 - **Базовое ПО** - то, все что установлено в аккаунт сервера, кроме системного ПО и дистрибутива данного продукта
 - **Прикладное ПО** - все, что установлено из дистрибутива данного продукта.
- В общем случае узел сервера представляет из себя следующие группы файлов (системное ПО в состав узла не входит):
- Файлы базового ПО
 - Файлы прикладного ПО (включая модифицированные файлы базового ПО)
 - **Операционные данные**, появляющиеся в процессе работы процесса сервера, включая лог-файлы.

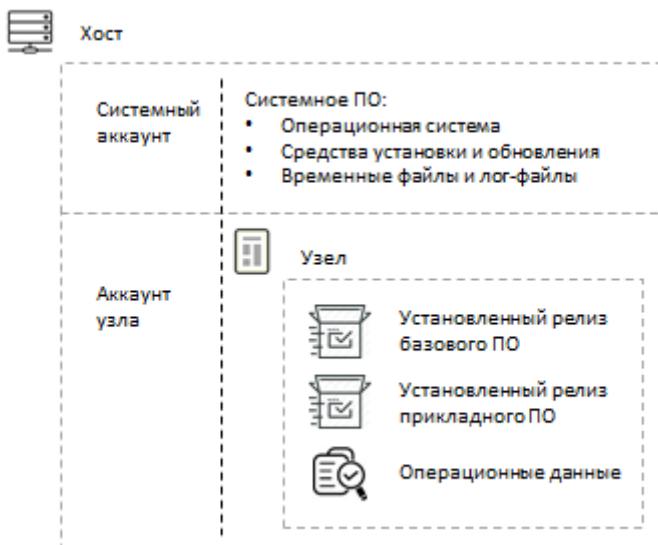


Рис. 169. Структура файлов хоста и узла

В свою очередь хост сервера в общем случае состоит из следующих групп файлов:

- Файлы системного ПО, включая операционную систему, средства установки и обновления, временные файлы и файлы с данными, появляющиеся при работе с системным ПО.
 - Файлы одного или нескольких узлов одного или нескольких серверов
- Теперь разберемся, из чего конкретно состоит прикладное ПО:

Исполняемые файлы из дистрибутива

- Конфигурационные файлы - общие для продукта и одинаковые в разных средах, возможно с индивидуальными значениями параметров конфигурации
- Конфигурационные файлы - в общем случае разные для каждой среды (например, сертификаты)
- Код в специальном формате базового ПО (например, таблицы и хранимые процедуры в базе данных)
- Данные продукта - устанавливаются как часть продукта (например, предопределенный классификатор в форме записей в таблице)

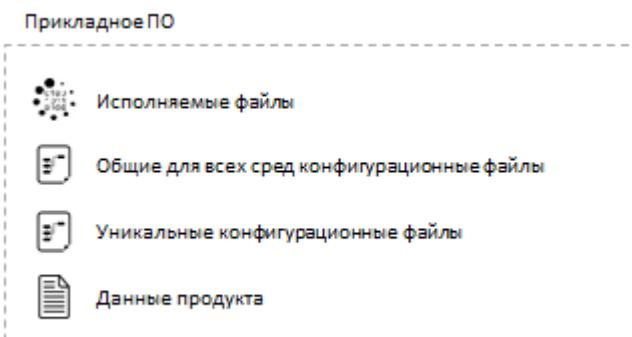


Рис. 170. Структура файлов прикладного ПО

Файлы можно для удобства объединять в устойчивые группы (компоненты), которые размещаются в определенном каталоге сервера или его подкаталоге. Тогда для сервера мы имеем множество каталогов - точек размещения элементов дистрибутива.

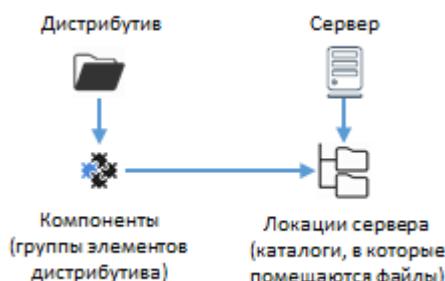


Рис. 171. Размещение элементов дистрибутива в файловой системе

Элементы "горячего" обновления и части баз данных не имеют своего места в файловой системе (даже если они где-то находятся) - это внутренность черного ящика базового ПО. Относя компоненты на спецификацию дистрибутива, а каталоги на спецификацию среды (так как в другой среде каталоги могут быть и другими), мы получаем оптимальное декларативное описание размещения элементов.

Операции в среде

Говоря об операциях в средах, понятно, что операции зависят от продукта и нельзя любой продукт свести к черному ящику с известным набором управляющих элементов. Однако есть так же и противоположная крайность - отказ от выделения одинаковых операций, так что каждый продукт становится произведением искусства со своей индивидуальной инструкцией, терминами и интерфейсами. Все это мешает созданию эффективных процессов эксплуатации продуктов, увеличивает время обучения, усложняет автоматизацию.

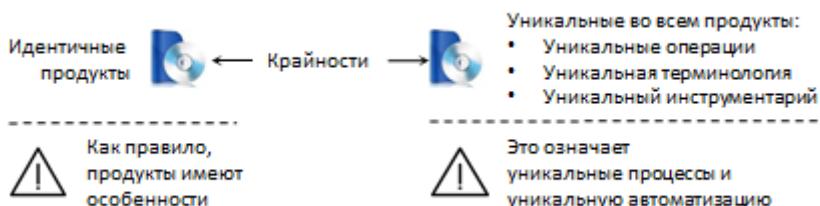


Рис. 172. Крайности при систематизации операций

Несмотря на наличие уникальных особенностей, в среде каждого продукта есть множество операций, которые по сути своей идентичны и можно организовать процесс эксплуатации, в первую очередь отталкиваясь от этих операций. Это позволит определить подход, который даст быструю адаптацию к любому новому продукту. Возможно, для некоторых продуктов уникальные черты таковы, что важно будет строить процесс отталкиваясь в первую очередь от них, но для большинства программных продуктов проще использовать готовый подход на основе единой модели операций (операционной модели), расширив его в части специфики продукта.

Операционная модель может включать следующие виды операций:

- Управление прикладными процессами - узлами серверов (запуск, остановка, контроль состояния)
- Управление кластеризацией - расширение и сокращение размеров кластера
- Управление конфигурационными файлами - изменение значений параметров конфигурации, сохранение конфигурации, восстановление конфигурации.
- Установка релизов - проверка возможности изменений из дистрибутива, выполнение изменений, откат изменений, проверка соответствия среды и изменений, получение информации о текущем состоянии с точки зрения версий
- Установка и обновление базового ПО

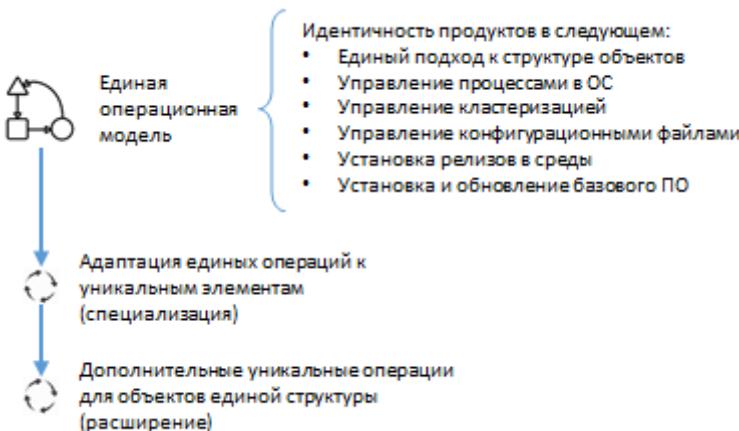


Рис. 173. Единая операционная модель

Тестирование релизов

Чтобы отдать дистрибутив релиза для установки в среды промышленной эксплуатации, команде разработки необходимо получить уверенность, что продукт будет работать после установки из этого дистрибутива. Уверенность не может быть абсолютной, но она должна быть достаточной. Для этого обычно продукт проверяют в некоторой среде, созданной специально для тестирования.

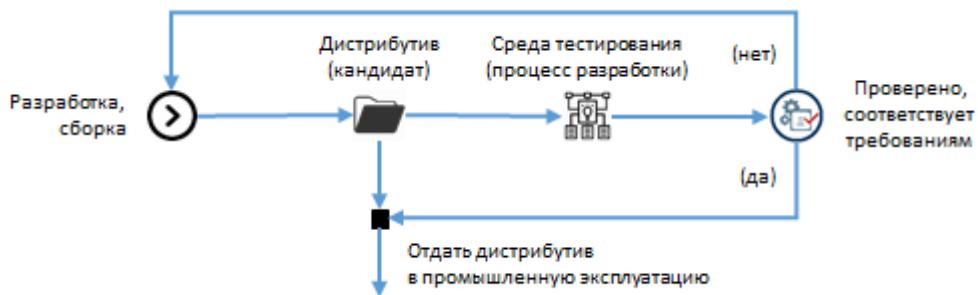


Рис. 174. Принципиальный порядок тестирования релиза

Далее, команда эксплуатации, которая ставит тиражируемый продукт или для которой откат при каких-то проблемах является нормальным и проработанным вариантом, может ставить сразу в среду промышленной эксплуатации, но в иных случаях, особенно для критических сервисов и сложных процедур обновления, специалистам нужно убедиться, что в продукте нет проблем, которые команда разработки упустила по каким-либо причинам, и проверить процедуру обновления, которая для данного релиза может быть уникальной. А если учесть, что процедура обновления применяется на среде промышленной эксплуатации, которая в такой конфигурации единственная, то понятно, что никто эту процедуру и не проверял, а специалисты эксплуатации могут оказаться не готовы к ее уникальным пунктам работ. Иными словами, команде эксплуатации также нужна достаточная степень уверенности в результате и эту уверенность можно получить, выполнив обновление в своей среде тестирования.

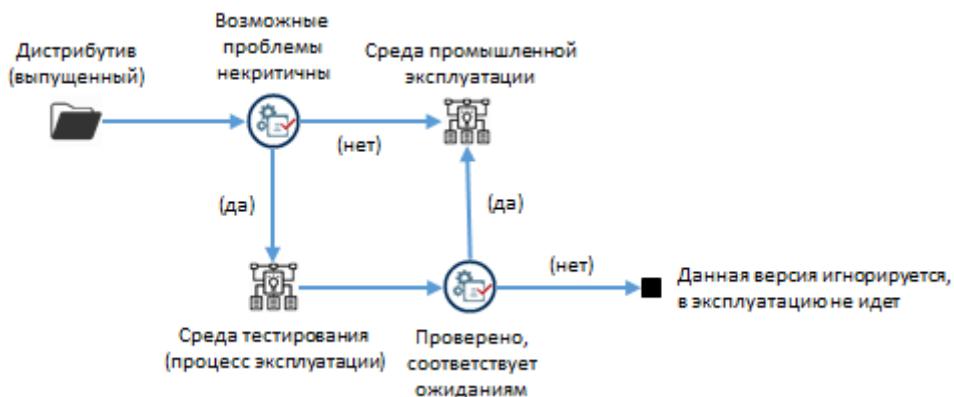


Рис. 175. Принципиальный порядок действий с релизом в процессе эксплуатации

В результате, чтобы все были уверены, получается, что нужны и существенные ресурсы для среды тестирования, и время на выполнение всех этих процедур. Одним из вариантов сокращения является поддержания для выпуска релизов специальной среды тестирования, подобной среде промышленной эксплуатации, доступность ее для специалистов эксплуатации, сведение процедуры обновления к унифицированной и практически полностью автоматизированной операции, и полная открытость процесса выпуска релиза. В этом случае проверки на уровне эксплуатации могут не выполняться. Такая среда выпускающего тестирования может быть как под контролем разработки, так и под контролем эксплуатации.



Рис. 176. Выделенный релизный процесс

Альтернатива, когда релиз выпускается с хорошим контролем на уровне эксплуатации, но с невысоким контролем при выпуске релиза, очевидно хуже:

- При выпуске релиза выполняется тестирование серого ящика, когда разработчики и тестировщики имеют представление, как устроен код и куда стоит смотреть, чтобы найти ошибку. Эксплуатация не сможет обнаруживать некоторые ошибки или потратит на это больше времени.
- Раннее обнаружение и исправление ошибки будет означать существенное снижение и общих затрат и времени в релизном цикле.



Рис. 177. Выпуск релиза силами эксплуатации

Для того, чтобы тестирование означало хороший контроль, необходимо:

- **Эквивалентность сред тестирования и промышленной эксплуатации.** Поскольку среды разные, то в чем-то они неизбежно будут отличаться, однако это должны быть пункты, вероятность влияния которых на результат тестирования мала.
- **Выполнение оптимального для данного релиза набора тестов.** В любом релизе будет изменение чего-либо и части, которые не меняются. Если всегда выполнять полное тестирование, то при релизах с короткими циклами затраты будут неприемлемыми. Поэтому тесты должны быть выборочными и хорошо подобранными.
- Наконец, выполнение **тестирования должно быть добросовестным**. Но необходимо учесть, что как разработчик, так и тестировщик, оба совершают ошибки. И единственный вариант, когда это становится практически неважным, если разработчик проверяет свой код в ходе отладки, а тестировщик – повторно в ходе независимого тестирования. Вероятность, что они в одном месте пропустят ошибку, достаточно мала, чтобы иметь достаточную уверенность в качестве результата (принцип 4-eye).

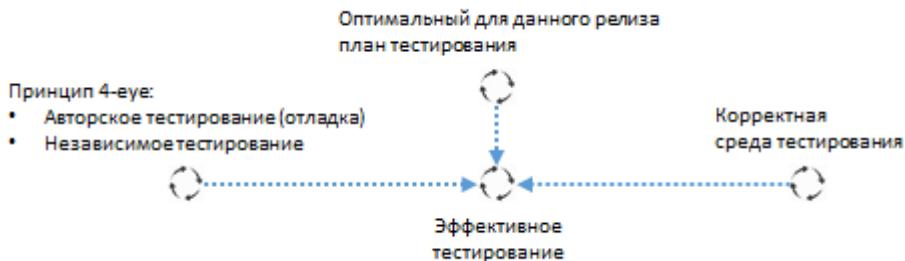


Рис. 178. Эффективное тестирование

Создание сред для тестирования

Как создать и поддерживать среду для тестирования, эквивалентную среде промышленной эксплуатации?

Сложности такой среды в следующем:

- Продукт не является обычно самодостаточным, и должен быть подключен к средам смежных продуктов, однако таких сред может не быть или они в нужный момент могут не работать
- Мощность среды промышленной эксплуатации может быть настолько высокой, что для целей тестирования таких ресурсов могут не выделить и если при тестировании функций это может быть неважным, то при тестировании производительности метрики становятся несравнимы с аналогичными в среде промышленной эксплуатации
- При тестировании обнаруживаются ошибки и продукт дорабатывают, изменения могут исключаться из релиза - все это приводит к тому, что нужно в каждой итерации откатывать релиз, что для сложных продуктов, особенно с большими базами данных может быть неприемлемо, что означает нарушение эквивалентности сред
- При большом количестве инкрементальных релизов высока вероятность того, что так или иначе постепенно накопится разница, которая нарушит эквивалентность
- Изменения, которые производит эксплуатация в среде промышленной эксплуатации, и которые не связаны с релизами - могут не быть известны тем, кто поддерживает среду тестирования
- Уникальные файлы и настройки (например, сертификат, который имеет срок действия) могут создать отличия в поведении приложений

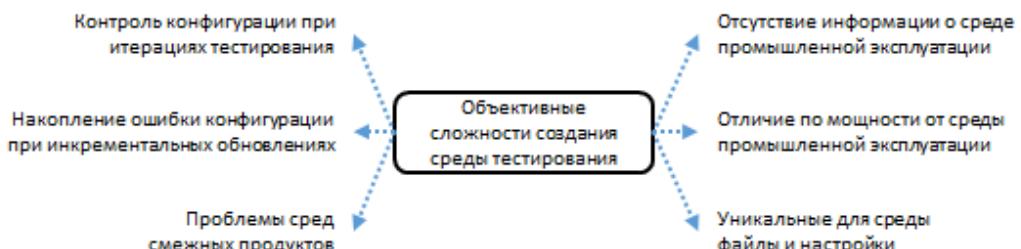


Рис. 179. Сложности создания сред для тестирования

Возможный рецепт для создания и поддержки такой среды (модель чистой комнаты) следующий:

- Наличие и поддержание эталона продукта - полного дистрибутива и полного комплекта конфигурационных файлов в виде шаблонов с переменными в месте значений параметров конфигурации, что позволит при наличии средства автоматизации пересоздать всю или часть среды для выполнения новой итерации.
- Наличие эталона базы данных - копии базы данных в виде дампов из среды промышленной эксплуатации, возможно полностью или частично без операционных данных. Здесь важно

отметить, что наличие операционных данных позволяет тестировать продукт на реальных данных, что повышает качество тестирования. Поскольку изменения базы данных носят большей частью инкрементальный характер, то попытка вести эталон на основании дистрибутивов неизбежно приведен к расхождениям.

- Регулярное полное восстановление среды из эталона продукта и эталона базы данных. Регулярное обновление эталона базы данных. Автоматизация этих процессов.
- Автоматизация установки релиза
- Автоматизация сверки релиза и среды, эталона продукта и среды
- Поддержка идентичности конфигурации процессов
- Использование минимальной конфигурации кластеризации
- Измерение коэффициентов подобия для оценки параметров производительности
- Автоматизация формирования данных для тестирования.
- Для решения вопросов интеграционного тестирования необходимо определить принципы взаимодействия сред по типам - к средам промышленной эксплуатации подключаться нельзя, контролируемые среды могут зависеть только от контролируемых, отсутствующие продукты необходимо заменять эмуляторами, отсутствующая интеграция должна документирована, чтобы заранее знать, какие ошибки в данной среде не являются ошибками продукта, а являются следствием принятой к использованию конфигурации.



Рис. 180. Модель чистой комнаты

Закрытые данные и данные для тестирования

Сертификаты и пароли, персональные данные, а также другие данные, которые специально выделены как закрытые для данного продукта, являются закрытой информацией, которая не может из среды промышленной эксплуатации попадать куда-либо, например, в среду тестирования. Тем не менее, необходимо тестировать соответствующие операции, что требует выполнения следующих операций:

- Сертификаты и пароли для среды промышленной эксплуатации должны храниться в защищенном месте, откуда невозможно копирование кем-либо, кроме авторизованного администратора, и либо не помещаться в репозиторий контроля версий, либо данный репозиторий должен находиться на выделенном для этих целей экземпляре сервера системы контроля версий
- Для использования в тестовых средах должны существовать свои пароли и сертификаты, которые не должны по умолчанию копироваться из среды промышленной эксплуатации

- Сертификаты и тому подобные файлы желательно иметь с названиями, которые отражают их принадлежность среде. Названия таких файлов и пароли следует выносить в параметры конфигурации среды.
- При создании копии базы данных среды промышленной эксплуатации для тестовой среды необходимо выполнять процедуру деперсонификации, которая либо удаляет персональные данные, либо модифицирует их так, чтобы оригинал стало невозможно получить.

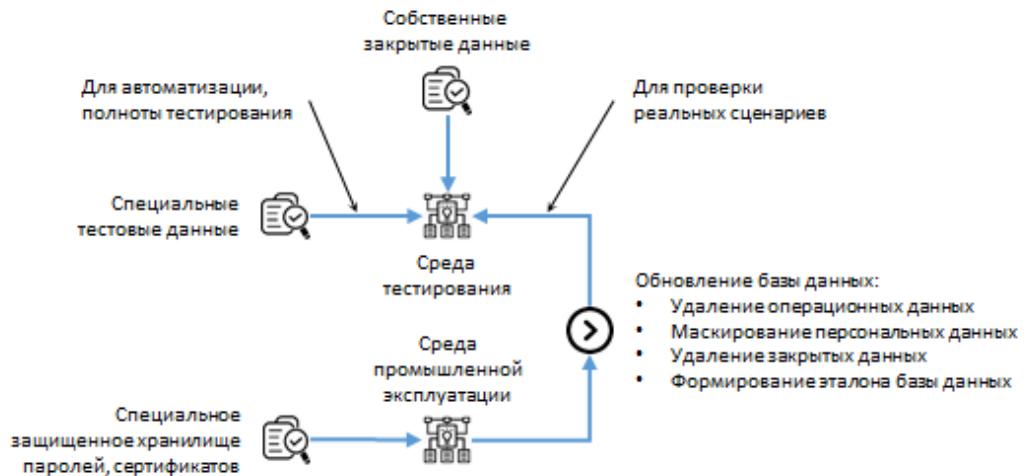


Рис. 181. Управление закрытыми данными

Данные из промышленной базы данных определяют основные тестовые сценарии, однако при написании средств автоматического тестирования и для тестирования случаев, которые должны поддерживаться, но пока не были зафиксированы в промышленных операциях, а также для тестирования ситуаций, требующих много предварительных действий и настроек, удобно при помощи какого-то скрипта вносить данные непосредственно в базу данных после обновления из среды промышленной эксплуатации, так чтобы эти данные для тестирования были автоматически готовы для использования.

Установка изменений

ПРЕДМЕТ И ПАКЕТ ИЗМЕНЕНИЙ

ИЗМЕНЕНИЯ В ИНФРАСТРУКТУРЕ

ИЗМЕНЕНИЯ В ОПЕРАЦИОННОЙ СИСТЕМЕ

ИЗМЕНЕНИЯ В БАЗОВОМ ПО

ИЗМЕНЕНИЯ В ПРИКЛАДНОМ ПО

ПРОЦЕДУРЫ ЗАПУСКА И ОСТАНОВКИ ПРОЦЕССОВ

ВРЕМЯ НЕДОСТУПНОСТИ СЕРВИСА

ХОЛОДНОЕ И ГОРЯЧЕЕ ОБНОВЛЕНИЕ

ДОВЕДЕНИЕ ОБНОВЛЕНИЯ ДО КОНЦА

ОБЕСПЕЧЕНИЕ ВОЗМОЖНОСТИ ОТКАТА ИЗМЕНЕНИЙ

УСТАНОВКА ИНКРЕМЕНТАЛЬНЫХ РЕЛИЗОВ

ИЗМЕНЕНИЕ КОНФИГУРАЦИОННЫХ ФАЙЛОВ В ПРИЛОЖЕНИЯХ

ОБНОВЛЕНИЕ ЗАКРЫТЫХ ДАННЫХ В ПРИЛОЖЕНИЯХ

ОПЕРАЦИИ ИЗМЕНЕНИЯ БАЗ ДАННЫХ

ИЗМЕНЕНИЕ СХЕМ И ЭКЗЕМПЛЯРОВ БАЗ ДАННЫХ

СКРИПТЫ ОТКАТА ИЗМЕНЕНИЙ БАЗ ДАННЫХ

ИЗМЕНЕНИЕ ПАРАМЕТРОВ КОНФИГУРАЦИИ В БАЗАХ ДАННЫХ

Что именно изменяется в среде? Как связаны изменения в среде и релизы, что такое внедрительные изменения? Как выполнять изменения во взаимодействующих продуктах?

Предмет и пакет изменений

Как было рассмотрено выше в главе про то, что из себя представляют среды, среда состоит из элементов дистрибутива программного продукта, размещенных на элементах инфраструктуры - хостах, элементов установленных платформенных продуктов, которые составляют технологический стек, настроек (значений параметров) конфигурации, которые определяют специфику данной среды в отличие от других сред этого же продукта или взаимодействие со средами связанных продуктов, и операционных данных.

Одним целевым изменением назовем изменение, которое переводит одну среду промышленной эксплуатации одного программного продукта из текущего состояния в новое, ожидаемое с точки зрения заказчика состояние. В данном состоянии что-то может не работать, но это должно быть приемлемо в режиме эксплуатации.

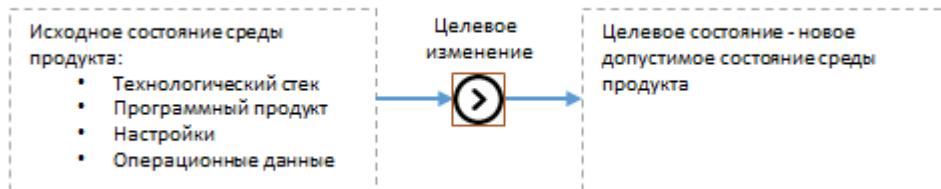


Рис. 182. Целевое изменение

Одно целевое изменение состоит из отдельных технических изменений, каждое из которых также меняет продукт, но состояние после такого изменения является неприемлемо с точки зрения эксплуатации.

Предмет одного технического изменения является один или несколько элементов конфигурации среды программного продукта. Предмет целевого изменения получается объединением предметов технических изменений.

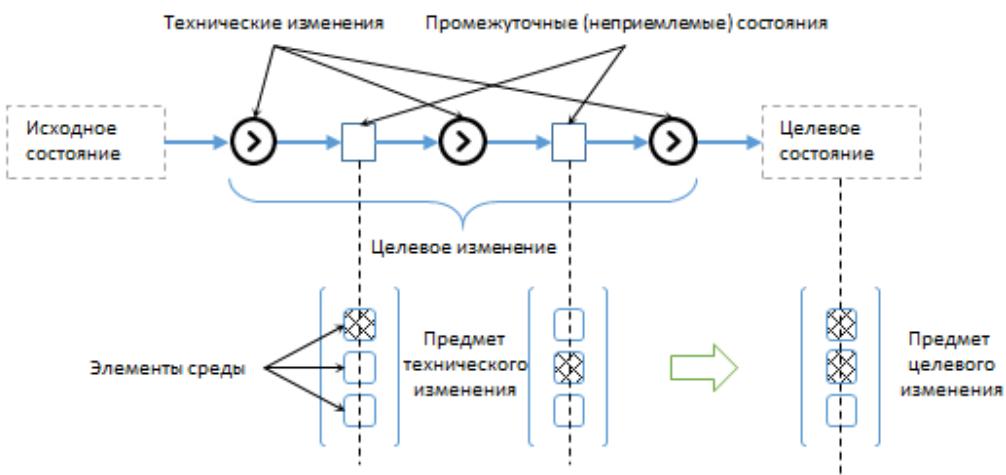


Рис. 183. Технические изменения и промежуточные состояния, предмет изменений

Для оптимизации работ несколько целевых изменений могут быть объединены в пакет, который выполняется по совместному плану. Отдельное целевое изменение может быть вызвано релизом программного продукта (релизное изменение) или быть следствием решений в процессе эксплуатации, направленных на замену неисправного оборудования, изменения мощности, изменения существующих параметров конфигурации, которые никак не связаны с изменением в программном продукте (внеризлзное изменение).

Смешивать релизные и внорелизные изменения в одном пакете не стоит, поскольку внорелизные изменения оцениваются исходя из состояния продукта, соответствующего определенному релизу. Но если одновременно выводится новый релиз, это означает, что внорелизные изменения оценивались исходя из заведомо другого состояния продукта, что может привести к проблемам.

Предмет изменения пакета изменений является объединением предметов отдельных целевых изменений.

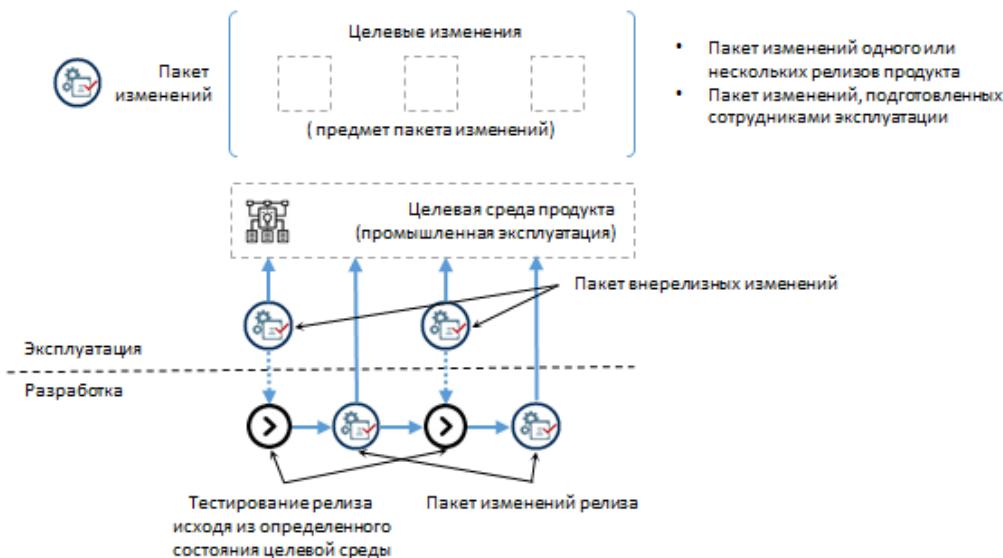


Рис. 184. Релизные и внорелизные изменения

Поскольку среда продукта состоит не только из элементов дистрибутива продукта, но и из базового (платформенного) и системного ПО (операционная система хостов и ее компоненты), настроек конфигурации, то пакет изменений релиза может включать кроме изменений продукта, также и требуемые изменения в базовом и системном ПО и настройках конфигурации. Это означает, что пакет изменений релиза продукта связан не только с самим продуктом, но и с его окружением.



Рис. 185. План изменений релиза

В случае, когда релиз одного продукта требует также и релиза другого продукта (коррелированные релизы), пакет изменений содержит несколько релизных изменений, относящихся к разным продуктам. Эти изменения выполняются в определенном порядке, который описан в плане установки пакета изменений. Для устойчивости процесса обновления и снижения рисков желательно сделать эту зависимость односторонней - т.е. так, чтобы можно было установить один продукт, но не устанавливать другой продукт, иначе при неудачной установке второго продукта придется также откатывать установку первого и релизы второго продукта начинают мешать релизам первого, а время неприемлемого состояния (downtime) вырастает до суммарного времени обновления продуктов.

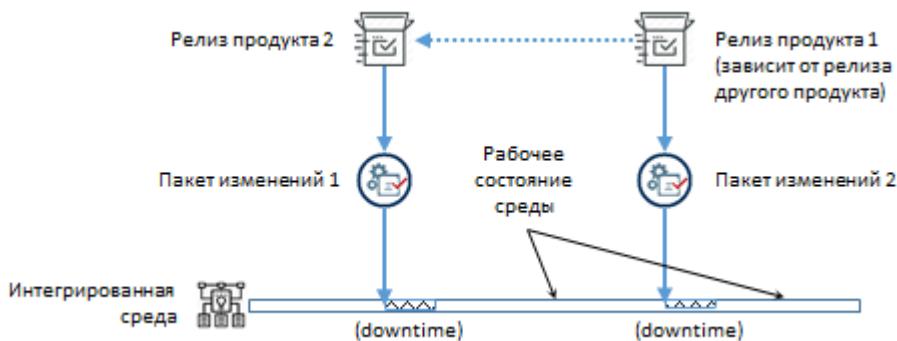


Рис. 186. Коррелированное изменение продуктов

План выполнения пакета изменений необходимо выстраивать по следующим принципам:

- все подготовительные изменения, которые не приводят к изменению работающих приложений (например, создание виртуальной машины для нового приложения), необходимо выполнять заранее, минимум за один день до выполнения изменений в приложениях
- подготовительные действия могут выполняться в разные дни, разными людьми, в зависимости от ситуации
- основные операции, выполняющие изменение области работающих приложений, выполнять как одну транзакцию, стараясь выполнить ее как можно быстрее
- после основных операций могут также быть запланированы действия, которые не влияют на работающие приложения (например, удаление виртуальной машины, используемой выведенным из эксплуатации приложением)

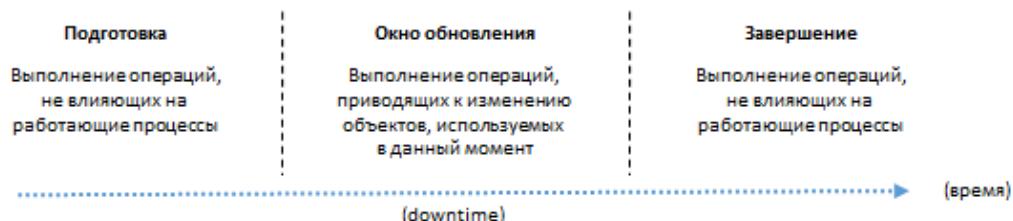


Рис. 187. Структура действий по установке релиза

Предмет изменений нам интересен тем, что в большой системе, если нет специальных причин, мы при обновлении будем менять и перезапускать только часть процессов, соответствующую предмету изменений, сокращая и время операций и связанные с ними риски, а часто и позволяя обновление без приостановки сервиса.

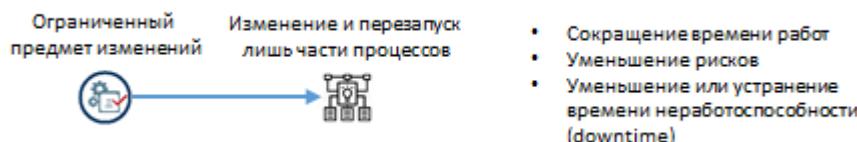


Рис. 188. Оптимизация при обновлении сложных систем

Изменения в инфраструктуре

Слово инфраструктура не является строгим термином и может обозначать разное в зависимости от контекста. Одни относят к инфраструктуре только оборудование и коммуникации, другие - включают туда системное ПО, а третьи - и базовое ПО, например, СУБД.

В рамках данной книги под инфраструктурой понимается оборудование, средства виртуализации, сети и унифицированные, созданные для использования в разных продуктах образы операционной системы. Средства виртуализации сегодня настолько распространены, что практически стандартной практикой стало использование корпоративных образов операционных систем, так что установка новой машины выполняется в одно действие с интерфейса системы виртуализации и занимает несколько секунд.

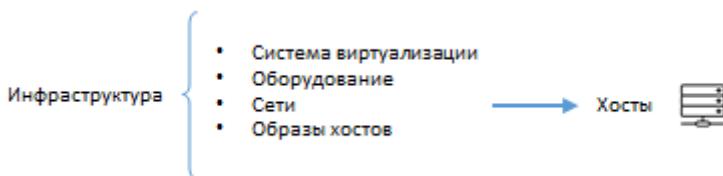


Рис. 189. Состав инфраструктуры

Образы операционной системы также включают в себя набор настроек и патчей, которые определяются корпоративной политикой безопасности. Если образы не используются, то неизбежно для относительно большого парка ИТ-систем возникает неуправляемая ситуация, когда никто не знает, какие уязвимости устранены в системе, а какие нет. Обновление же операционных систем превращается в рискованную и в длительную процедуру без четкой логики.

Поэтому стандартным подходом в ИТ-компаниях является выбор нескольких вариантов эталонных образов, смена их время от времени для обновления версий операционных систем и закрытия проблем безопасности, и выстраивание планов обновления инфраструктуры работающих приложений.



Рис. 190. Использование типовых образов

Разумеется, нельзя при каждом выпуске патча безопасности обновлять все системы путем замены образов, поскольку это хотя и наиболее гарантированная форма, но и затратная, требующая отключения сервиса. Также иногда это просто невозможно, если контроль над программным продуктом утерян и невозможно быстро и надежно установить текущую версию продукта на новую чистую машину.

Такие продукты образуют называемые устаревшие ("legacy") системы, которые с каждым годом отстают по технологиям от современных платформенных решений, и вносить изменения в которые уже становится невозможным. В результате на хосты с работающими приложениями накатываются патчи без смены образов.

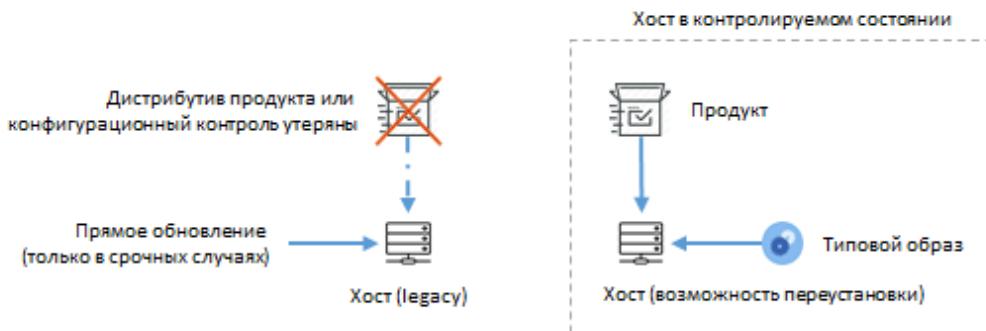


Рис. 191. Контролируемые хосты и неподдерживаемые (legacy) системы (без учета операционных данных)

При этом необходимо различать людей, которые занимаются инфраструктурой и поддерживают, например, реестр образов для виртуальных машин, и саму инфраструктуру, поскольку также весьма распространенным вариантом является включение в образ типичного базового ПО - например, образ Linux CentOS 6.6 с предустановленным PostgreSQL.

В этом случае администраторы прикладных систем или специалисты по СУБД готовят эталонную машину, а системные инженеры, отвечающие за инфраструктуру, снимают с эталонной машины образ для помещения в реестр стандартных образов. Тем не менее к инфраструктуре данное базовое ПО не относится, поскольку за СУБД в составе образа отвечают уже другие специалисты.

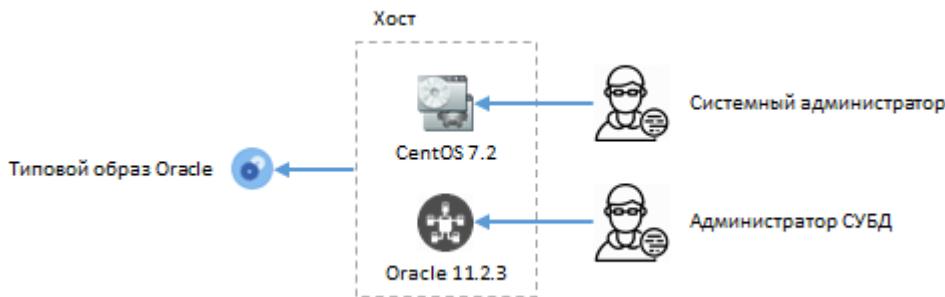


Рис. 192. Создание типовых образов для определенных задач

Еще один аспект унификации в образах - установка опциональных элементов операционных систем, создание структуры файловой системы и выделение стандартных ресурсов виртуальной машины - процессора, диска, оперативной памяти.

Хотя каждый прикладной сервис уникален и требует специфических для него ресурсов, на деле управление ресурсами отдельно по каждой машине неудобно - порождает то, что называется "микроменеджментом", когда слишком много элементов управления, за которыми вручную или даже с частичной автоматизацией уследить невозможно и когда благие намерения оптимизации и полного контроля порождают прямо противоположный результат.

Благодаря технологиям сверхподписки ("oversubscription") в системе виртуализации, выделенные, но неиспользуемые машиной ресурсы не пропадают, а используются так, что суммарные логически выделенные ресурсы могут, например, в три раза превышать физические ресурсы dataцентра. Это дает возможность технологии использования машин с типовыми выделенными ресурсами - например, 2Гб оперативной памяти, диск 40 Гб, 2 процессорных ядра.

Если какой-то прикладной процесс использует кластер из двух таких машин и нагрузка вырастает так, что логические ресурсы машины целиком почти используются, кластер расширяют еще одним узлом. Если же логические ресурсы существенно недозагружены, их задействует система виртуализации через механизм сверхподписки.

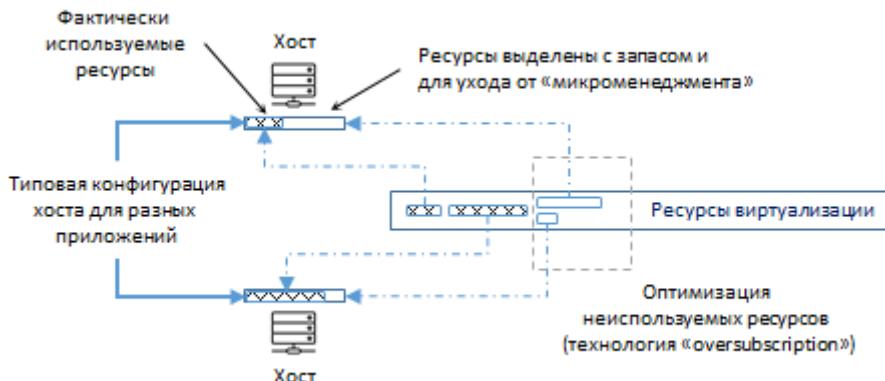


Рис. 193. Систематизация конфигурации инфраструктуры

Изменения в операционной системе

Если даже не говорить про изменения в операционной системе, относящиеся к инфраструктуре, то тем не менее существуют системные пакеты, патчи и разнообразные настройки на уровне операционной системы, которые необходимо установить для работы выбранного базового ПО и конечных приложений.

Речь здесь идет только о тех пакетах и настройках, которые считаются частью операционной системы и контролируются теми, кто отвечает за выпуск версий операционной системы. То, что не относится к версиям операционной системы (например, JDK/Crypto/Pro и т.п.), является базовым или прикладным ПО.

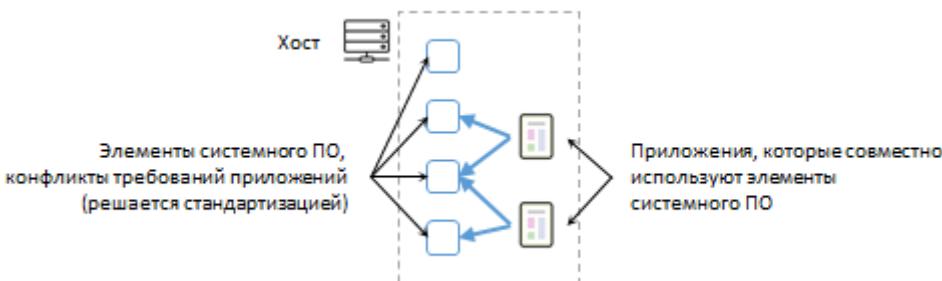


Рис. 194. Зависимости от элементов системного ПО

Данные изменения и файлы являются частью среды программного продукта. Однако важно учесть, что если на одном хосте работают два разных программных продукта, то они могут пересекаться по используемому системному ПО, которое устанавливается как часть их окружения. Поскольку это системное ПО, то оно находится в общей области и не изолировано в аккаунтах приложений. Это порождает конфликты, когда при обновлении одного продукта может возникнуть проблема в другом продукте. Поэтому логичными являются следующие рекомендации:

- избегать использования нестандартного системного ПО, не входящего в типовые корпоративные образы
- избегать размещения нескольких программных продуктов на одном хосте
- если такое неизбежно, всегда анализировать возможное взаимовлияние изменений в таких продуктах и использовать совместные пакеты и планы изменений

Изменения в базовом ПО

Начальная установка базового ПО при наличии инструкций и дистрибутивов как правило не вызывает проблем. В качестве базового ПО обычно используются продукты, имеющие большое количество инсталляций и наработанную базу знаний о том, как лучше устанавливать, какие при этом могут быть проблемы и как их решать. Многочисленные форумы помогают администраторам в конце концов установить и настроить ПО. Однако важно не забывать о следующем:

- Конфигурация и принцип настройки базового ПО должны определяться со стороны разработки, поскольку именно на стадии разработки пришлось разобраться с особенностями базового ПО и тем, как именно оно используется.
- Конфигурация базового ПО в среде разработки и в среде промышленной эксплуатации отличается, поскольку в последней необходимо обеспечивать гарантированный сервис в потоке реальных операций.



Рис. 195. Формирование конфигурации базового ПО

Для установки базового ПО как правило существуют дистрибутивы, созданные их разработчиками. Однако если есть корпоративная политика по дополнительным настройкам, связанным с управлением доступом, аудитом и другими служебными операциями, то вместо повторения настроек при каждой установке и контроля осуществления этих настроек, то выгоднее создать внутренний корпоративный дистрибутив с уже выполненными настройками и использовать его. Более того, даже если изменений во внешнем дистрибутиве нет, правильнее всегда использовать корпоративные дистрибутивы базового ПО, что позволит, не меняя процесса, при необходимости внести нужные изменения.

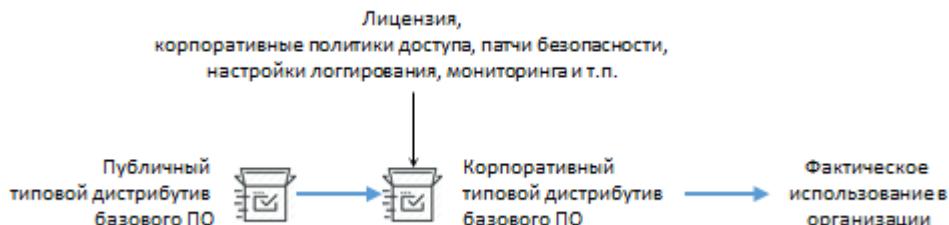


Рис. 196. Корпоративный дистрибутив базового ПО

Обновление базового ПО не должно быть частой операцией, поскольку приложения пишутся с опорой на особенности версии базового ПО, включая иногда и ее ошибки. Изменение версии базового ПО требует в общем случае регрессионного тестирования приложения и частые обновления приносят и лишний риск и лишние затраты.



Рис. 197. Принципы обновления версий базового ПО

Напомним о границе между базовым и прикладным ПО. Прикладное ПО продукта идет в его дистрибутиве, в то время как базовое ПО ставится из своего дистрибутива. Совмещение базового ПО и прикладного ПО в одном дистрибутиве (или, например, контейнере docker) порождает проблему невозможности управления базовым ПО со стороны эксплуатации, независимо от программного продукта, и будет скрывать состав прикладного ПО, а также иногда нарушать лицензионное соглашение.

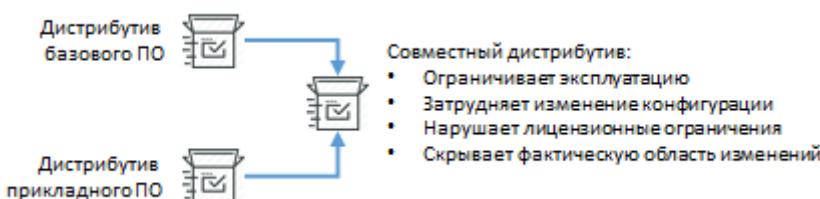


Рис. 198. Проблемы включения базового ПО в состав прикладного ПО

Особо стоит отметить случай, когда в составе базового ПО есть конфигурационный файл, который настраивается под продукт и установка новой версии базового ПО должна эти настройки сохранить, обновив тем не менее какое-то другое его содержание. Поскольку конфигурационный файл для этого и существует, программа обновления обычно это и делает, но необходимо заранее в этом убедиться.

Как ранее определено в главе про среды прикладных продуктов, данный файл в исходной своей версии является частью базового ПО, в как текущее содержание - частью прикладного ПО, так что при установке релиза прикладного продукта данный файл обновляется обычным способом как файл прикладного продукта. Однако если было выполнено обновление базового ПО, которое привело к обновлению данного файла, то его новое содержание необходимо отразить в файле, который находится в репозитории прикладного продукта.

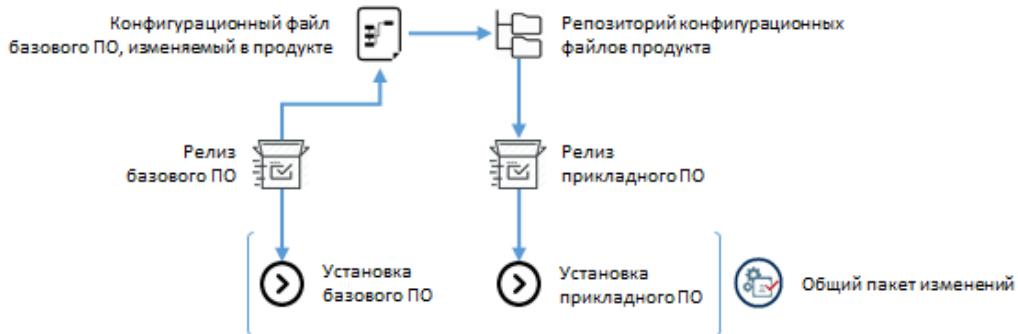


Рис. 199. Изменение базового ПО и его модифицированных конфигурационных файлов

Изменения в прикладном ПО

Прикладное ПО сразу стоит разбить на приложения и базы данных, так как действия будут существенно разные и будут использовать совершенно разные понятия.

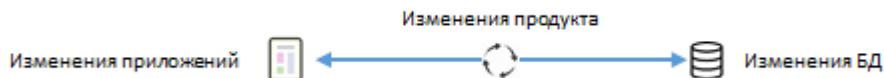


Рис. 200. Отдельные категории изменений

Давайте сначала обозначим то, что делает обновление приложений операциями, к которым надо подходить, хорошо понимая возможные варианты и их последствия, а затем более детально разберем эти особенности ниже:

- Приложения в среде промышленной эксплуатации выполняют конечные операции. А если это система уровня 24x7 и прекращение операций недопустимо, то операции могут и должны успешно выполняться даже в момент обновления. Если же система допускает прекращение операций, то необходимо свести время простоя системы, то есть прекращения работы одного или нескольких ее сервисов ("downtime") к минимуму.
- Для обновления приложений без остановки сервиса часто применяется так называемое "горячее обновление". Однако если это не нужно, если непрерывность сервиса реализуется другим способом или если обновление по техническим причинам нельзя выполнить в горячем режиме, то применяется процедура с остановкой процесса - "холодное" обновление
- Как описано в главе про среды, среда состоит из сегментов, сегмент из серверов, а сервер является в общем случае кластером процессов, работающих на разных хостах в инфраструктуре. Обновление кластера является операцией, которую можно выполнить несколькими способами, каждый из которых имеет свои особенности.
- Приложения взаимодействуют друг с другом, что в конечном счете означает, что нельзя обновлять функциональные сервера независимо друг от друга. Это взаимовлияние необходимо учитывать при запуске и остановке системы, при выполнении частичных обновлений и при необходимости полностью исключить простой системы.
- Если в процессе установки было выявлена критическая проблема и устранить ее не получилось, необходимо отменить операции изменений. Но для этого надо к этому быть заранее готовым.
- Файлы приложений могут быть существенного размера и при установке в распределенную среду в географически удаленную точку или при низком качестве сети, скорость передачи информации может быть настолько низкой, что простую процедуру обновления станет нельзя применять неприемлемого времени простой системы.
- Наконец, есть конфигурационные файлы, которые состоят из единого для всех содержания и параметров конфигурации. В некоторых случаях различия таковы, что в одной среде один файл, а в другой - другой и нет попытки поддерживать единое содержание.

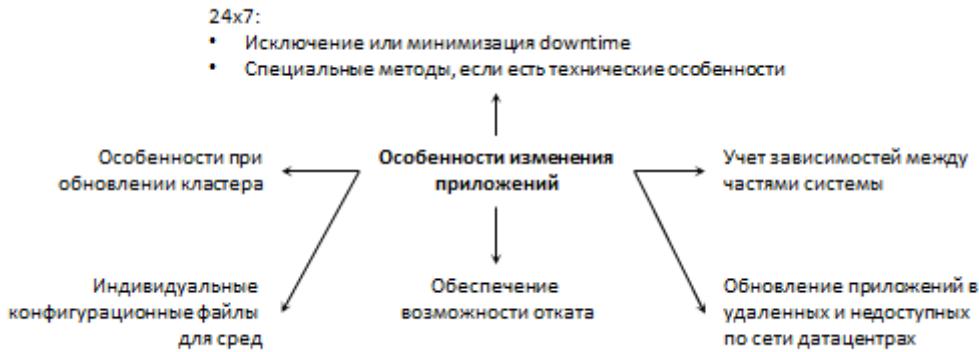


Рис. 201. Особенности обновления приложений

Особенности изменения баз данных, которые мы рассмотрим, следующие:

- Изменения баз данных как правило отличаются от изменений приложений тем, что, хотя там тоже есть код и базовое ПО, но в базах данных сред промышленной эксплуатации также присутствуют и операционные данные, которые нельзя отбросить. По этой причине изменение базы данных не сводится к "удалить старую версию файла и положить новую версию файла", а является более сложной операцией преобразования, которое требует определенного исходного состояния (для приложений мы можем удалить старую версию независимо от того, какая она) и приводит в новое состояние, которое зависит и от дистрибутива, и от исходного состояния ("stateful"). Операция обновления приложения - от исходного состояния для каждого отдельного приложения не зависит ("stateless"), хотя при использовании инкрементальных дистрибутивов применение к неправильной версии также породит некорректное состояние.
- Изменения могут относиться к метаданным (DDL-операции) и к данным (DML-операции)
- Изменения данных могут быть посредством обычных транзакционных операций, а могут относиться к операциям массовой загрузки данных
- Откат изменений баз данных также имеет подводные камни и даже не всегда возможен, что может завести ситуацию с пакетом изменений в тупик.
- В одной среде может быть несколько баз данных, а в каждой несколько схем. Т.е. конфигурация баз данных в продукте может быть достаточно сложной, отличаться между средами и нужна удобная и четкая логика какие изменения к чему применяются.
- Если что-тошло не так посредине обновления, то в силу невозможности простого отката, необходима четкая логика что делать в таком случае.

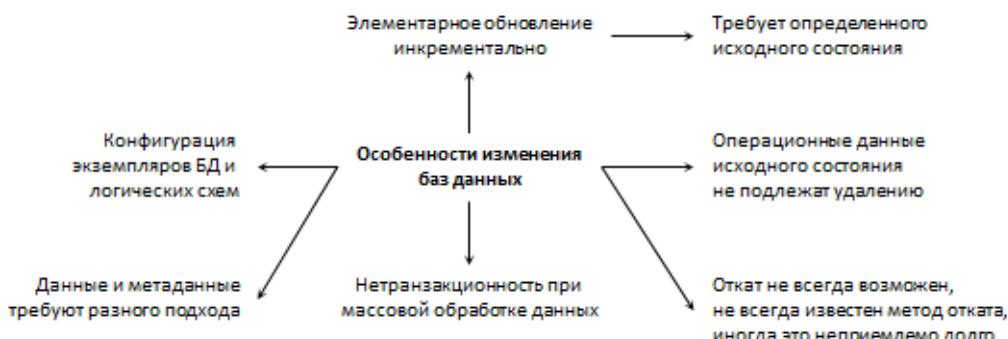


Рис. 202. Особенности обновления баз данных

То, что относится и к приложениям, и к базе данных:

- Обновления могут быть следствием релиза и следствие собственных операций процесса эксплуатации

- Обновления могут касаться закрытых файлов и данных, которые являются частью среды программного продукта и содержание которых должно оставаться недоступным для всех, кроме авторизованных администраторов. Это пароли, сертификаты безопасности и т.п.

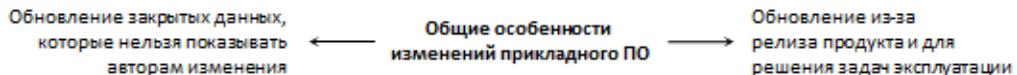


Рис. 203. Общие особенности обновления

Процедуры запуска и остановки процессов

Чтобы понимать причины проблем, связанных с установкой сложных многокомпонентных систем, необходимо как минимум понимать реальную последовательность запуска одного процесса:

- Запуск процесса операционной системы, самого приложения или сервера приложений, инициализация ядра сервера приложений
- Инициализация одного или нескольких приложений, в ходе которой происходит обращение к другим частям системы - например, соединение с базой данных и чтение настроек, обращение к связанному серверу для формирования начального состояния и т.п.
- Открытие своих операций для доступа - только в этот момент вызванная операция сможет быть принята и пойти по своему алгоритму выполнения



Рис. 204. Общий порядок запуска и остановки при промышленной эксплуатации

Процедура запуска процесса может занимать весьма существенное время - от десятков секунд для обычных серверов приложений типа Tomcat, WildFly, WebLogic до десятков минут для специализированных серверов типа ESB. Обращение же к серверу будет успешным только после завершения процедуры запуска процесса.

При остановке процесса в среде промышленной эксплуатации необходимо учесть следующее:

- Процессы, которые могут обращаться к останавливаемому серверу, желательно остановить, если они являются частью данного продукта и если их обновление также планируется. Средства маршрутизации в кластере должны обходить останавливающий узел. Это может делаться самим кластером базового ПО, но также это может быть и специальное переключение, выполненное администратором
- В момент сигнала к остановке сервер может обслуживать входящие обращения и в среде промышленной эксплуатации их обрыв может породить инциденты и нарушение операций. Желательно прекратить прием новых операций, но текущие операции выполнить до конца. Как правило, сервер приложений обеспечивает такой "щадящий" механизм остановки.

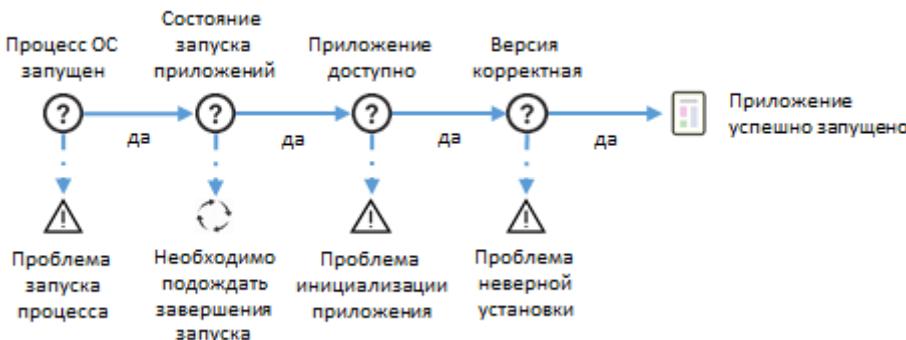


Рис. 205. Анализ состояния приложения

Таким образом при проверке статуса готовности приложения к обработке операций мы должны допускать одну из ситуаций:

- Процесс не запущен, обращаться к нему нет смысла
- Процесс инициализации приложения не завершен, о чем можно узнать, обратившись к серверу приложения, но к самому приложению обращаться нельзя
- Требуемого приложения нет или процесс инициализации завершен неуспешно.
- Приложение успешно инициализировано, но имеет неверную версию - требуемый интерфейс отсутствует
- Приложение успешно инициализировано и доступно для операций

Время недоступности сервиса

Публичные системы обычно являются системами режима 24x7, которые работают без прекращения сервиса. Тем не менее, для выполнения в такой системе может быть заложена возможность приостановки сервиса на какое-то время. На это время для пользователей вывешивается сообщение, так называемый дисклаймер ("disclaimer"), который говорит, по какой причине и на какой период прекращены операции. Если изменения плановые, то для такой приостановки стараются выделять специальное окно во время наименьшего использования системы.

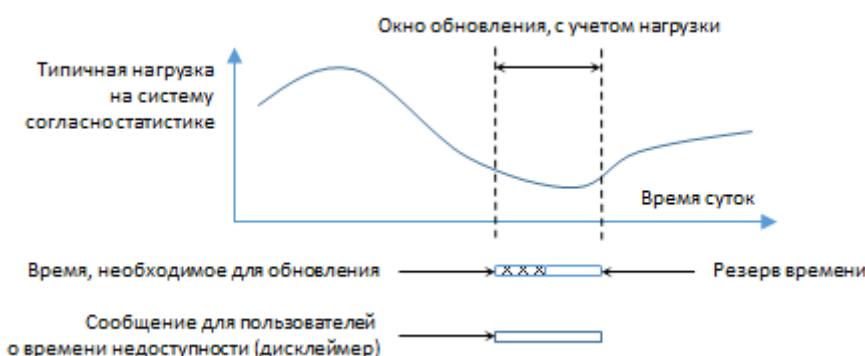


Рис. 206. Выбор времени и длительности обновления

Однако плановые операции могут выполняться и быстрее (что хорошо), и медленнее, так что есть риск не уложиться в обещанный пользователям период. Поэтому при выполнении изменений, а особенно при установке релизов программных продуктов, когда риски особенно велики, необходимо, во-первых, оставлять запас по времени между планируемым временем недоступности и выделенным окном, а самое главное, необходимо применять специальные технологии для снижения времени недоступности.

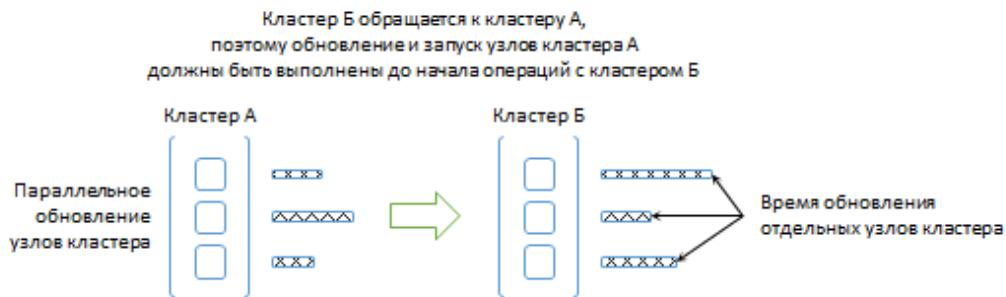


Рис. 207. Параллельное и последовательное выполнение работ в кластеризованных системах

Например, при обновлении распределенных систем количество обновляемых узлов и техническое время обновления одного узла могут быть такие, что линейное обновление одного узла за другим приведет к чрезмерному времени недоступности, превышающему размер окна обновления. Это означает необходимость распараллеливания операций, т.е. обновление узлов инициируется параллельно для нескольких узлов.

Тем не менее, запустить процедуру обновления сразу для всех узлов как правило нельзя из-за зависимости между процессами. Например, как указано выше, при старте приложения оно может обращаться к другому серверу и запуск будет неуспешным, если процедура запуска процесса другого сервера еще не завершена.

Если в вашей системе несколько сотен процессов и десятки зависимостей, то можно существенно сократить время обновления или перезапуска системы, применяя специально проработанный порядок операций. Одним из вариантов формализации порядка запуска, который позволяет указать, что можно, и что нельзя запускать параллельно, следующий:

- Всегда раздельно обновлять сегменты среды.
- По каждому сегменту объединить функциональные серверы в группы запуска, в которых все процессы - как разных серверов, так и разных процессов кластера одного сервера, запускаются одновременно.
- К следующей группе запуска переходить только тогда, когда все процессы предыдущей группы запустились успешно - т.е. их операции стали доступны для использования.

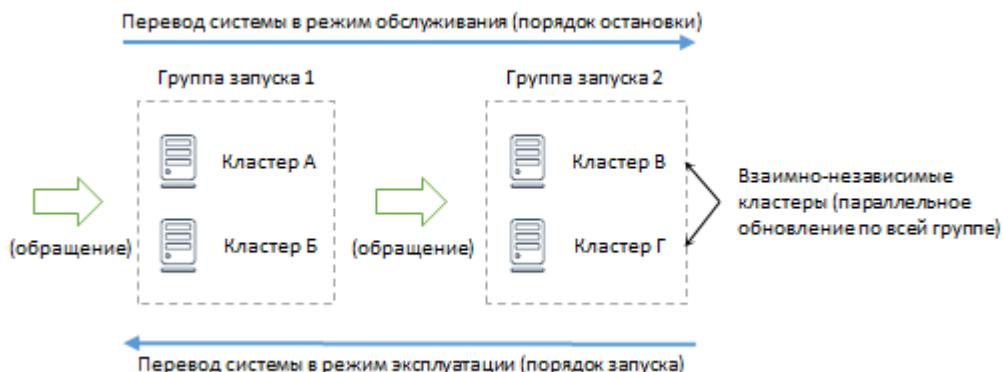


Рис. 208. Оптимизация времени обновления для сложных систем

Имея структуру запуска, мы можем применять ее в обратном порядке для остановки системы, которая с учетом ожидания завершения операций, также может быть длительной. Остановку систему также называют переводом системы в режим обслуживания.

Холодное и горячее обновление

Приложение в процессе реализует некоторую функцию и предоставляет определенный сервис. При холодном обновлении процесса мы его останавливаем, заменяем версию приложения и повторно запускаем. При этом возникает период времени, когда процесс не может выполнять операции.

При горячем обновлении мы как правило пользуемся механизмами сервера приложений, которые позволяют загружать новые приложения и переключаться на них только после завершения загрузки, таким образом, без остановки сервиса. Однако для сервера приложений как правило существуют библиотеки вне приложений и конфигурационные файлы, изменение которых в горячем режиме либо невозможно, либо будет конфликтовать с изменением приложений.

С другой стороны, понятие горячего и холодного обновления мы можем отнести как к приложению в процессе, так и к сервису. Зная про кластерную структуру функционального сервера, мы можем понять, что сервис предоставляется всем кластером и можно остановить один процесс кластера, переустановить версию приложения, запустить, затем сделать это для следующего процесса и так далее. В результате у нас получится горячее обновление сервиса при помощи холодного обновления процесса приложения.

Если мы обновляем кластер таким образом, то удобно разделить все узлы кластера на две группы, и сначала запретив обращения на одну из них, обновить их приложения, а затем проделать то же самое со второй группой узлов.

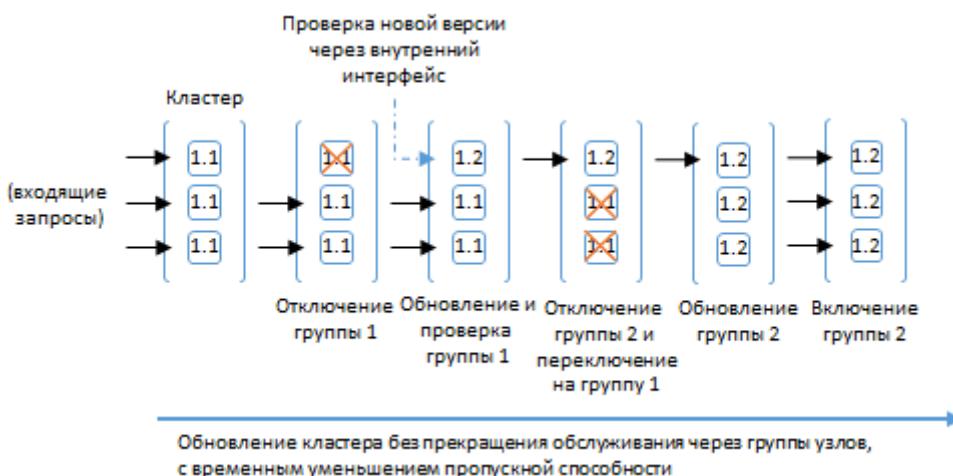


Рис. 209. Обновление кластера без поддержки сервера приложений

Еще одним вариантом горячего обновления обновления кластера процессов является доступная на некоторых серверах приложений функция горячего обновления через один, административный узел. Т.е. при обновлении мы выполняем операции только на одном узле, и он уже в свою очередь координирует обновление на других узлах.

Однако с учетом обновлений библиотек и конфигурационных файлов, а также из-за нестабильной работы серверов приложений при попытке поддержки одновременно нескольких версий одного приложения, такой вариант может не использоваться.

Также если процесс сервера из-за приложения оказался в некорректном состоянии, то горячее обновление также может оказаться недоступным. По этой причине сервера приложений как правило допускают разные варианты обновления и иногда надежнее самостоятельно управлять процессом обновления.



Рис. 210. Элементы холодного и горячего обновления в одном сервере приложений

Доведение обновления до конца

Инструкции по установке персональных продуктов часто содержат фразу, что если возникла проблема при установке, то нужно полностью удалить приложение и поставить его заново. Полное удаление и начало с чистого листа не подходит для средних и крупных серверных продуктов, части сред которых однократно создаются, причем разные части в разное время, а потом лишь меняются, в той или иной степени.

Установка пакета изменений также может столкнуться с проблемами. После появления проблемы мы можем либо игнорировать ее и продолжить установку, если программа установки учитывает такую логику, а проблема не ведет к непредсказуемому состоянию среды и не порождает критических ошибок с точки зрения выполнения своих функций.

В противном случае установка обновления прекращается и среда де-факто содержит часть изменений, причем в общем случае в некорректном состоянии. Причину остановки обновления обычно можно выяснить в процессе обновления, проанализировав ошибку, если применяемые средства автоматизации имеют систему обработки ошибок и ведения протокола.

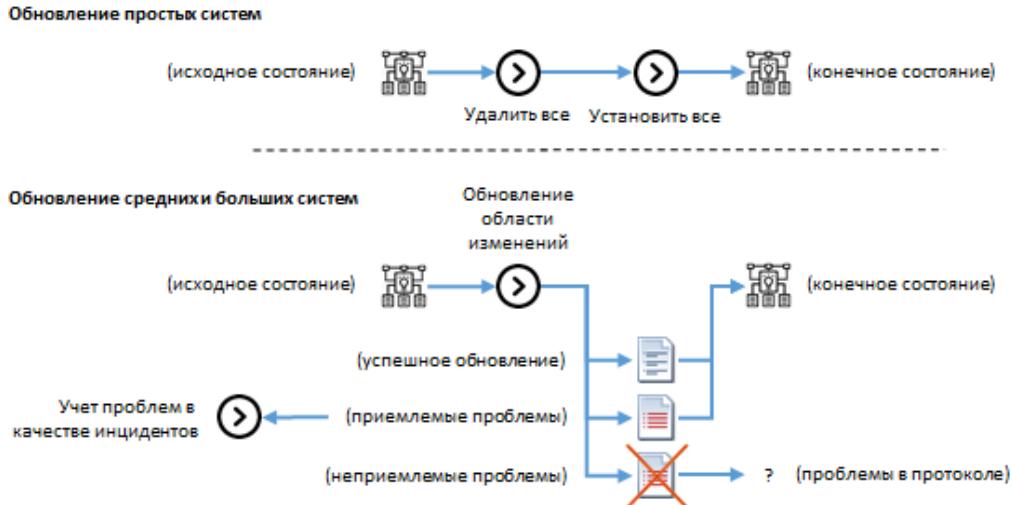


Рис. 211. Возможные варианты при обновлении систем

Предположим, сам по себе пакет изменений абсолютно корректен и не содержит ошибок, а причина ошибки при его установке носит внешний характер, понятна и может быть устранена в рамках выделенного окна для установки. В этом случае можно попробовать довести обновление до конца.

После устранения источника проблемы можно повторить операции, но для этого необходимо понять, что пакет изменений в общем случае состоит из:

- Удаления старых файлов (и каталогов) и добавления новых. Такая операция, если файлы учтены как элементы конфигурации, является декларативной и в любой возможной ситуации при установке четко известно, что необходимо и достаточно сделать для успешного завершения. Такая операция опирается только на декларируемые в учетной системе метаданные и не использует неявных знаний. Если файл, например, имеет вид log4j-1.2.16.jar, а удаление старой версии выполняется операцией rm log4j-*.jar, то такая операция будет идемпотентной, т.е. результат операции не будет зависеть от того, какая версия продукта установлена сейчас и был ли какой-либо сбой при предыдущем выполнении этой операции.
- Выполнение программ в составе пакета изменений, которые что-то изменяют в среде, следуя своей внутренней логике, например, изменяя операционные данные, переводя их на новый формат (например, изменяя структуру таблиц в базе данных).

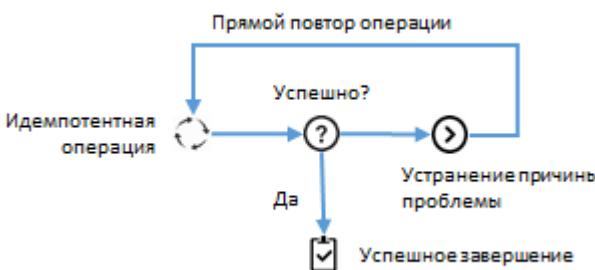


Рис. 212. Идемпотентная операция

Эти действия для одного процессса на одном хосте можно считать строго последовательной цепочкой, которая в конкретном звене была оборвана, т.е. данное звено было выполнено с ошибкой.

Чтобы довести обновление до конца, необходимо правильно выполнить данную операцию и продолжить выполнение, выполнив все оставшиеся операции.

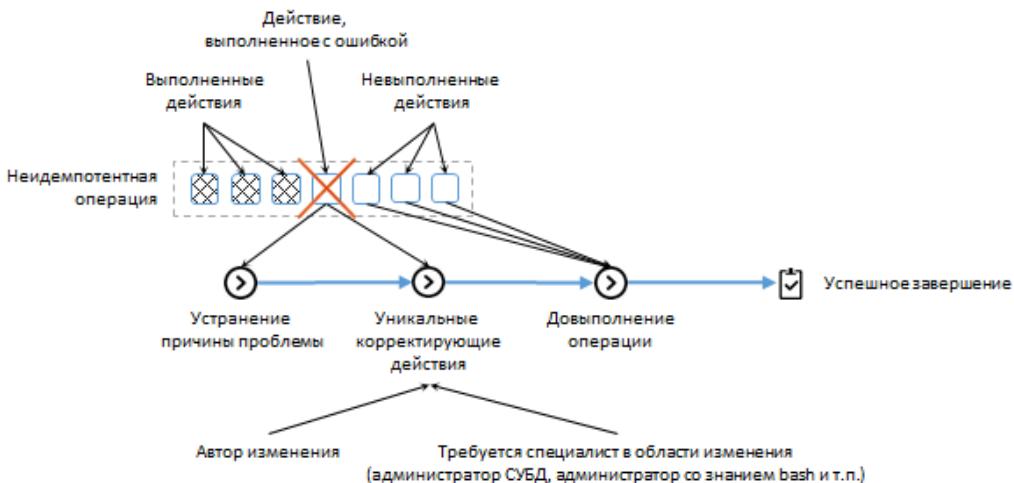


Рис. 213. Неидемпотентная операция

Точечное выполнение операции, где была ошибка, является сложным моментом, и для случая запускаемых программ может быть корректно выполнено только человеком, который понимает, как эта операция фактически выполняется. Если это был скрипт изменения базы данных, то нужен администратор СУБД, если это скрипт на языке bash, то нужен человек, который в нем разбирается.

Возможно, данная программа сама сможет разобраться с частично выполненными собою изменениями и при повторном применении успешно выполнит операцию, однако если данный факт не документирован, придется связаться с автором данного изменения и узнать, что именно стоит сделать.

Если система установки ведет учет выполненных операций, то исправив точечно одну операцию, остальные мы можем применить, просто заново запустив установку с командой применить все изменения, что не были применены.

Обеспечение возможности отката изменений

Из чего состоит откат изменений? Из воссоздания состояния до выполненных операций. Какие операции имеются в виду? Для ответа давайте сначала поймем причину отката.

Откат осуществляется либо из-за невозможности завершить операции пакета изменений, либо из-за того, что последующая проверка выявила, что пакет изменений содержит неприемлемые изменения.

Если невозможно оставить в систему в таком состоянии для режима эксплуатации, то выполняется полный или частичный откат изменений. В последнем случае откат приводит к состоянию, которое отсутствует с точки зрения релизной истории и носит временный характер.

Поскольку предметом тестирования при выпуске релиза был весь комплект изменений, то частично выполненные изменения соответствуют конфигурации продукта, которая не проходила тестирование. Это порождает риск, который тем не менее может быть принят и не привести к реальным проблемам. Решение такого рода можно принять только с участием разработчиков, которые в состоянии оценить внутренние зависимости и последствия.

Для внорелизного пакета изменений как правило нет предварительного тестирования изменений как комплекта, и каждое изменение либо является регулярным и выполняется как часть администрирования, не требующего предварительного тестирования, либо как отдельное изменение, которое тестируется отдельно от остальных изменений пакета.

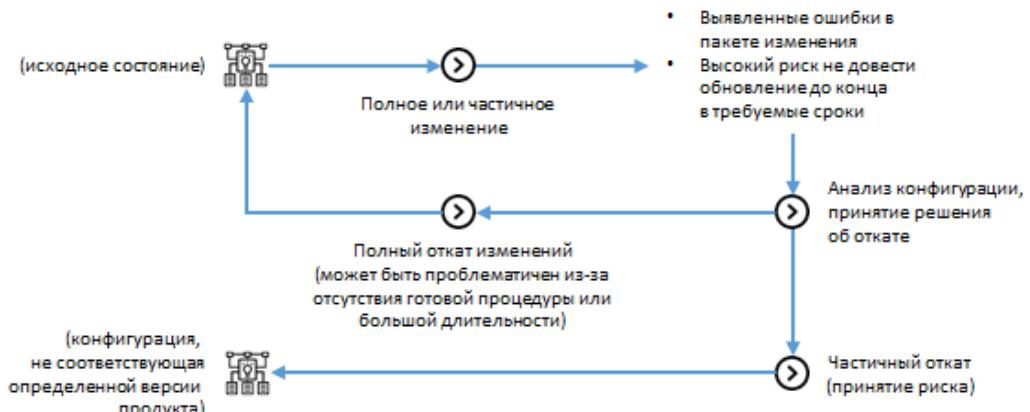


Рис. 214. Основные варианты отката изменений

После успешного выполнения пакета изменений, что предполагает проверку успешности и признание изменений завершенными, пакет изменений все-таки может быть отменен, если изменения дошли до пользователей и оказалось, что эксплуатация не обнаружила критических проблем, которые тем не менее были. Это время начальной эксплуатации, когда пакет изменений еще не закрыт.

Однако через некоторое время, за которое могут уже появиться и примениться другие пакеты изменений, когда в целом новое состояние продукта прошло проверку реальными пользователями, откат старого пакета изменений скорее всего принесет больше неопределенности и будет нежелателен. В этом случае следует решать проблему как инцидент той или иной степени критичности и выпустить новый пакет изменений, в том числе новый релиз.



Рис. 215. Начальный и основной период эксплуатации после установки изменений

Операции удаляют старые файлы, добавляют новые, изменяют операционные данные, выполняют учетные действия, выполняют программы в составе пакета изменений, которые что-то изменяют в среде, следуя своей внутренней логике, а также создают временные файлы. В идеале откату подлежит все, кроме временных файлов, которые чистятся как часть общего обслуживания системы.

Сложность состоит в том, что совокупность операций изменения может оборваться в любом месте по неизвестной изначально причине (недостаточно места, оборвется сессия, не хватит прав и др.) и нужно обеспечить откат из любого места этой цепочки до ее начала.

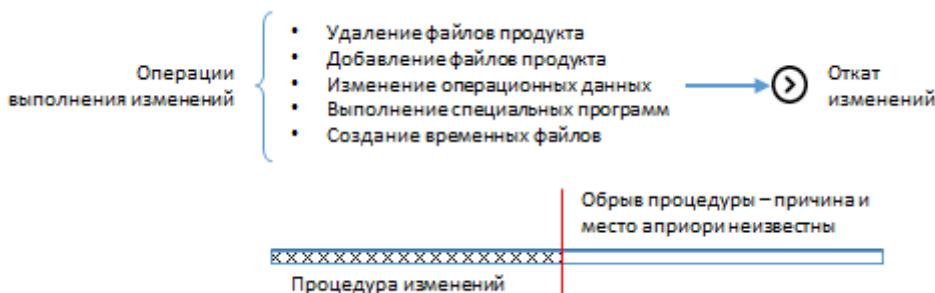


Рис. 216. Операции процедуры изменений и их откат

Самый простой и понятный способ отката, если у нас есть явное изменение файлов, список которых понятен и открыт, которые документированы с точки зрения администрирования системы. Такая информация позволяет говорить о декларативности, что обеспечивать прозрачность и возможность использования данных в разных алгоритмах. Для подготовки к откату достаточно выполнить следующие операции:

- Знать, какие новые файлы появились, чтобы их потом удалить при откате.
- Сохранить файлы, которые изменялись или удалялись. Здесь важно учесть, что названия файлов при изменении могут меняться, например, с log4j-1.2.16.jar на log4j-1.2.17.jar.

Учетные данные (о факте установке, версиях и т.п.) также декларативны и их откат не вызывает проблем. Достаточно привязывать учетные данные к конфигурационным состояниям, учитывая для каждого учетного факта соответствующий пакет изменений.



Рис. 217. Подготовка операции отката для простых файлов

Серьезную проблему порождают программы изменений в составе пакета изменений. Это черные ящики, которые выполняют изменения, которые неизвестны. Это недекларативные алгоритмы и формальный откат по декларативным данным невозможен. Для каждой программы изменений для отката нужна соответствующая программа отката изменений, которая также является черным ящиком и после выполнения которой неизвестно - результат достигнут или нет. Тестирование программ отката изменений с большой вероятностью не выполняется и при возникновении проблемы иногда предпочитают откатывать изменения самым простым способом - сохранять копию всего дерева установки и восстанавливать ее при проблемах.



Рис. 218. Неопределенность при использовании недекларативных процедур

Но самую большую проблему порождает то, что пакеты изменений приводят к изменению операционных данных - либо как программа преобразования, входящая в состав релиза, либо как изменение логики изменений данных так, что с каждой операцией пользователя появляются новые данные, которые не вписываются в состояние, которое было до релиза.

Если операции имеют коммерческую или юридическую значимость, то нельзя просто удалить эти данные, их приходится как-то учесть и объяснить пользователю, что произошло. Для публичных систем с большим количеством операций это может быть большим объемом работ, которые являются крупной нестандартной задачей.

Для пакетов, приводящих к изменению операционных данных, стоит соблюдать следующие правила:

- Крайне нежелательно оставлять состояние, которое может порождать некорректные данные, и лучше заблокировать выполнение операций, чем потом тратить силы и время на исправление ситуации.

- Если изменение является программой преобразования данных, то при наличии в составе пакета изменения, которое откатывает данную операцию, у администраторов появляется быстрый способ решить проблему, если она вдруг возникнет. К сожалению, это зачастую лишь благие намерения, поскольку тестирование процедуры отката производится только при очень зрелом процессе управления выпуском релиза и даже предоставленный код отката может не работать.



Рис. 219. Некорректные операции с операционными данными

Установка инкрементальных релизов

Если продукт большой, имеет много функциональных серверов с множеством компонентов, а релизы выпускаются часто, то выпуск полного продукта с полным дистрибутивом неудобен по следующим причинам:

- Каждый элемент, находящийся в дистрибутиве, может быть установлен, поэтому выпускающая группа, включая разработчиков, тестировщиков, системных инженеров и менеджеров обязана проводить операции по обеспечению качества, связанные с этими элементами - обеспечивать хотя бы минимальное тестирование, проверку статуса, управление кодом и т.д. Это лишние затраты, если, например, заранее известно, что целью релиза является обновление двух из трех десятков приложений на одном функциональном сервере и связанные изменения базы данных.
- При повторной сборке элемент дистрибутива получает новое состояние, несмотря на отсутствие каких-либо изменений. Например, по той причине, что jar - это архив класс-файлов java, а при компиляции время создания класс-файлов будет разным при разной сборке, конечный файл будет иметь разный размер и хеш-сумму просто из-за факта отдельной сборки. Но тогда поскольку файл другой, и он есть в дистрибутиве, придется его устанавливать, хотя в списке изменений будет прочерк, и нечего будет указать в качестве причины для остановки сервиса. Т.е. мы порождаем риск, не принося полезного результата.
- Изменения в коде могут иметь такой характер, что выпускать новую версию некоторых компонентов в эксплуатацию нельзя - она не будет работать. Причем это может произойти даже без изменения кода продукта, а вследствие, например, изменения элементов, от которых зависит продукт, и которые автоматически используются при сборке программного кода. Полное отчуждение внешних компонентов решит проблему, но тогда появится другая - когда на самом деле надо будет изменить, у вас останется старая версия.
- Также немаловажным фактором является объем дистрибутива - для полного релиза он максимальен и элементарное копирование по сети может привести к трате времени и лишним проблемам.



Рис. 220. Дополнительные затраты и риски из-за необязательных элементов дистрибутива

Все это приводит к мысли о необходимости инкрементальных релизов - когда изменения заведомо производятся в ограниченной части среды и в дистрибутиве находятся только элементы, которые требуется обновить. Однако при использовании инкрементальных релизов возникают следствия, о которых необходимо знать:

- Инкрементальные релизы имеют зависимости, они не идемпотентны и должны быть применены только к состоянию, соответствующему предыдущему релизу. Кумулятивные релизы, описанные ранее, смягчают это требование - будучи инкрементальными, кумулятивные релизы могут быть применены к среде, соответствующей целому множеству версий продукта.
- Строго полным будет только релиз, в котором есть все бинарные файлы, все конфигурационные файлы, и все изменения базы данных с начала релизной истории продукта. Однако это иногда практически невозможно и бессмысленно формировать базу данных из изменений за несколько лет. Фактически даже те релизы, в которых есть все бинарные и конфигурационные файлы среды, являются инкрементальными - из них можно восстановить всю среду кроме баз данных. И это весьма удобный вариант - поскольку сохранение и восстановление баз данных обычно делается в среде промышленной эксплуатации при помощи процедур создания и загрузки дампов.
- Если из среды пропадет какой-то файл, то необходимо будет его искать по все релизной истории и взять из последнего дистрибутива, где он присутствует.
- Файл, который работает в среде, будет непросто идентифицировать с точки зрения релиза - он мог быть обновлен не в последнем, а в предыдущем релизе. При воспроизведении инцидентов важно будет понимать точную конфигурацию среды промышленной эксплуатации и уметь находить несоответствие между файлами и релизной историей.



Рис. 221. Особенности инкрементального релиза

Если вам понадобится создать тестовую среду, подобную среде промышленной эксплуатации, или восстановить какую-либо среду после масштабного сбоя, то это можно сделать только в следующем порядке:

- Загрузить текущие дампы из баз данных среды промышленной эксплуатации
- Установить последний полный дистрибутив
- Установить последовательно все выпущенные инкрементальные релизы (или соответствующие им кумулятивные релизы).

Поскольку инкрементальных релизов может быть очень много, то процесс может быть слишком длительным и неудобным. По этой причине, как было сказано выше, стоит поддерживать актуальный полный дистрибутив, обновляя его после каждого завершенного инкрементального релиза. И разумеется, не забывать о регулярном бэкапе баз данных среды промышленной эксплуатации.

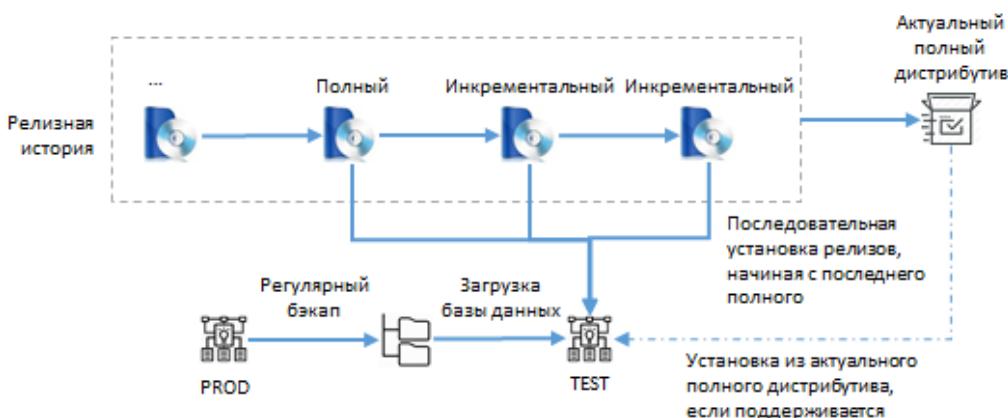


Рис. 222. Формирование тестовой среды из полных и инкрементальных релизов

Сама установка инкрементального релиза протекает следующим образом:

- Элементы дистрибутива копируются только на те серверы, где они используются. Специальным случаем является релиз исправления критичной ошибки ("hotfix"), в котором могут обновляться не все серверы, где есть этот элемент, а только тот один, где необходимо исправить критическую ошибку.

- Именно затрагиваемые серверы останавливаются и запускаются как описано в процедуре выше. Здесь важно отметить, что иногда придется и другие серверы перезапускать, если они зависят от данных, которые меняются при обновлении. В распространенном подходе к промышленным системам считается, что просто так зависимый сервер перезапускать не надо - он должен уметь обнаружить недоступность устанавливаемого сервера и восстановить соединение после завершения процедуры запуска устанавливаемого сервера.
- Предметом проверки после запуска являются затрагиваемые серверы и операции, которые демонстрируют, что другие серверы системы восстановили соединения с измененными серверами.

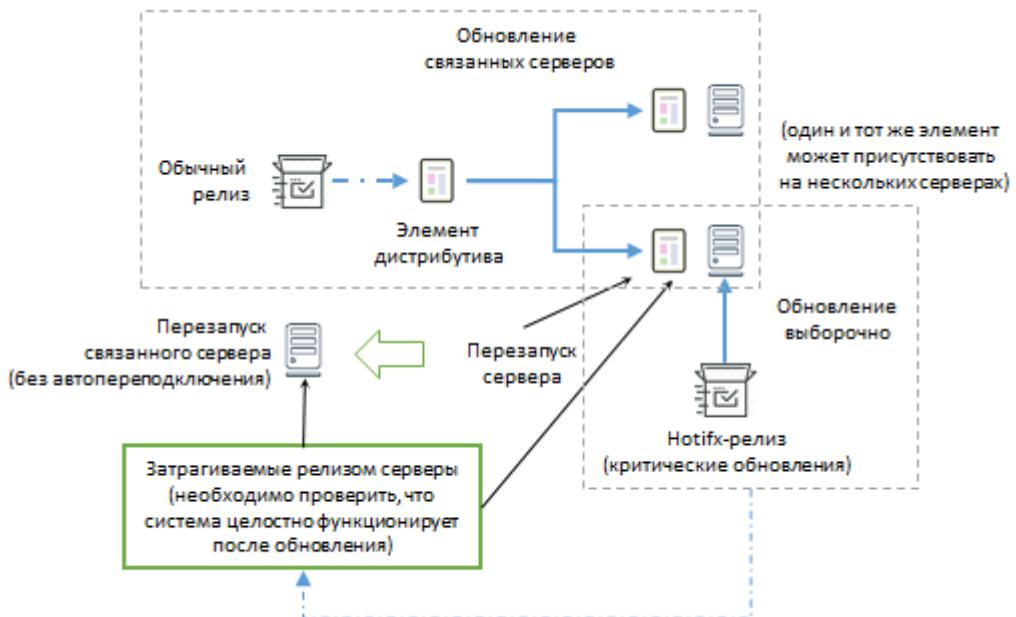


Рис. 223. Множественное влияние одного элемента дистрибутива

Изменение конфигурационных файлов в приложениях

Конфигурационные файлы являются текстовыми файлами для редактирования в произвольном редакторе и исходно выделены из приложения по следующим причинам:

- Чтобы можно было легко модифицировать логику приложения в вариантах, которые предусмотрены самим приложением, на уровне администрирования, без программистов.
- Также при помощи конфигурации программисты выводят параметры, которые не подлежат изменению администраторами, но которые будет удобно изменить самому программисту, без правки обычного кода.
- Используя многочисленные компоненты от третьих лиц из других продуктов, конфигурационные файлы служат для интеграции этих элементов в одну программу и для связи данного процесса с другими процессами системы
- Особым случаем конфигурационного файла является, например, сертификат, когда все содержимое файла является значением одного параметра конфигурации продукта.

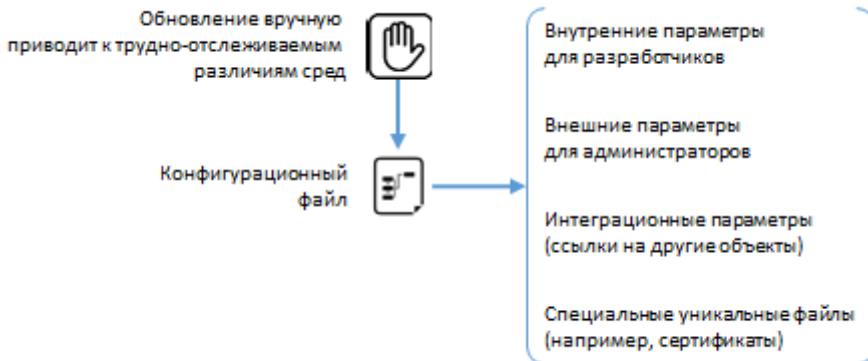


Рис. 224. Предмет изменения в конфигурационных файлах

Возможность обновлять конфигурационные файлы редактированием является в промышленных средах палкой о двух концах, поскольку хотя файл и легко изменить, зафиксировать факт изменения сложно, а изменение вручную приводит к тому, что изменено не то, не там, не везде где надо и т.д. Считается, что изменение вручную является крайне нежелательным и для этого используются специальные инструменты.

Конфигурационный файл может быть одного из следующих типов, имеющих разную процедуру изменения:

- Файл, который передается в дистрибутиве, не подразумевает никакого изменения в среде, хотя теоретически допускает, что это понадобится в одном из следующих релизов или для каких-нибудь специальных целей.
- Файл, который передается в дистрибутиве, содержит параметры конфигурации со значениями в виде переменных, которые требуется изменить или определить при развертывании конкретной среды продукта
- Файл, который описывается в руководстве администратора и может присутствовать в дистрибутиве в качестве примера, но конечное содержание может быть разным и разница не определяется только разными значениями параметров конфигурации. Такой файл может быть уникальным по структуре и содержанию в разных средах. В этом же числе файл, который имеет предопределенную структуру (например, сертификат) и связь с параметрами конфигурации, но файл имеет разное название в разных средах.

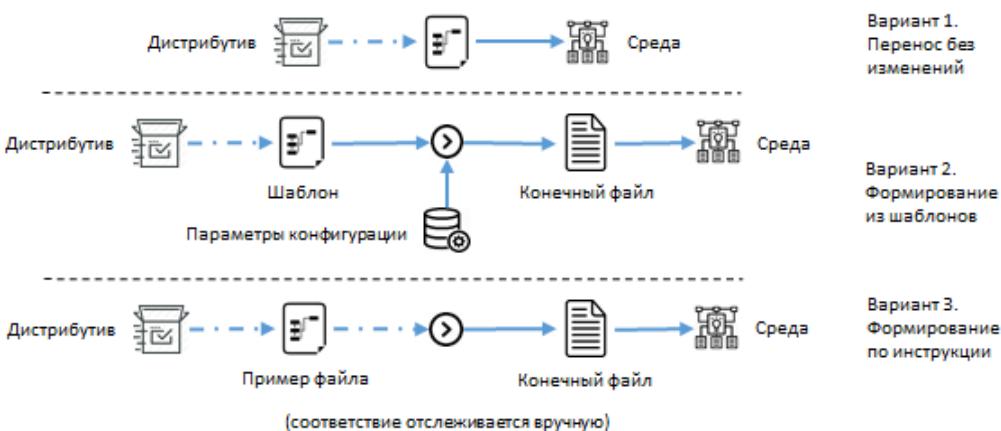


Рис. 225. Процедуры формирования конфигурационных файлов

Конфигурационные файлы могут появляться и удаляться в конкретном релизе. Изменение файла является следствием изменения значения параметра конфигурации, изменения остальной части файла или и того и другого. Изменение значения параметра конфигурации может быть требованием разработчиков программного продукта, связанным с логикой программы, или решением эксплуатации, связанным только с ее задачами.



Рис. 226. Авторы изменений в конфигурационных файлах

То, что файлы могут быть изменены вручную, может использоваться для решения критических инцидентов, но система установки должна позволять обнаруживать такие изменения. Порядок управления конфигурационными файлами в данном случае следующий:

- На основании критического инцидента вручную выполняется изменение, которое фиксируется в инциденте.
- При закрытии инцидента инициируется плановое изменение, которое повторяет выполненное действие, проводится через тестирование и выводится плановым способом ближайшим релизным или внорелизным пакетом изменений.



Рис. 227. Решение срочных инцидентов

Обновление закрытых данных в приложениях

Некоторые параметры конфигурации и даже целиком файлы могут относиться к закрытым данным. Например, пароль к базе данных или сертификат безопасности. Такие параметры и файлы не могут храниться в учетной системе, с теми же правами доступа.

Отметим, что все остальные файлы и данные желательно держать доступными для разработчиков и тестировщиков, поскольку это поможет лучше понимать конфигурацию продукта в среде промышленной эксплуатации и поможет избежать многих ошибок. Такие данные не могут находиться в дистрибутиве.

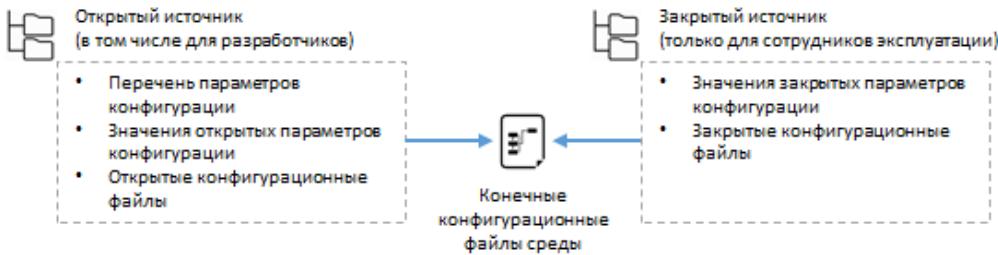


Рис. 228. Открытые и закрытые данные

В результате в общем случае конфигурационный файл берется как уникальный объект для данной среды, в том числе, закрытый, или из дистрибутива, путем подстановки в него значений параметров конфигурации для данной среды. Значения параметров конфигурации могут зависеть не только от среды, но и от конкретного узла, и от конкретного сервера, куда помещается данный файл. Это означает, что, например, для одного шаблонного файла в одной среде будет несколько вариантов заполнения.

Значения параметров конфигурации могут быть взяты из открытых данных учетной системы или из закрытых данных. Для снижения рисков из-за множества вариантов формирования конфигурационных файлов система установки должна иметь возможность увидеть конечное содержание файлов, каким оно будет после установки, до выполнения операции.

Обновление закрытого файла:

- Изменение содержимого закрытого файла выполняется только вручную по инструкции из плана установки пакета изменений

Обновление закрытых данных в открытом файле:

- Шаблон файла обновляется из дистрибутива релиза
- Закрытые данные - значения параметров конфигурации меняются в месте своего хранения согласно инструкции из плана установки пакета изменений (самих значений там быть не должно, план является общедоступным документом)
- После подстановки значений файл размещается в рабочей области среды

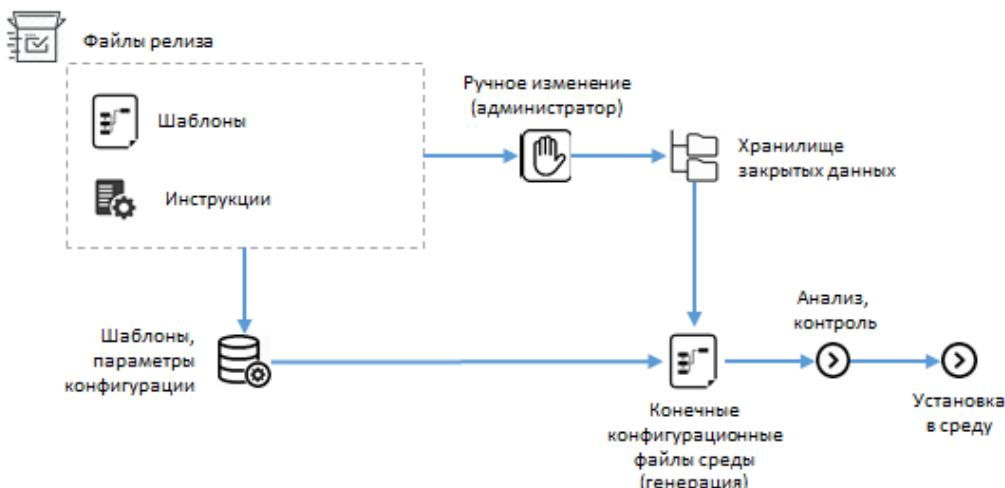


Рис. 229. Обработка конфигурационных файлов в релизе

Операции изменения баз данных

Изменение базы данных изначально относится к одному из следующих типов:

- Изменение хранимых процедур - аналогично изменению приложений, но это часть базы данных. Самое простое изменение, поскольку является идемпотентной операцией - можно без опасений повторно применять. Относится к DDL-операциям. Все остальные изменения, описанные ниже, как правило, идемпотентными не являются.
- Изменение структуры данных, типов данных и т.п. - как правило в структурах уже есть данные и такое изменение включает и связанную модификацию данных. Изменение структуры данных является DDL-операцией и в общем случае нетранзакционно. Накопленные данные могут быть такого объема, что их обновление в одной транзакции также будет невозможно. Отсутствие транзакций приводит к тому, что требуется анализ на уровне программного кода, чтобы завершить или откатить оборванную операцию. Такие операции называют миграцией данных. Несмотря на наличие в любой СУБД такого понятия как точка отката, в общем случае такая операция не применима в системах 24x7, где пользовательские операции в базе данных продолжают выполняться в процессе обновления и откат базы данных к точке сохранения приведет к неприемлемой потере данных. Относится к релизным изменениям.



Рис. 230. Сложности отката изменений в базе данных

- Изменение данных продукта - данных, которые входят в состав версии продукта (например, встроенный справочник). Относится к релизным пакетам изменений.
- Изменение операционных данных - аналогично миграции, но без изменения структуры данных. Может относиться как к релизным, так и внорелизным изменениям. Служит, например, для исправления заполнения существующих данных для соответствия новой логике.
- Изменение параметров конфигурации - параметры конфигурации могут храниться не только в конфигурационных файлах, но и в таблицах базы данных. Это позволяет вносить изменения в них в оперативном режиме при помощи интерфейса продукта. Тем не менее, изменение может идти и в виде скрипта изменения базы данных, в том числе и в составе дистрибутива от разработчиков. Может относиться как к релизным, так и внорелизным изменениям.

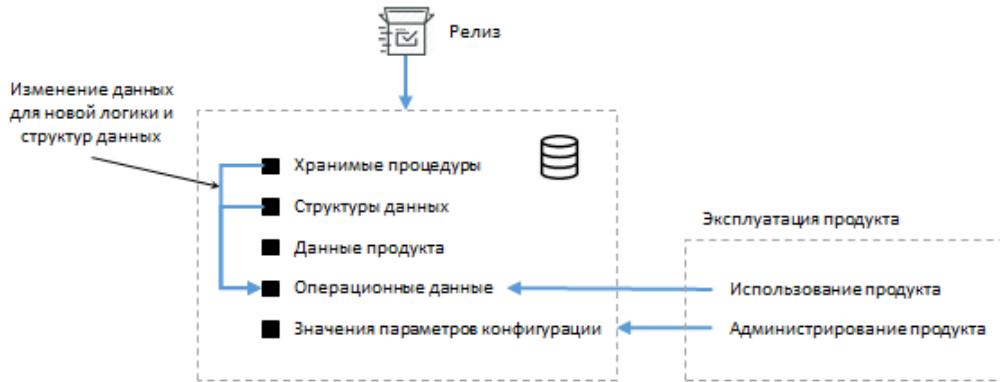


Рис. 231. Изменение базы данных со стороны разработки и эксплуатации

- Массовая загрузка данных - с использованием специального нетранзакционного, но очень быстрого механизма наполнения базы данных из внешнего файла. Производится как релизное или внерелизное изменение.
- Импорт дампа - начальная инициализация базы данных в среде. Может быть вариантом миграции данных, например, из одного вида СУБД в другой. Также практичный способ формирования баз данных в тестовых средах, способ первоначального развертывания в среде промышленной эксплуатации.



Рис. 232. Массовые операции изменений базы данных

Каждая СУБД предоставляет язык, на котором эти изменения описываются (например, PL-SQL для Oracle DDL- и DML-изменений). Изменение - это файл или группа файлов (например, ctl-файл и файлы данных). Для физической фиксации изменений, все эти файлы должны быть помещены в дистрибутив. Но поскольку это изменения, а не текущее состояние, то они имеют несколько другой характер, чем файлы в репозитории обычного программного кода. Эти изменения должны быть привязаны к релизу.



Рис. 233. Разные методы формирования файлов релиза для изменения БД и приложений

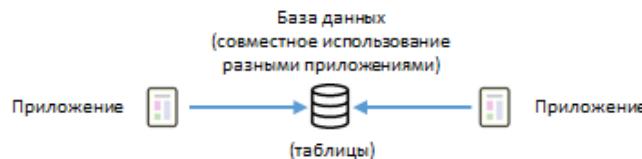
Файлы дампов и файлы данных для массовой загрузки могут быть настолько большими, что будет неверно их хранить в системе контроля версий вместе с кодом и файлы могут помещать вручную непосредственно в дистрибутив.

Изменение схем и экземпляров баз данных

Простая система состоит из одной базы данных, в которой есть какое-то количество таблиц. Исторически база данных была местом, где разные приложения обменивались данными. Однако в более сложных системах таблиц становится так много, что их разбивают по схемам, чтобы упорядочить работу с базой данных, и каждое приложение работает со своим подмножеством схем.

Более того, если среда распределена по нескольким географически удаленными сегментам, то как правило в каждом сегменте появляется одна или несколько локальных экземпляров баз данных со своим набором схем, которые могут служить и как реплика данных из другой базы данных и как место для своих собственных данных, иметь одинаковую или уникальную структуру данных.

Система с централизованной базой данных



Распределенная система с несколькими базами данных и схемами

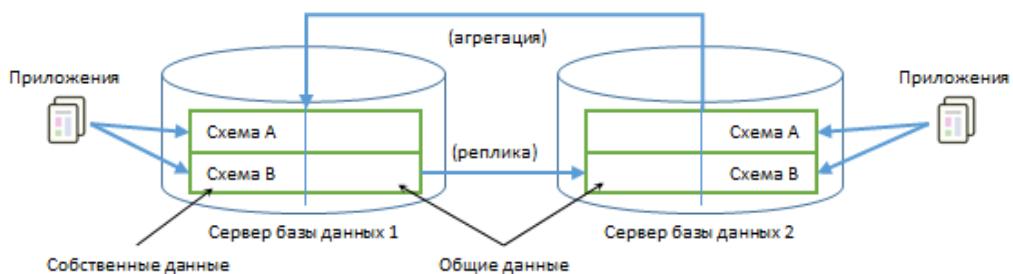


Рис. 234. Конфигурация баз данных

Схемы с точки зрения приложения важны с логической точки зрения - независимо от своего расположения в схеме с каким-то названием внутри какой-то базы данных, для приложения важно, чтобы схема содержала определенный список таблиц с определенной структурой, определенные хранимые процедуры и т.п. Это означает, что внутри одной физической схемы могут быть размещены таблицы и процедуры нескольких логических схем.

Конкретное решение о базах данных и физических схемах является частью конфигурации конкретной среды программного продукта и в дистрибутиве изменение должно ссылаться на логическую схему.

Если одна и та же логическая схема присутствует в нескольких базах данных, то либо есть указание, для какой базы данных это изменение было создано (например, наполнение уникальными данными) или это изменение применяется к нескольким базам данных (например, синхронно изменения структуру определенной таблицы).



Рис. 235. Физические и логические схемы баз данных

Некоторые изменения затрагивают несколько схем, используя SELECT, связывающий таблицы из разных схем. Для таких изменений можно создать специальную административную схему-пользователя с правами ко всем прикладным схемам базы данных. Это позволит без использования суперпользователя базы данных менять данные в прикладных схемах, а пользователей обычных схем ограничить полномочиями только своих собственных схем.

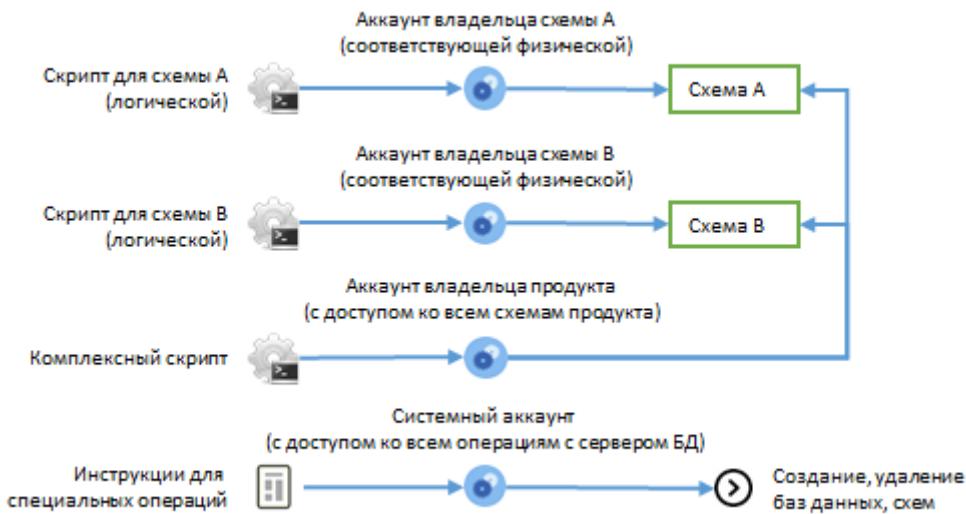


Рис. 236. Доступ к схемам баз данных при установке релиза

Факт применения изменения для задач контроля состояния и управления изменениями удобно хранить в самой базе данных - в отдельных служебных таблицах. Что позволяет, сняв с базы данных дамп, и загрузив ее в другую базу данных, понимать, какой версии продукта соответствует данная база данных.

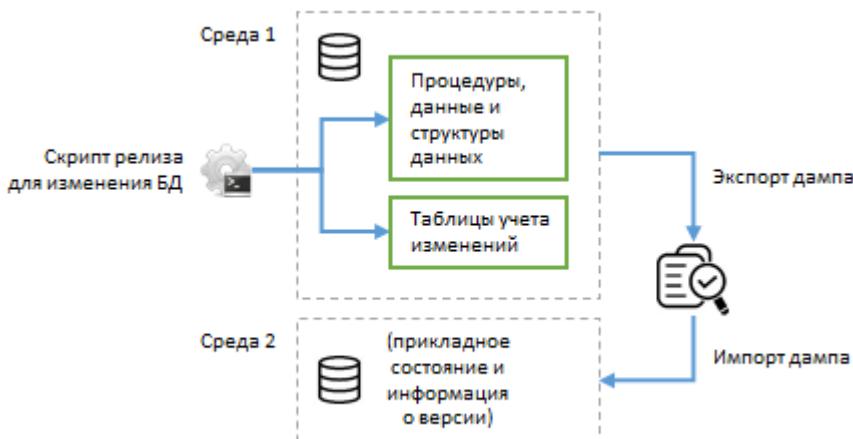


Рис. 237. Перенос баз данных вместе с информацией об установлененных релизах

Создание баз данных и их схем, их удаление и изменение свойств - редкие системные операции, и подобны инфраструктурным изменениям в составе релизного пакета изменений - созданию аккаунтов в операционной системе для развертывания приложений.

Скрипты отката изменений баз данных

Откат изменений баз данных, как уже было сказано - одна из наименее однозначных и рискованных операций в силу невозможности в общем случае создать алгоритм, обратный исходному, с учетом операционных данных. Тем не менее, в простых случаях такие скрипты можно относительно легко сделать и их наличие сделает ненужным присутствие программиста в момент установки релиза, которая может происходить в выходной день, ночь, во время соответствующего выделенного окна.

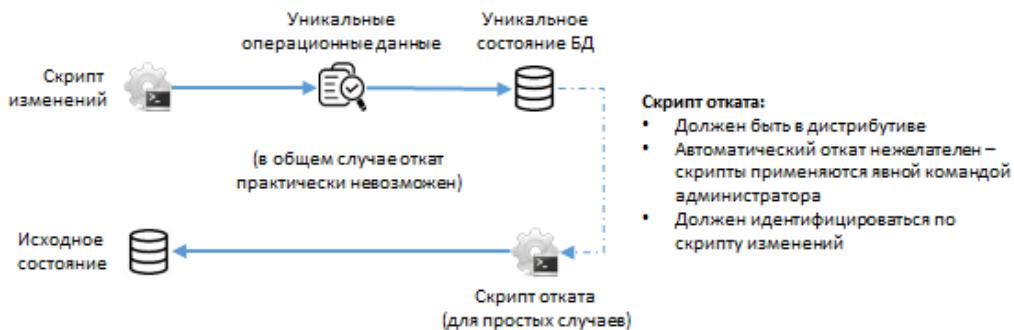


Рис. 238. Использование скриптов отката

Скрипты отката должны быть файлами, которые не применяются в автоматическом режиме, но которые легко найти в составе дистрибутива. Скрипт отката следует писать отдельно для каждого файла с самим изменением и называть так, чтобы можно было легко идентифицировать файл отката по файлу изменения.

Изменение параметров конфигурации в базах данных

Если конфигурационные параметры хранятся в базе данных, то их также можно менять при помощи скриптов изменений. Эти изменения будут уникальными скриптами изменений, которые относятся не к продукту, а к конкретной среде и к ее базе данных. Для остальных же скриптов, также как и для конфигурационных файлов, следует применять принцип шаблонизации - скрипт в составе дистрибутива должен быть одинаковым, с переменными, которые заполняются тем или иным способом.

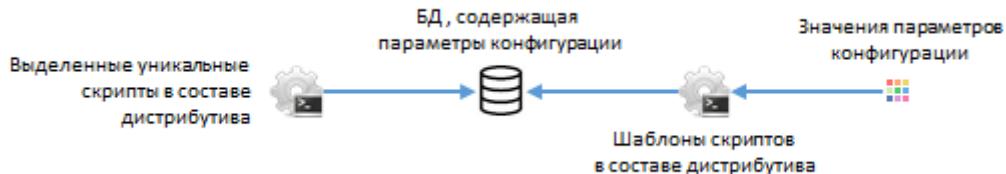


Рис. 239. Варианты изменения параметров конфигурации в БД

Мониторинг

ПРИЧИНА И ПРЕДМЕТ МОНИТОРИНГА

ВРЕМЯ РАБОТЫ ПРИКЛАДНОГО ПРОДУКТА

Активный мониторинг услуг

Ресурсный мониторинг

Мониторинг прикладных процессов

Мониторинг операций

Обработка событий мониторинга

Обновление системы мониторинга

Мониторинг и данные управления конфигурацией

Зачем нужен мониторинг? Что он измеряет и с какой целью, как выстраивается мониторинг, как релизы программных продуктов влияют на мониторинг?

Причина и предмет мониторинга

Если для сред разработки мониторинг, если он вообще есть, является просто действием или вспомогательной функцией, то для сред промышленной эксплуатации мониторинг воспринимают уже как отдельную независимую систему, которая является основным предметом заботы и разработки для сотрудников эксплуатации. У этого несколько причин:

- Несмотря на все разнообразие программных продуктов и технологий, проблемы, которые с ними связаны, одни и те же, и сотруднику, которому приходится следить за серверами баз данных двух разных систем, будет избыточным и непрофессиональным разрабатывать две разных системы отслеживания ситуации с этими серверами, смотреть в два экрана, создавать две линии реагирования и так далее. Профессиональный подход требует сокращения затрат там, где есть что-то похожее. Поэтому в службе эксплуатации, которая занимается сразу несколькими продуктами, неизбежно приходят к единой системе мониторинга.
- Многие проблемы прикладных продуктов не существовали бы, если бы разработчики о них знали. Поэтому мониторинг, созданный разработчиками и встроенный в сам продукт, будет страдать от тех же проблем, что и продукт. С точки зрения качества лучше, если мониторингом занимаются независимые от разработчиков специалисты в виде отдельной независимой системы.

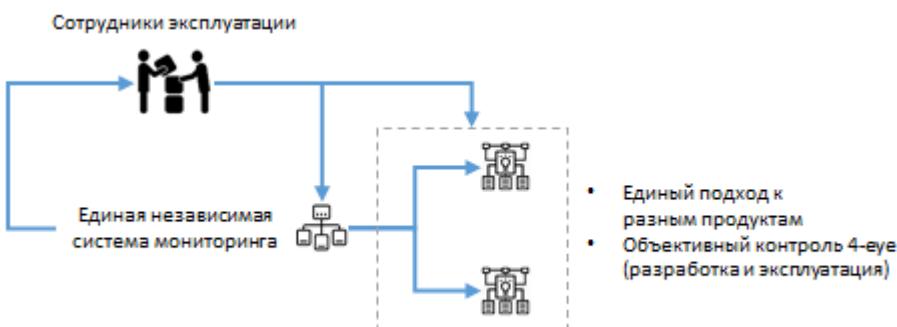


Рис. 240. Единая независимая система мониторинга

Мониторинг как профессиональный сервис ставится в случаях, когда эксплуатируемые системы важны для заказчика как предоставляющие определенные услуги пользователям. Мониторинг для службы эксплуатации преследует две цели:

- Обеспечение требуемого уровня сервиса
- Подтверждение оказания услуг



Рис. 241. Цели использования системы мониторинга

Уровень сервиса обозначает все те ожидания в отношении предоставления услуг, которые должны быть выполнены. Причины, по которым эти ожидания существуют, могут быть разными - требования могут поступить от заказчика или являться внутренними административными требованиями. Так или иначе, все эти требования должны быть собраны в утвержденный регламентирующий документ об уровне оказания услуг (SLA - Service Level Agreement) и который является единственным источником информации о том, что и как надо контролировать и отслеживать в системе мониторинга.

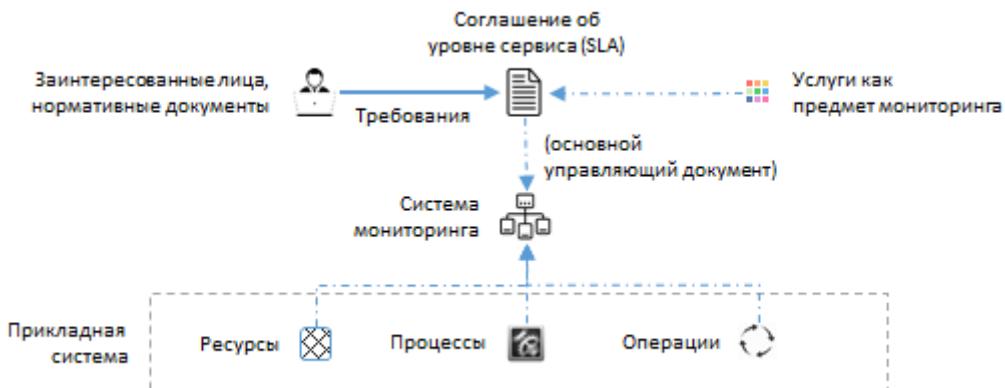


Рис. 242. Обоснование системы мониторинга

Предметом мониторинга являются:

- Предоставляемые услуги согласно SLA.
- Ресурсы, используемые для оказания услуг - электростанция, датацентр, канал передачи данных, система виртуализации, используемая оперативная память и т.п.
- Прикладные процессы уровня операционной системы, которые составляют прикладную систему - которые являются типичной точкой отказа в процессе оказания услуг или при обновлении.
- Отдельные операции, выполняемые в прикладной системе - в случае, если прикладная программа интегрирована с системой мониторинга так, что при сбое операции автоматически передается сигнал в систему мониторинга.

Задачами, типовыми операциями мониторинга являются:

- Контроль доступности услуг и ресурсов
- Учет объема предоставления услуг
- Заведение инцидентов и запросов на обслуживание
- Аналитическая отчетность по данным мониторинга
- Контроль успешности обновления прикладных программных продуктов

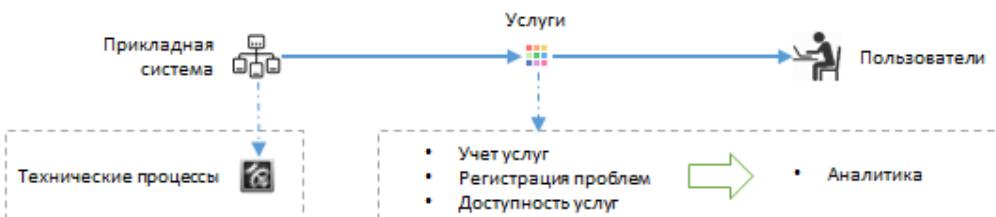


Рис. 243. Задачи системы мониторинга

Время работы прикладного продукта

Разные промышленные серверные продукты обладают разным суточным ритмом, что влияет на принципы их мониторинга и обновления:

- Системы 24x7, которые работают в непрерывном режиме. Тем не менее, такие системы могут выводить из режима эксплуатации в сервисный режим для выполнения обновления, если оно невозможно без прекращения операций. Также режим операций может быть разный - например, пользователям может предоставляться сервис в рабочее время с 9:00 до 21:00, но в

ночное время могут запускаться задачи пакетной обработки - расчетов, генерации отчетов и т.п. Такой график необходимо учесть в системе мониторинга и процедуре обновления. Еще один вариант - системы, предоставляющие сервис круглосуточно в рабочие дни, и не предоставляющие в выходные. Поскольку система в выходные дни запущена, она теоретически может предоставлять и услуги. Однако для пользователей услуги закрыты и могут также действовать разные ограничения, что означает, что система мониторинга на этот период должна быть отключена в части активного мониторинга услуг.

- Системы с периодическим использованием. Например, запущенные в рабочее время и останавливаются на ночь. Мониторинг как следствие на этот период также должен быть отключен полностью.
- Продукты для выполнения отдельных задач. Такие продукты запускаются, выполняют работу по задаче от пользователя, пока не сделают, и останавливаются. Это специализированные продукты, не относящиеся к области массового обслуживания. Их пользователи единичные, обновление производится между операциями, а мониторинг услуг в своем обычном виде не выполняется.

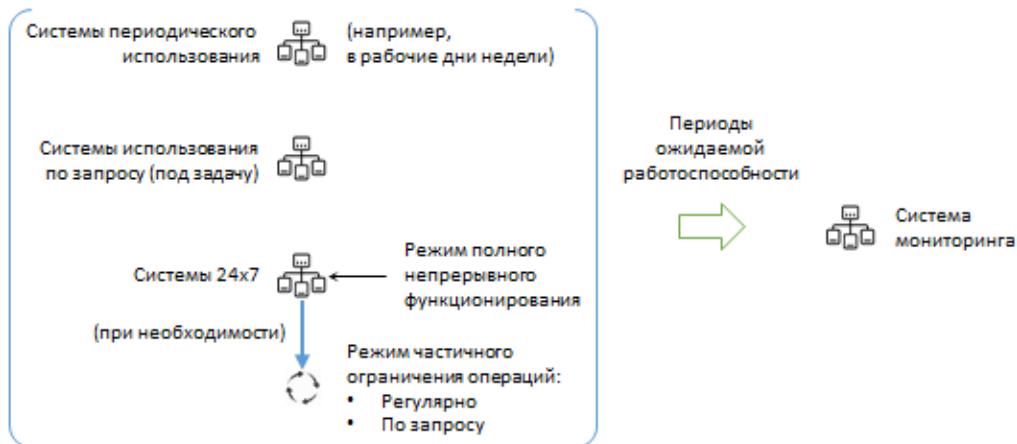


Рис. 244. Периоды ожидаемой работоспособности

Активный мониторинг услуг

Активный мониторинг услуг выполняется для контроля их доступности в то время, когда они должны предоставляться. Предметом мониторинга являются не элементы технических средств, а те услуги, ради которых существует прикладная система. Это означает, что мы должны выполнить сценарий предоставления услуги (заполнения формы, выполнения транзакции и т.п.). В идеальном варианте нужно от лица пользователя с его компьютера выполнить операцию, чтобы убедиться, что операция доступна для реального выполнения. Однако это не всегда возможно и используется компромиссный подход:

- Компьютер может быть недоступен для размещения элементов системы мониторинга. В таком случае, мы размещаем элементы системы мониторинга в месте аналогичном компьютеру пользователя - в той же сети, точно так же внешнем пространстве по отношению к обслуживающему пользователя серверу и т.п.
- Выполнение тестовой операции от лица пользователя как правило считается существенным нарушением прав доступа и не применяется. Для этих целей в системе авторизации создается псевдо-пользователь - служебный аккаунт, используемый именно для целей мониторинга.
- Рассматривая цепочку приема и прохождения операции пользователя в системе, мы можем направлять нашу тестовую операцию как на внешний интерфейс, так и на внутренний. Последнее используется, если внешний интерфейс имеет средства специальной защиты и вызов производится изнутри сети.
- Не всегда можно выполнить операцию, поскольку операция может иметь юридическую и финансовую значимость, и в тестовых целях недопустима. Как правило это верно для систем,

подключенных к системам в других организациях. По этой причине сценарий может быть намеренно урезан (например, создаем транзакцию, но не подтверждает ее).

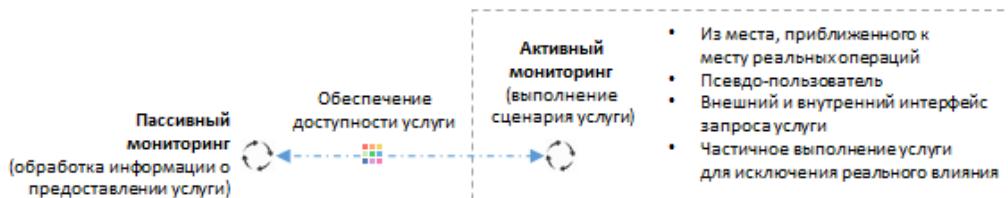


Рис. 245. Активный и пассивный мониторинг

Ресурсный мониторинг

Ресурсный мониторинг обеспечивает контроль ресурсов, используемых для предоставления услуг.

Ресурсы могут быть выстроены в иерархию по зонтичному принципу. Например, канал передачи данных, через который проходит поток запросов по услугам, требует отдельного мониторинга и находится в "ручке зонтика". Его отказ приведет к отказу предоставления всех услуг, которые как бы часть "купола зонтика". По этой причине мониторинг услуг при недоступности канала связи теряет свой смысл. Это создает зависимости между элементами мониторинга.

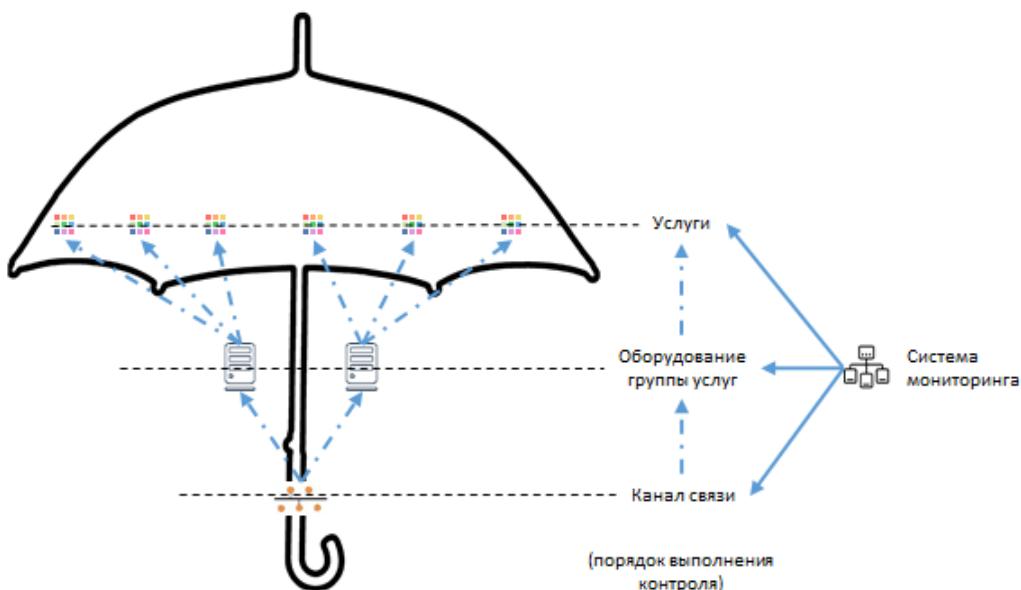


Рис. 246. Зонтичный принцип мониторинга услуг

В целом мониторинг ресурсов делится на:

- Активный мониторинг критичных ресурсов. Каждая проблема, выявленного таким мониторингом как правило требует срочного решения.
- Проактивный мониторинг утилизации ресурсов. В этом случае предметом мониторинга являются типовые параметры, такие как процент утилизации дискового массива, оперативной памяти и т.п.. Совмещение данных в аналитическом отчете за некоторый временной период даст понимание о тренде утилизации ресурсов и даст возможность при оптимальном резерве оборудования вовремя осуществлять дополнительные поставки оборудования.
- Проактивный мониторинг по параметрам производителя. При этом отслеживанию подвергается множество параметров, каждый из которых не является критичным, однако в

сумме они могут достаточно точно предсказать появление критической ошибки, например, ошибки чтения-записи отдельных секторов могут предсказать время выхода дискового массива из строя.



Рис. 247. Операции ресурсного мониторинга

Мониторинг прикладных процессов

Прикладные процессы определяют содержание прикладных систем и их успешное функционирование определяет возможность обеспечить предоставление услуг. Недоступность услуги указывает на невозможность целевого использования прикладной системы, в то время как недоступность процесса поможет определить место сбоя в системе. Прикладные процессы работают в среде, содержащей элементы стандартизированной инфраструктуры с типовыми операционными системами и базовым ПО. Основная причина сбоев на уровне прикладных процессов - ошибки в приложениях, которые являются частью прикладных продуктов и которые проявляются в процессе использования или сразу после установки релиза программного продукта.



Рис. 248. Причина и следствие проблемы

Ресурсы, необходимые для предоставления услуг, выделяются и распределяются исходя из количества, типов и структуры прикладных процессов. Поэтому данная структура служит средством контроля архитектуры программного продукта и обоснованности выделенных ресурсов. Как было описано ранее в конфигурации сред, прикладные процессы сгруппированы в кластеры, каждый из которых соответствует функциональному серверу или определенной функции в архитектуре программного продукта.

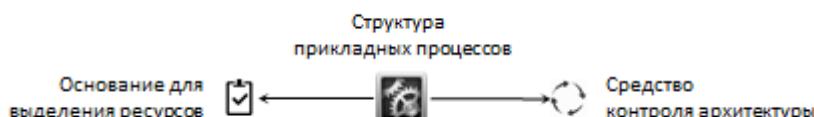


Рис. 249. Цели отслеживания структуры прикладных процессов

Выполнение мониторинга прикладных процессов дает возможность:

- Контроля корректности выделения ресурсов в привязке к конкретному продукту и системе

- Быстрого поиска неисправностей в системе
- Первоначальной проверки успешности процедуры установки релиза

Поскольку прикладные процессы существуют в строгой иерархии прикладных систем, становится возможным агрегирование событий мониторинга до уровня продуктов и систем.

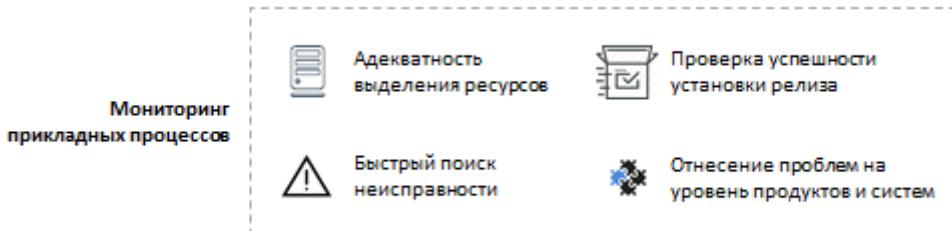


Рис. 250. Задачи мониторинга прикладных процессов

Мониторинг операций

В программировании есть такая типичная методика информирования о внутренних ошибках как исключения. Обработка исключений является частью большинства программ и состоит в том, что в зависимости от контекста, выполняемой операции и вида исключения порождать информацию об ошибке выполнения и присваивать ей определенный уровень критичности. Данная ошибка в момент появления может быть передана в систему мониторинга как сигнал, который далее, после обработки, может превратиться в зарегистрированный инцидент.

Это совершенно другая событийно-ориентированная (event-driven) концепция по сравнению с другими видами мониторинга, основанными на пассивном опросе процессов и интерфейсов продукта (polling). Преимущество событийной системы в том, что дополнительной нагрузки на продукт от системы мониторинга нет, а о проблеме можно будет узнать без задержки.

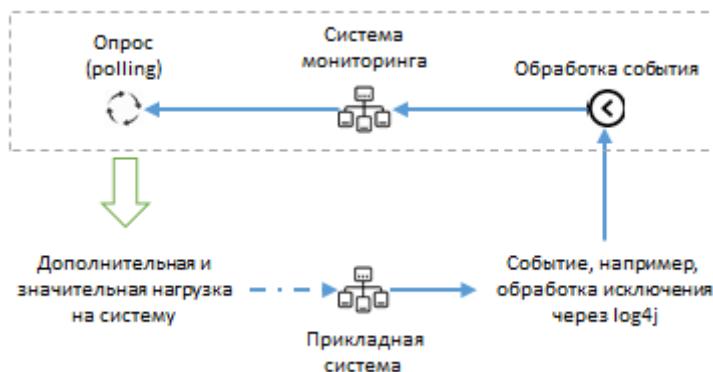


Рис. 251. Основные варианты мониторинга операций

Однако место возникновения и форма представления ошибки носят слишком технический характер и зачастую сложно сделать вывод о том, в чем собственно проявляется проблема для пользователя, есть ли она вообще и как это связано с SLA. Практическое использование такой системы информирования о проблемах имеет смысл только при наличии атрибутов в сообщении об ошибке, которые позволяют привязать данное сообщение к критериям SLA, что требует участия разработчиков прикладного продукта в настройке системы мониторинга.



Рис. 252. Настройка системы мониторинга операций

Обработка событий мониторинга

В системах массового обслуживания количество операций может исчисляться сотнями и тысячами в секунду. Количество контролируемых элементов и отслеживаемых состояний системы в реальных системах не позволяет реагировать на каждое отдельное событие вручную. Сигнал, пришедший от механизмов событийного или опрашивающего мониторинга, называется системным событием и может требовать одного из следующих вариантов обработки:

- Это свидетельство реальной новой проблемы в прикладной системе, которой на основании SLA присваивается уровень критичности и заводится инцидент, который решается в соответствующие сроки. Это может делать человек, а может - и программа по некоторому алгоритму. Здесь важно понимать, что есть время реакции на проблему, время нахождения какого-либо решения (workaround) и время полноценного решения. До решения проблемы, что может быть весьма длительным периодом, системные события продолжают порождаться - систему мониторинга не перенастраивают.
- Системное событие - повторное срабатывание системы мониторинга для уже рассматриваемой или зарегистрированной в качестве инцидента проблеме. Для этого в промышленных системах мониторинга реализованы технологии дедупликации, которые не создают новую запись для анализа оператором, а лишь увеличивают счетчик срабатываний для уже существующей записи.
- Системное событие - следствие некорректной настройки системы мониторинга, которая либо слишком остро реагирует на некритичную проблему, либо фиксирует как проблему нормальное поведение системы. В этом случае заводится запрос на обслуживание, после выполнения которого данное системное событие больше не будет генерироваться.
- Системное событие - следствие настройки системы, которая обычно фиксирует именно проблему, но с некоторой небольшой вероятностью может быть ложным срабатыванием (false alarm). Например, можно выставить срабатывание системы мониторинга на уменьшение размера свободного диска до 10% для всех видов дисков. Однако то, что критично для 1T диска, не будет критичным для 20T диска. Данное событие может быть просто проигнорировано.



Рис. 253. Автоматическая обработка событий

При анализе системного события используется информация о конфигурации - по иерархии элементов конфигурации, о лицах, которые будут анализировать системные события и решать инциденты соответствующего типа (списки маршрутизации).

Обновление системы мониторинга

Содержание системы мониторинга определяется SLA, конфигурацией прикладного продукта, конфигурацией среды промышленной эксплуатации и выбранной платформой реализации. Процессы эксплуатации и появление новых релизов программных продуктов меняют конфигурацию и как следствие, должны привести к изменению системы мониторинга. Однако, как и для любой многоэлементной и многоплановой системы, такое изменение требует времени. После фактического изменения конфигурации продукта или среды продукта появляется период, в течение которого показания системы мониторинга будут неверны. В высоконагруженных прикладных системах это приведет к появлению вала ложных срабатываний, которые помешают службе эксплуатации выполнять свои обязанности по другим вопросам.



Рис. 254. Причины изменения системы мониторинга

Чтобы избежать такой ситуации, выполняется один из следующих подходов:

- Систему мониторинга отключают, полностью или частично, на период установки релиза и последующей перенастройки системы мониторинга. Период максимальной недоступности системы мониторинга должен быть определен в SLA.
- Изменения системы мониторинга прорабатываются одновременно с разработкой релиза, тестируются в среде вывода релиза, где есть тестовый вариант системы мониторинга и выводятся как часть пакета изменений. В этом смысле модуль настроек мониторинга продукта является частью программного продукта. Отметим, что, хотя это и самая эффективная схема с точки зрения технологического процесса, она не работает, поскольку процесс разработки и среда выпуска релиза находятся в совершенно другом контексте операций, под контролем другой команды или даже компании, работающей асинхронно по отношению к службе эксплуатации.

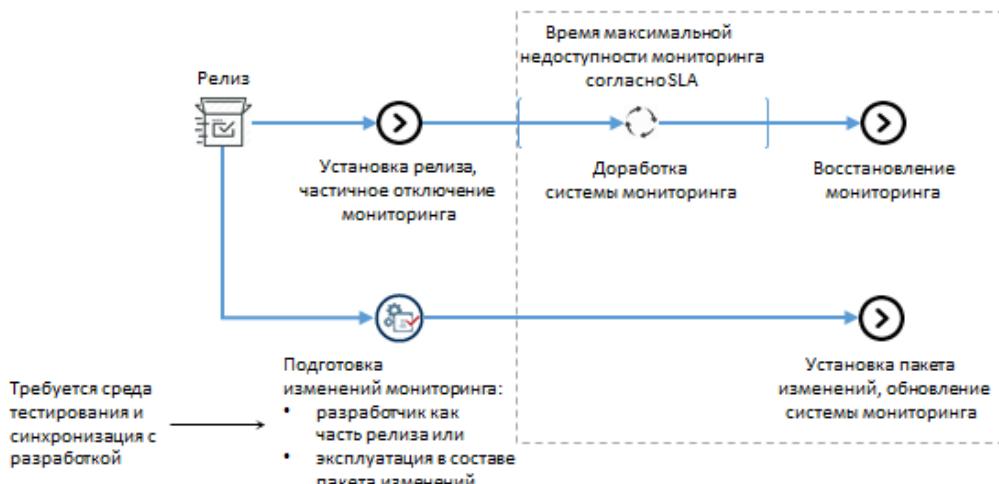


Рис. 255. Варианты обновления системы мониторинга

Мониторинг и данные управления конфигурацией

Несколько раз выше по тексту, для операций с одним элементом возникает необходимость доступа к информации других элементов - зонтичная зависимость между объектами мониторинга, вхождение прикладных процессов в прикладные продукты и системы, ответственные сотрудники и прочее. Все вместе это часть базы данных конфигурационного управления - БДКУ (CMDB), информация которой позволяет "обогащать" событие системы мониторинга дополнительной информацией, позволяющей правильно обработать данное событие.

Таким образом, правильно выстроенный процесс мониторинга интегрирован с процессом управления конфигурацией, в рамках которого ведется учет элементов конфигурации продукта и конфигурации среды промышленной эксплуатации.

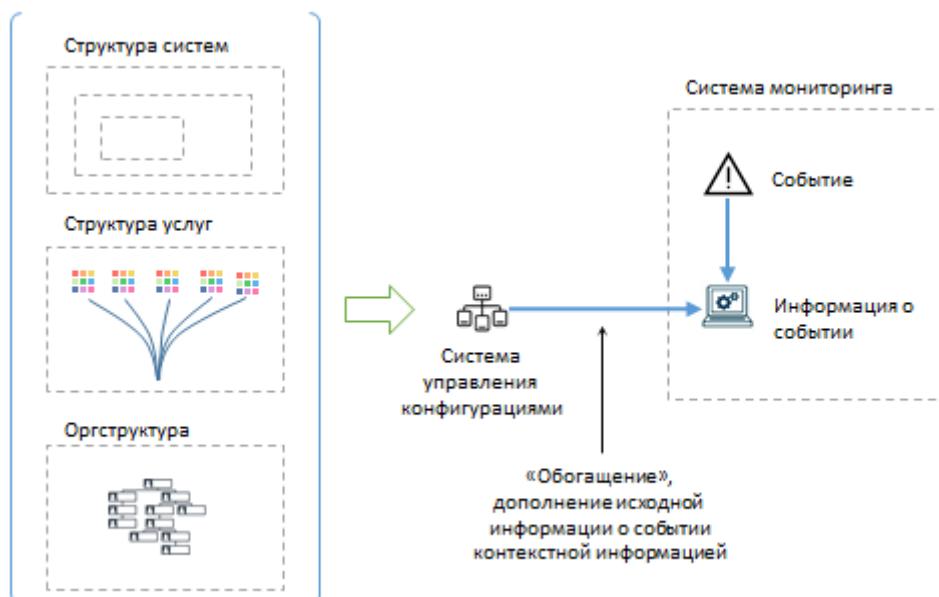


Рис. 256. Взаимодействие систем управления конфигурациями и мониторинга

Документация

ПОТРЕБНОСТЬ В ДОКУМЕНТАЦИИ

КАТЕГОРИИ ДОКУМЕНТАЦИИ

ДОКУМЕНТАЦИЯ ПРОДУКТА

ДОКУМЕНТАЦИЯ РЕЛИЗА

ОБНОВЛЕНИЕ ДОКУМЕНТАЦИИ

ФОРМАТЫ ДОКУМЕНТАЦИИ

В рамках данной главы определяется место документации, с которой работают в релизах, определяется смысл документации и действия, необходимые для появления нужной документации и как сделать создание и использование документации максимально эффективным.

Потребность в документации

О документации говорят все, ею занимаются все - от программистов до руководителей, ее сдают в рамках контрактных работ, для ее создания существуют специальные инструменты, технологии, которые применяются специально подготовленными людьми - техническими писателями. Для государственных контрактов состав и структура документов полностью определены ГОСТ 34, который повсеместно используется, к которому многие привыкли. Однако также, как и с тестированием, непонимание принципов существования и использования документации ведет к тому, что результат большого труда по созданию документации оказывается с точки зрения использования весьма скромным. Самое печальное, что в коммерческой области, где стандарты зачастую не применяются, документация иногда отсутствует полностью или пишется в абсолютно произвольной форме, иногда без какой-то логики в части того, что надо писать, что не надо.

Отсутствие документации всеми понимается как недостаток, однако, когда она появляется, оказывается, что поддерживать ее в актуальном состоянии затратно, а найти в ней нужную информацию непросто, да и хранить ее непонятно где и для кого. На тему документации есть прекрасные стандарты - как сделать ее актуальной, как ее правильно оформлять, вести версии и т.п. Но все мы знаем, что подавляющее число участников рынка ИТ все эти стандарты не применяет, поскольку затратить сил можно много, а полезный результат не появляется. Более того, стремясь соблюсти форму - например, правильно оформив таблицу рецензирования, можно совершенно не отразить содержание - рецензент будет совсем не тот, кто указан, дата ревизии не будет совпадать с фактической датой и т.п. Поэтому, зная про все эти особенности формализмов, большинство даже не смотрит на формальные атрибуты большей части документов, даже если они есть, просто не доверяя им, что приводит к еще менее качественному их заполнению.

Распространение интернета и веб-сайтов приводит многих к мысли, что документация в ее старом бумажном или даже в виде документов Word устарела, не позволяет отразить сетецентричный характер информации, обеспечить эффективную навигацию по ссылкам и поиск документов. Многочисленные системы документооборота пытаются сохранить эту форму документации даже в нашем быстро меняющемся и в чем-то хаотичном мире.

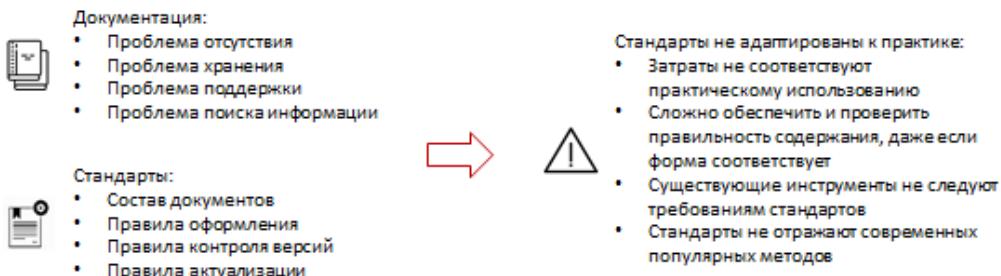


Рис. 257. Документация и стандарты

Зачем же вообще писать документацию и в какой форме? Что в ней писать, а что - нет? Какой набор документов должен быть, кто их создает, в контексте чего и когда, как обновляет и что к этому подвигает? Пункты ниже служат для того, чтобы постепенно прояснить данные вопросы.

Чтобы как-то уйти от произвольной трактовки документации по принципу "что хочу, то и пишу", для начала давайте не будем обсуждать список документов как таковой. В реальности в какой-бы то ни было форме документ - это набор отдельных текстовых фрагментов, оформленных каждый под какой-то минимальной темой, определенной заголовком этого фрагмента, опять же не очень важно, как оформленных при помощи шрифтов, размеров, отступов и т.п.

Главное для нас здесь то, что несколько предложений и рисунков (или даже анимаций) собраны в единый фрагмент. Вот структуру фрагментов мы и будем дальше обсуждать. И только определившись со структурой, мы сможем связать эту информацию с конкретными документами. Правила, которые мы сформулируем в отношении фрагментов документации, повлекут за собой правила для конкретных документов. Для удобства вместо словосочетания "фрагмент документа" будем использовать слово "информация".

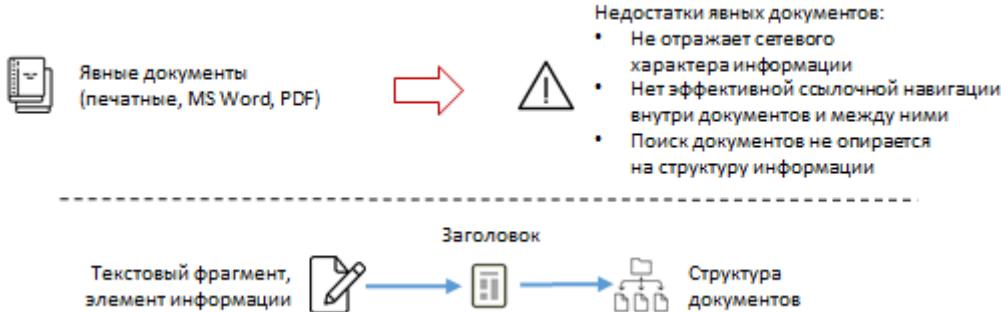


Рис. 258. Явные документы и элементы информации

Категории документации

Жизнь течет, время идет и вся информация, которая нас окружает, говорит либо о том, что есть, либо о том, что было, либо о том, что будет. Не будем обсуждать здесь все приемы художественной литературы, а ограничимся технической информацией, которая может быть:

- Созданной сейчас информацией про то как все устроено или что произошло "сейчас" - "as is". Эта информация с течением времени становится историческим фактом. К сожалению, под видом факта в такой документации может оказаться ошибочная информация или неверная интерпретация, и такую вероятность необходимо учитывать.
- Созданной сейчас информацией про то, как надо, чтобы оно было, с пониманием, когда это должно быть - "to be". Когда время становится не очень четко определенным будущим, речь идет о "концепции". Когда время этого будущего проходит, "to be" информация теряет свой смысл - она отражает историю уникальных операций, и не служит основанием для принятия решений. Устаревшие планы конечно тоже существуют в нашей жизни, но мы не будем делать их частью наших правил и рекомендаций.
- Созданной когда-либо информацией про то, как правильно должно быть то, что будет происходить в разное время, и сейчас, и потом - это то, что называется регламентами. Регламенты не совпадают с фактическим состоянием дел, но позволяют внести оценочную категорию - "хорошо" ли то, что происходит, или нет. Регламенты с течением времени могут успешно оставаться действующими и влиять на то, что происходит "сейчас".
- Созданной сейчас информацией про то, что происходило раньше, недавно или совсем давно - отчеты и исторические справки, которые имеют источником своего содержания исторические факты.

Документация о прошлом и настоящем ("as is")

- Как что-то устроено сейчас (состояние)
- Что произошло сейчас (событие)
- Что было и происходило раньше (отчеты, исторические справки)

С течением времени становится историческим фактом, в том числе искаженным:

- Некорректная фиксация факта
- Не факт, а его интерпретация, причем ошибочная или неоднозначная

Документация о будущем ("to be"):

- Известные сроки:**
- Концепции (примерные сроки)
 - Планы (определенные сроки)

Как надо, чтобы было всегда - заведомо отличается от того, что есть и будет, но вводит оценку «хорошо»/«плохо» в отношении фактов

Рис. 259. Документы о прошлом, настоящем и будущем

Достаточно удобным способом понять, когда и как создавать документы, является разграничение информации на документацию продукта, процесса и проекта:

- **Документация продукта** - это набор информации, описывающая требования к продукту, тестирования продукта, внутренний дизайн продукта и эксплуатационную документацию, в которой дано описание продукта для конечного пользователя и для сотрудников эксплуатации, которые видят интерфейс продукта, элементы дистрибутива, конфигурации среды. Документация продукта - это внутренне согласованная информация, которая описывает состояние продукта "as is".
- **Документация процесса** - делится на регламентирующую документацию, говорящую о том, как должен выполняться процесс (в части своего интерфейса или внутренней реализации), и на отчетную и фактическую информацию, фиксирующую отдельные факты в отношении выполненных операций процесса.
- **Документация проекта** - документация, регламентирующая проект целиком, документация "to be" или "as is" в отношении промежуточных состояний выпускаемого продукта (например, постановка задачи, информация о сборке). Также в рамках проекта создается документация организованных в рамках него процессов, и документация продукта в отношении выпускаемых в рамках проекта релизов.



Рис. 260. Документы проекта, процесса и продукта

Документация продукта

Релизный процесс связан с версией продукта, поэтому его касается в первую очередь документация продукта, которую рассмотрим подробнее.

Основными пользователями документации продукта являются разумеется пользователи продукта и сотрудники, отвечающие за эксплуатацию продукта. Но напомним, что продуктом является в первую очередь программный код, поэтому документация продукта существует не только в отношении дистрибутива и элементов среды продукта, но и программного кода.

Поэтому пользователями также являются члены команды разработчиков, включая программистов, аналитиков и тестировщиков. Информацию о продукте для разработчиков, которая не предназначена для конечных пользователей и сотрудников эксплуатации, назовем конструкторской документацией.



Рис. 261. Техническая документация продукта

В момент создания документация продукта описывает текущее состояние, которое в развивающемся продукте будет меняться. Такое состояние имеет смысл фиксировать в документации только если продукт в этом состоянии будет использован в течение относительно длительного времени. Такому условию соответствуют состояния, соответствующие выпущенным релизам.

Создание целостной "as is" документации продукта на промежуточные состояния практически невозможно и не нужно, поэтому существуют лишь частичные фрагменты документации на промежуточные состояния, и они не относятся к документации продукта. Более того, если релизы выпускаются часто, далеко не каждый из них может сопровождаться обновленной документацией, т.е. актуальная документация существует только на некоторые версии продукта.



Рис. 262. Документация продукта и выпуск релизов

В отношении требований здесь возможны два варианта:

- Требования являются частью документации продукта, если они существуют как постоянная система информации применительно ко всему продукту, иногда с использованием системы управления требованиями. Список реализованных требований перетекает из релиза в релиз, из контракта в контракт и служит основанием для системы регрессионного тестирования, служит ограничением для процессов анализа и проектирования. Новые требования, будучи реализованы, включаются в комплект документации продукта.
- Исходные требования контракта и проекта, требования к релизу относятся не к состоянию, а к изменению, и к документации продукта не относятся. Это исходные требования "to be" которые после завершения соответствующего периода теряют смысл. Если репозиторий требований продукта не ведется, то такие требования в качестве выполненных не переносятся в документацию продукта и существуют лишь как история операций. Те, кто выбирает такую схему работ, аргументируют выбор тем, что в качестве требований выступает, например, руководство пользователя.



Рис. 263. Требования вне и внутри проекта

Аналогично для тестирования также возможны два варианта:

- База знаний тестирования продукта, которая накапливается от контракта к контракту, от релиза к релизу, является частью продукта. Это может быть комплект сценариев и методов регрессионного тестирования продукта, который обеспечивает проверку полного комплекта требований к продукту. Если требования не ведутся, такие сценарии могут быть вместо требований, будучи информацией, производной от требований - также как документация для пользователя. Отметим, что наличие документации по тестированию продукта не означает, что при выпуске релиза все эти тесты проводятся. На практике, в любом тестировании производится выбор - какие тесты из общего наработанного пакета тестов, и в какой степени проводятся в качестве регрессионного пакета. Те же тесты, которые проводятся для новых требований, по факту успешного прохождения включаются в общий пакет документации продукта.
- Точно также, как и с требованиями, база данных тестирования продукта может не вестись и тестирование каждый раз прорабатывается отдельно для каждого контракта, проекта, релиза. В таком случае описание сценариев и методов тестирования не является частью документации продукта. Если продукт и технологии сильно меняются, это может быть понятно, но в общем случае содержание тестовых сценариев является ценной информацией о поведении продукта, которую не стоит терять.

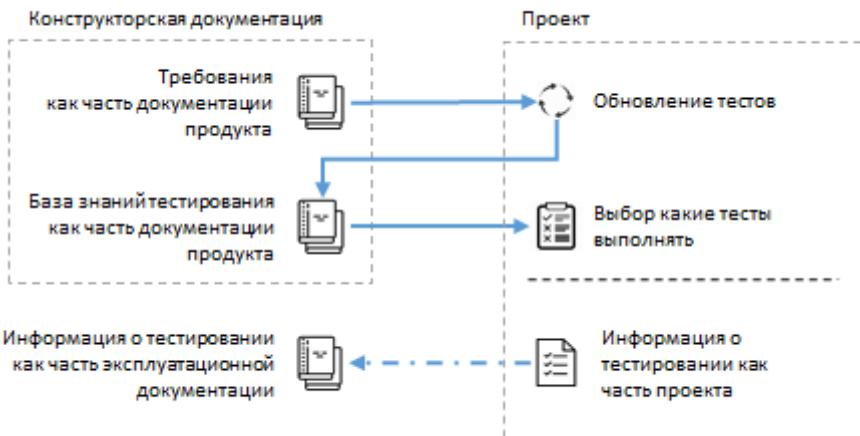


Рис. 264. Документация по тестированию в проекте и вне его

Под активностью дизайна или проектирования понимается та активность, которая решает как обеспечить выполнение требований, если есть варианты (или наоборот, решение изначально отсутствует и необходимо его найти), и как сделать создание и сопровождение продукта удобным и наименее затратным.

Обычно эту активность делят на проектирование архитектуры (HLD - Hi-Level Design), проектирование внутренних компонентов и пользовательского интерфейса (DD - Detailed Design & UI Design).

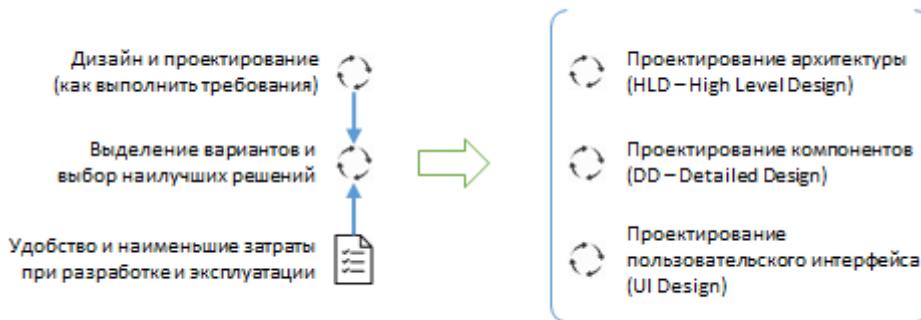


Рис. 265. Цели и структура работ технического проектирования

И хотя к каждому из пунктов так же как для требований и тестирования применимо деление на информацию проекта и продукта, разные виды документации накладывают свои особенности:

- **Проектирование архитектуры** относится ко всему продукту. Поскольку все-таки это один документ, весьма интересный техническим специалистам разработки и эксплуатации, такой документ с той или иной степенью формализации создается в каждом проекте. Единственная проблема заключается в том, что проект далеко не всегда полноценно занимается ровно одним продуктом, а может, например, создавать один продукт и частично модифицировать несколько других. Поэтому документ с архитектурой, не разделенный по продуктам или в конце концов не связанный с архитектурной документацией для немодифицируемых частей продуктов, де-факто будет документацией проекта и на вопрос через какое-то время, как устроен конкретный продукт, документированного ответа не будет. А еще через какое-то время, после естественной ротации персонала, никто не сможет это сказать и на словах - и продукт превратится в то, что называется "legacy".



Рис. 266. Ведение технической документации в привязке к продуктам

- **Документирование внутренних компонентов** – дело, не часто используемое из-за того, что нужно оно только проектной группе, передача работ другой команде считается негативным сценарием, а развитые инструменты разработки автоматически фиксируют такую документацию. Поэтому такого рода документы встречаются в основном как проектные постановки задач, или очень выборочная информация, о которой так просто не догадаешься по коду, в качестве документации продукта.

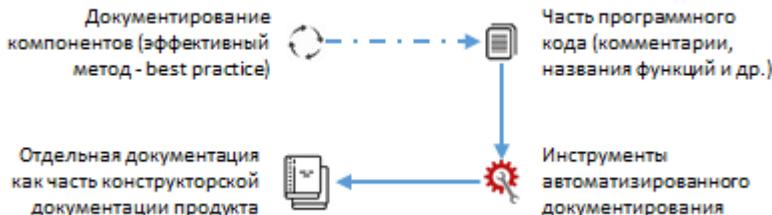


Рис. 267. Ведение технической документации по компонентам

- **Документирование пользовательского интерфейса** - в качестве "as is" является частью эксплуатационной документации продукта (руководство пользователя и администратора), а в качестве "to be" - обычной постановкой задач. Эксплуатационная документация является обязательством пользователю со стороны продукта - какие функции и как должны работать. Если в документации написано одно, а в продукте делается совсем другое, пользователь может инициировать заведение инцидента, поскольку для него именно этот документ первичен. Ситуация аналогична ценнику в магазине, который имеет приоритет перед фактической ценой продукта в торговой системе.

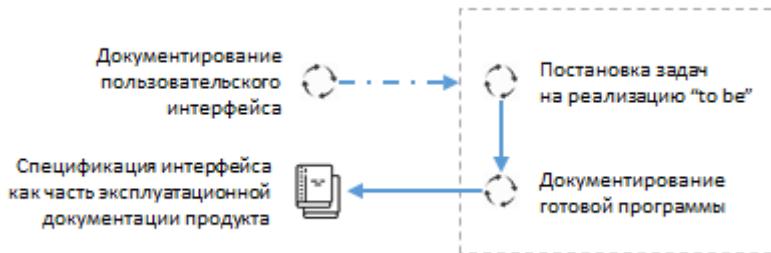


Рис. 268. Документирование пользовательского интерфейса

Кроме упомянутой технической документации, в которую входит конструкторская документация и эксплуатационная документация, к документации продукта можно отнести также документы нетехнического характера, которые как правило выводятся за рамки проекта разработки и релиза программного продукта:

- **Коммерческую документацию** - финансовые операции, связанные с продуктом, опции конфигурации для продажи и т.п.
- **Маркетинговую документацию** - презентации, анализ рынка и т.д.
- **Методическую документацию** - как лучше использовать продукт, как организовать связанные процессы и т.п.

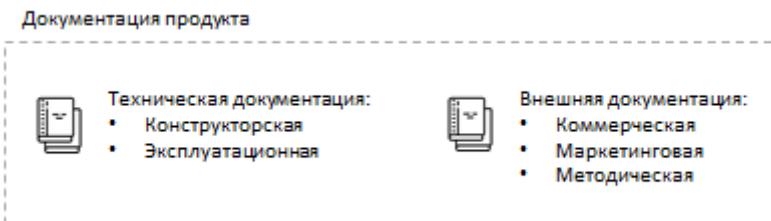


Рис. 269. Документация продукта в целом

Документация релиза

Релиз подобен мини-проекту или операции процесса в случае регулярных релизов и служит для вывода в среду эксплуатации определенного комплекта изменений. Эти изменения переводят продукт из состояния предыдущей версии в состояние новой версии. Поэтому, говоря о документации релиза, мы имеем в виду две категории документов:

- **Информация об изменениях релиза** - процессная или проектная информация
- **Информация о конечном состоянии** - документация продукта

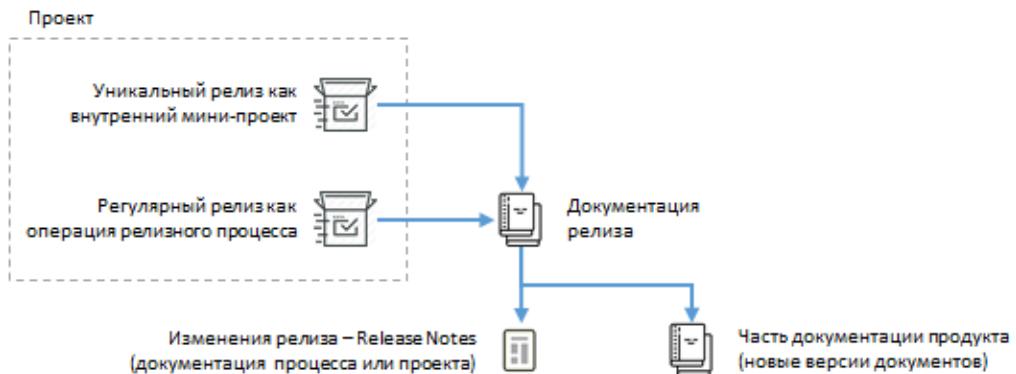


Рис. 270. Состав документации релиза

Информация об изменениях - то, что обычно называется Release Notes, содержит следующее:

- Состав изменений на логическом уровне в виде функциональных изменений для пользователей и для администраторов. Детальность информации может быть разной - от перечисления одной строчкой на изменение до деталей уровня руководства пользователя и администратора.
- Состав изменений на логическом уровне в части того, какие ошибки исправлены. Под ошибками здесь подразумеваются как минимум те проблемы, которые были зафиксированы ранее в качестве инцидентов. Ошибки, которые обнаружены самим разработчиками в ходе тестирования релиза, и которых нет в среде промышленной эксплуатации, в данный перечень помещаться не должны. Ошибки, которые были в эксплуатации, исправлены, но не были заведены в виде инцидентов, могут как присутствовать в списке, так и не присутствовать, в зависимости от политики и сути ошибок.
- Известные ошибки - список ошибок, которые были обнаружены, но не были исправлены при тестировании релиза, или те, что были заведены на предыдущие версии продукта в качестве инцидентов, но до сих пор не решены. Понятно, что такой список может быть весьма обширен и обладает негативной окраской. Поэтому возможно лучше дать пользователям непосредственный доступ к баг-трекеру, чтобы они смогли увидеть какие ошибки в каждый момент существуют.
- Информация об алгоритме установке релиза - которой должно быть достаточно, чтобы выполнить установку данного релиза. Это не означает, что от релиза к релизу придется повторять подробное описание каких-то одинаковых методов. Информация об установке может ссылаться на другие документы, в том числе выпущенные в предыдущих релизах, на страницы сайта компании. Правило только одно - не должно быть внешних источников, без которых релиз не получится установить, и на которые нет ссылки в данной информации, кроме информации, которая изначально определена как внешняя по отношению к продукту. К последнему относятся значения параметров конфигурации, а также любые инфраструктурные элементы и настройки, выбираемые по усмотрению эксплуатацией.
- Состав файлов релиза - то, что подлежит изменению на физическом уровне, а также файлы, средства установки или данные для него.

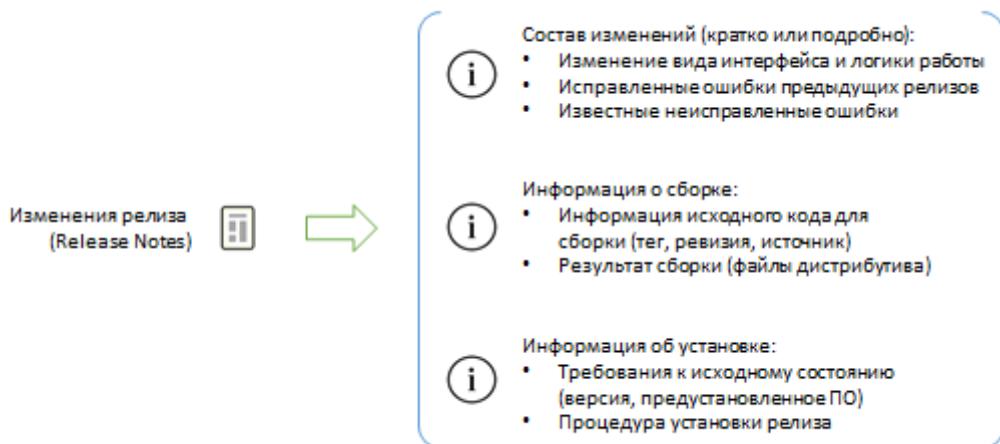


Рис. 271. Содержимое Release Notes

Информация о состоянии - новая версия документации продукта:

- Эксплуатационная документация.** Данная документация входит в состав дистрибутива. Если в данном релизе документация не актуализируется, соответственно ничего нет. Также возможен промежуточный вариант - часть документов обновляются, а часть - нет. В идеале не обновившиеся документы должны оставаться корректными, однако разработчик может сэкономить на выпуске документации для промежуточных релизов и возникнет противоречие между документацией и реальной программой.
- Конструкторская документация.** Данную документацию, также, как и исходный код, не помещают в дистрибутив, однако в случае заказного продукта это является потенциальной проблемой, поскольку передача другими каналами приводит к рассогласованию документации и дистрибутива версии продукта. Впрочем, обычно ситуация еще хуже - данную документацию в полноценной форме не сдают вообще, независимо от того, что написано в контракте. Это делает клиента зависимым от поставщика для последующих контрактов и процесс смены поставщика всегда протекает очень болезненно.



Рис. 272. Передача документации в ФАП

Обновление документации

Понимание сути и структуры документации, а также основных событий с точки зрения обновления позволяет сформулировать минимальные правила ведения документации:

- Необходимо определить политику поддержки документации - насколько часто она будет обновляться, в привязке к каким релизам.
- Чтобы можно было обновить эксплуатационную документацию спустя несколько релизов, необходимо в описании изменений иметь всю существенную информацию, но то, что может быть легко найдено (например, диалоговая форма), является в логическом описании изменения возможным, но необязательным.
- Конструкторскую документацию необходимо вести, если вероятно сохранение проектных решений от проекта к проекту.
- Документация продукта должна находиться в постоянном использовании, чтобы быть качественной и актуальной.
- Версии документов продукта и релиза должны совпадать с версиями релизов. Внутренние промежуточные версии могут вестись, но не являются обязательными. Проектные документы, напротив, должны иметь внутренние версии, если их изменение является критичным фактором (например, Техническое Задание).



Рис. 273. Основные аспекты обновления документации

Форматы документации

Формат документации, как это ни странно, имеет значение гораздо большее, чем просто удобство или факт следования определенному стандарту. В формат входят не только размеры, отступы и шрифты, но и интерактивность документов, возможность совместного редактирования, возможность гибкой организации ссылок внутри документа и между документами с соответствующей навигацией.

Создание хорошего документа - это серьезный и профессиональный труд, но, чтобы этот труд не пропал даром, документ должен использоваться. Чем больше используется документ, чем больше у него читателей и чем чаще они обращаются к документу, тем более они будут требовательны к автору, тем выше будет качество документа. Никому не нужный документ по определению некачественный. Чтобы повысить качество документа, необходимо сделать так, чтобы он в максимальной степени начал использоваться. Ну и разумеется, для того, чтобы им захотели пользоваться, надо правильно подойти к его содержанию.

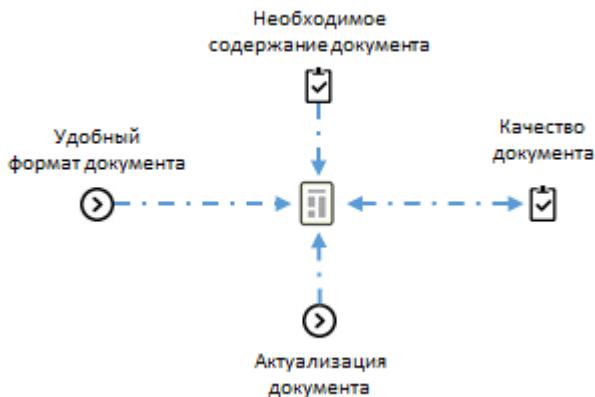


Рис. 274. Максимальное использование документа

Настоящую революцию в документации породил формат Wiki, в котором в любой момент чтения можно перейти к редактированию документа, а сразу после сохранения новая версия становится доступной для других пользователей. Т.е. из операций пользователей исчезло технологически тяжелые отправка документа по почте, проверка версии документа, и даже режим рецензирования, поскольку wiki-формат позволяет отслеживать измененные фрагменты в документе и их авторство. Более того, это породило другую культуру в рабочих группах - "Заметил ошибку? Исправь ее, если уверен".

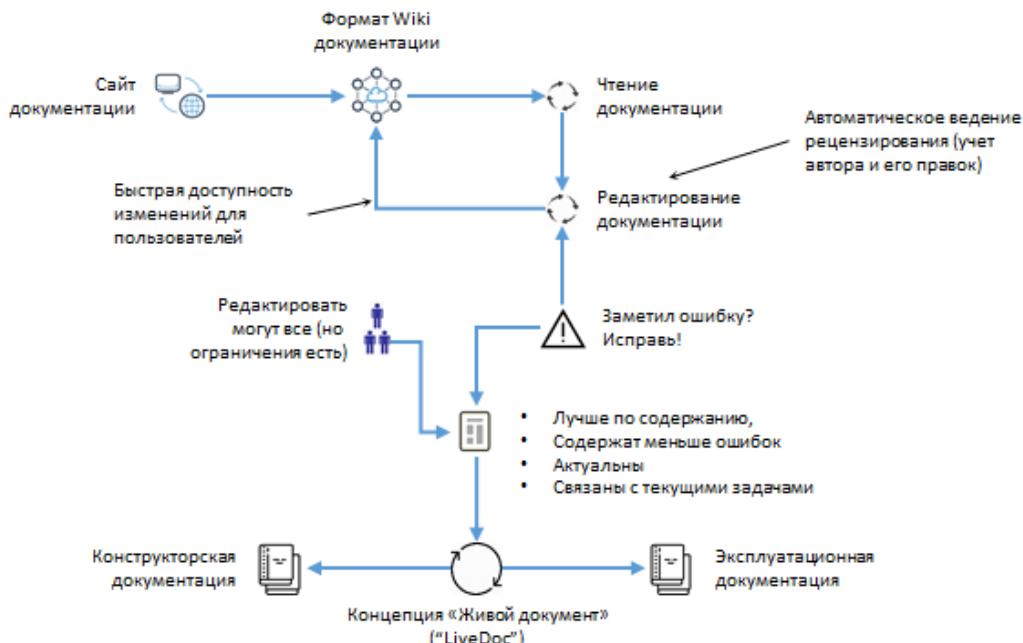


Рис. 275. Концепция LiveDoc

Это породило совсем другие принципы доступа - "редактировать нельзя, только если действительно нельзя", вместо "редактируют только авторы и рецензенты". Документ стал гораздо ближе к текущей работе членов группы, он стал их настоящим рабочим инструментом, актуальным и обновляющимся как часть обычной работы. Поэтому эта технология носит название LiveDoc - живой документ.

Да, это не ко всей документации применимо, например - определенно не к финансовой. Однако и конструкторская, и эксплуатационная, и даже регламентирующая документация во многих компаниях и для многих продуктов обновляются теперь именно так.

Релизные процессы

ПОТРЕБНОСТЬ В ПРОЦЕССАХ

МОДЕЛЬ ITIL ДЛЯ РЕЛИЗНЫХ ПРОЦЕССОВ

РЕЛИЗЫ И ИЗМЕНЕНИЯ

РОЛИ В РЕЛИЗНЫХ ПРОЦЕССАХ

СТРУКТУРА И СВЯЗИ ПРОЦЕССОВ

ИНИЦИАЦИЯ РЕЛИЗА

УПРАВЛЕНИЕ СОСТАВОМ РЕЛИЗА

РАЗРАБОТКА ЗАДАЧ РЕЛИЗА

ИНТЕГРАЦИЯ ПРОГРАММНОГО КОДА

СБОРКА ДИСТРИБУТИВА

УПРАВЛЕНИЕ СРЕДАМИ

ТЕСТИРОВАНИЕ РЕЛИЗА

ФИНАЛИЗАЦИЯ РЕЛИЗА

ПОДГОТОВКА К УСТАНОВКЕ

ИЗМЕНЕНИЕ СРЕДЫ ПРОМЫШЛЕННОЙ ЭКСПЛУАТАЦИИ

ЗАКРЫТИЕ РЕЛИЗА

В рамках данной главы определяется место процессного управления при работе с релизами, описываются процессы или части процессов, которые рекомендуется выстроить для применения данной методологии.

Потребность в процессах

Зачем нужен процесс? Эта тема достаточно хорошо освещена в литературе по бизнес-анализу и в стандартах, однако для специалистов, выполняющих или отвечающих за практическую деятельность, будет достаточно тяжело выделить время, чтобы пройтись по полным руководствам или методологиям, со всеми их терминами, аспектами и нюансами. Краткие справки зачастую грешат перечислением составляющих и не говорят о самом главном.

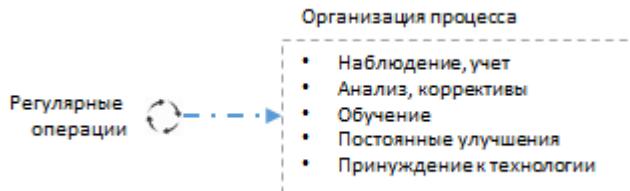


Рис. 276. Организация процесса

Поэтому кратко о процессах стоит сказать следующее. Если есть круг регулярной деятельности, за который отвечает определенный менеджер, привлекая к своим задачам множество других людей, то чтобы понять, что и как они делают, чтобы научить или даже заставить выполнять все эти работы правильно, используется такая формальная организационная технология как "процесс". О процессе стоит в первую очередь знать следующее:

- Менеджер может быть руководителем подразделения, которое выполняет работы, а может быть "горизонтальным менеджером", привлекая сотрудников из разных подразделений (что означает "горизонтальный" процесс). Главное в данном случае - что это деятельность, за которую кто-то определенный отвечает ("хозяин" процесса) и что хотя менеджер может и быть единственным исполнителем в своем процессе, процесс не будет полноценным для одного человека, поскольку научить себя и заставить себя - это неформализуемая деятельность.

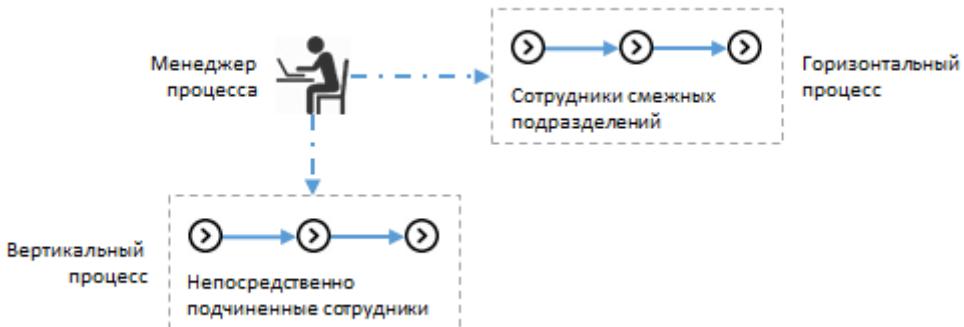


Рис. 277. Вертикальный и горизонтальный процесс

- Процесс отвечает за регулярную деятельность, и основан на той идее, что надо одни и те же задачи делать одинаково, используя наиболее эффективный алгоритм, который в процессе выполнения задач постепенно становится понятен, что приводит к постоянному улучшению процесса. Здесь становится ясно, что никакого улучшения и поиска оптимального алгоритма без менеджера процесса не будет и такая неуправляемая деятельность не соответствует ключевым свойствам процесса, как технологии. Тем не менее это частая ошибка при выделении процессов - называть процессом группу задач, не имеющую единого управления.

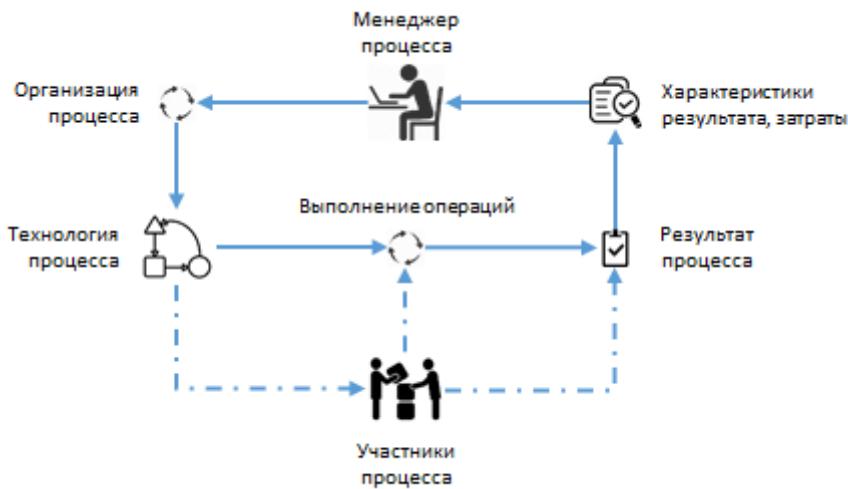


Рис. 278. Организация и технология процесса

- В каждой задаче процесса есть вход, выполняемые работы и выход. Вход поступает извне процесса, выход - отдается как результат процесса, а выполняемые работы - внутреннее дело процесса. При этом выход определяется и оценивается по тем целям, которые ставятся перед менеджером, а внутренние работы могут приводить к постановке задач для другого процесса, что делает последний вспомогательным по отношению к первому процессу.
- Менеджер может подчиняться другому менеджеру, подразделение может являться частью другого подразделения и это порождает иерархию процессов. Структура менеджмента - это и есть структура процессов.

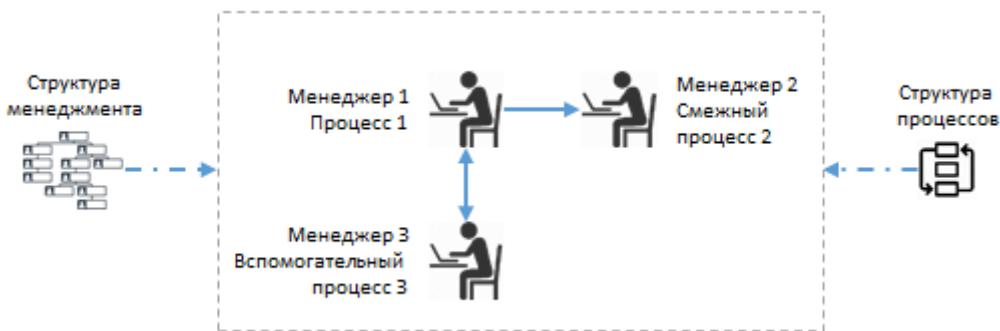


Рис. 279. Структура процессов и менеджмента – две стороны одной медали

- Регулярность процесса позволяет оценивать не конкретную задачу, а деятельность за период, что дает инструмент для оценки менеджера со стороны его руководителя. Для объективности важно так или иначе объективно фиксировать результаты отдельных задач процесса, чтобы затем вычислить метрики процесса (KPI) для оценки внешней и внутренней эффективности.
- Если некоторая регулярная деятельность не соответствует указанным выше критериям, ее стоит называть "активностью" в рамках какого-то другого процесса.

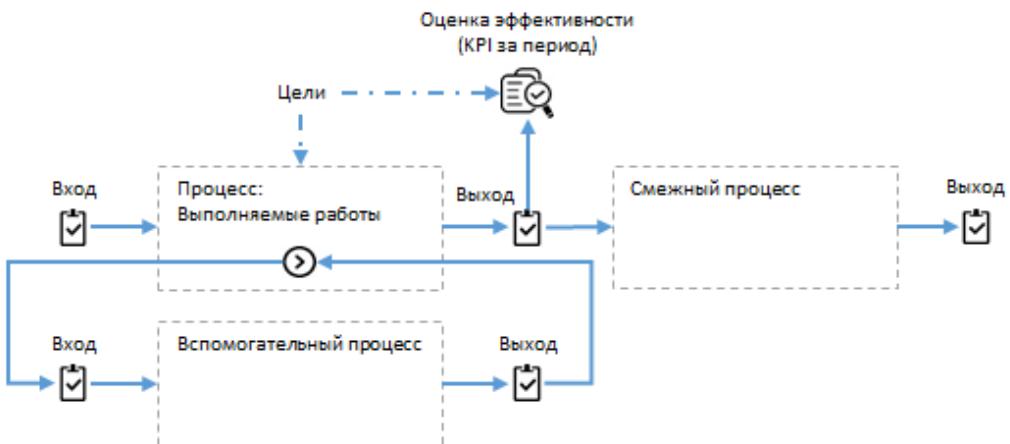


Рис. 280. Смежные и вспомогательные процессы

Выполнение задачи "правильно" означает выполнять ее в соответствии с правилами, которые в идеале должны сочетать следующее:

- Минимальные затраты на единицу результата, что связано с таким понятием как "внутренняя эффективность".
- Минимальные затраты на единицу пользы от результата, приносимой "клиенту" процесса - тому, кто дает процессу на вход задачу, и что связано с таким понятием как "внешняя эффективность".
- Качество и гарантия получения результата в нужный срок, к чему стоит относиться как к универсальному средству для достижения внешней эффективности. Соответственно - качество, которое не влияет на внешнюю эффективность - по определению лишнее. Это еще одна типичная ошибка при построении процессов - борьба за качество, которое никому не нужно.



Рис. 281. Внешняя и внутренняя эффективность

Модель ITIL для релизных процессов

С каждым годом методологий и стандартов становится все больше, что-то устаревает, что-то возникает или пополняется. Сравнение методологии UM со всеми существующими аналогами в части релизных процессов не только немалый труд, но и объемный результат, который будет интересно читать

лишь специалистам в области стандартов. С другой стороны, сопоставление с одной из существующих известных моделей позволит перейти от уже знакомых понятий к понятиям методологии УМ.

Для сравнения выбрана методология ITIL, которая также примерно соответствует группе стандартов ISO/IEC 20000. Правда, необходимо учесть направленность ITIL на оказание конечных услуг, т.е. на сервисы, за которые отвечает эксплуатация программных систем. Это процессный подход, который относительно слабо описывает проектный характер разработки и тиражируемые программные продукты. Но, поскольку УМ ориентирована на взаимодействие разработки и эксплуатации, обеспечение регулярности операций с целью повышения эффективности и качества результата, то при описании релизных процессов сравнение с ITIL будет достаточно удобным.

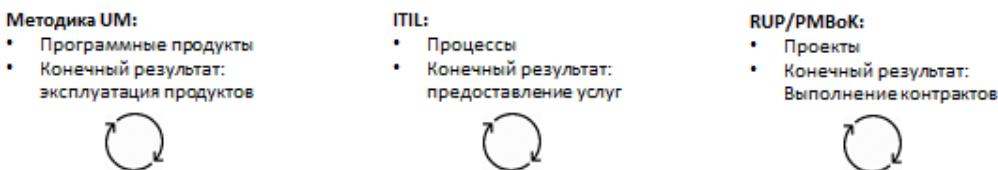


Рис. 282. Направленность разных методологий

ITIL определяет специализированный процесс для релизов:

- **Service Transition:** группа процессов преобразования услуг:
 - **Release and Deployment Management** - Управление релизами и развертыванием

Также ITIL определяет связь с другими процессами:

Входы:

- **Service Design:** группа процессов проектирования услуг
- **Service Operation:** группа процессов эксплуатации услуг (таких как Управление инцидентами, Управление проблемами)
- **Service Transition:** группа процессов преобразования услуг:
 - **Change Management** - Управление изменениями
 - **Transition Planning and Support** - Планирование и поддержка преобразования (как часть Project Management)
 - **Service Validation and Testing** - Подтверждение и тестирование услуг
 - **Service Asset and Configuration Management** - Управление сервисными активами и конфигурациями

Выходы те же, что и входы, плюс несколько дополнительных процессов:

- **Service Transition:** группа процессов преобразования услуг:
 - **Change Evaluation** - Оценка изменения
- Процессы вне организации:
 - **Customer Process** - Процесс пользователя
 - **External Supplier Process** - Процесс внешнего поставщика

Подпроцессы:

- **Release Management Support** - Поддержка управления релизами
- **Release Planning** - Планирование релизов
- **Release Build** - Сборка релизов
- **Release Deployment** - Установка релизов
- **Early Life Support** - Первичная поддержка
- **Release Closure** - Закрытие релизов

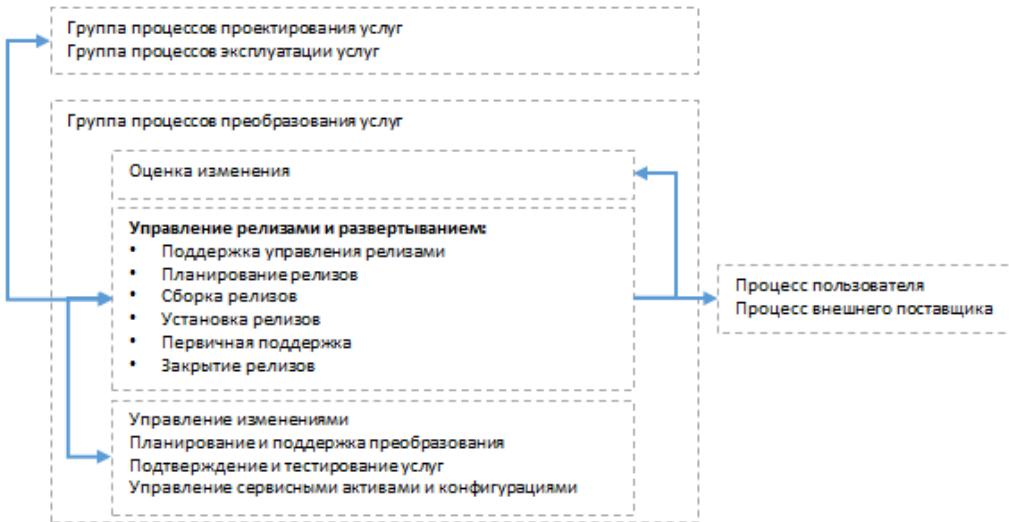


Рис. 283. Структура процессов ITIL, связанных с релизами

Понятно, что речь идет скорее про активности, чем про процессы. В целом читатель, который попытается разобраться, что же ему на самом деле рекомендует эта методология, скорее всего останется неудовлетворенным - слова слишком общие, а структура противоречива и слабо связана со структурой менеджмента в реальных организациях. Типичной ошибкой при внедрении ITIL является попытка создать процессы равными тем, что определены в ITIL, что чревато раздуванием штатов и чрезмерной формализацией, в результате чего общая как внешняя так и внутренняя эффективность структуры становится весьма низкой.

Если посмотреть на стандарт ISO/IEC 20000, который был создан на основе ITIL, то он определяет требования к процессам контроля (Процесс управления конфигурациями, Процесс управления изменениями, Процесс управления релизами), в которых тоже много общих слов, но есть относительно неплохие требования касательно процесса управления релизами, с которыми и стоит сравнить методологию UM:

- «Политика релиза, устанавливающая частоту и типы релизов, должна быть задокументирована и согласована.»
 - Это то, что в данной книге называется группой релизных циклов. (см. Операции - Выпуск релизов)
- «Поставщик услуг должен, совместно с бизнесом, планировать релизы услуг, систем, программного обеспечения и оборудования. Планы развертывания релиза должны быть согласованы и утверждены всеми соответствующими сторонами, например, заказчиками, пользователями и персоналом эксплуатации и поддержки.»
 - План установки релиза появляется как артефакт при планировании релиза, результат объединения конфигурационных изменений по отдельным задачам, планирования их выполнения как упорядочения и распределения по исполнителям и планирования установки в целом. (см. Операции - Установка релизов)
- «Процесс управления релизами должен включать в себя способ отмены или исправления релиза в случае неудачи.»
 - В данном случае речь про механизм отката (см. Операции - Установка релизов)
- «В планах должны быть записаны даты и результаты релизов, а также ссылки на связанные запросы на изменения, известные ошибки и проблемы.»
 - Здесь отмечено, что при планировании релизов необходимо планировать состав изменений в терминах выпускаемых функций и исправления ошибок. (см. Операции - Выпуск релизов)
- «Процесс управления релизами должен передавать соответствующую информацию в процесс управления инцидентами.»

РЕЛИЗНЫЕ ПРОЦЕССЫ

- Здесь затрагивается два важных момента - при выпуске релиза, некоторые известные ошибки, выявленные при тестировании, намеренно остаются не исправленными, и необходимо их заранее завести как инциденты. Также, если при установке возникли проблемы, которые не были исправлены как часть нормального процесса, или которые привели к нарушению согласованных условий и правил установки релиза, такие проблемы также стоит заводить как инциденты, чтобы учесть их в последующих операциях. (см. Операции - Установка релизов)
- «Должно оцениваться влияние запросов на изменение на планирование релизов.»
 - Что имеется в виду, здесь до конца не понятно, но можно предположить, что речь идет про то, что запросы на изменение как-то должны учитываться в планах выпуска релизов, чтобы адекватно выбирать изменения для релизов, с учетом возможности их реализации и критичности связанных проблем.
- «Процедуры управления релизами должны включать в себя актуализацию и изменение информации о конфигурациях и записей об изменениях.»
 - Это достаточно общее требование, поскольку не очень понятно, насколько детально фиксировать информацию и как обеспечивать качество этих записей. Но при полноценной автоматизации управления конфигурациями, это требование будет автоматически выполнено. (см. База данных конфигурационного управления)
- «Управление срочными релизами должно осуществляться согласно определенному процессу, имеющему интерфейсы с процессом управления срочными изменениями.»
 - Очень правильное требование, которое обязывает готовить релизные циклы и механизмы внесения срочных изменений заранее. (см. Операции - Выпуск релизов)
- «Для сборки и тестирования (до распространения) всех релизов, должна быть создана контролируемая среда приёмочного тестирования.»
 - Так же безусловное верное требование. Контролируемость означает, что известно какая в ней конфигурация, обеспечено соответствие тестируемому релизу и его текущему плану установки. (см. Операции - Среды)
- «Релиз и распространение должны быть спроектированы и реализованы так, чтобы обеспечить поддержку и сохранение целостности программного обеспечения и оборудования во время компоновки, сборки, перемещения релиза, его доставки до места установки и в ходе самой установки релиза.»
 - Поскольку речь идет о сквозном процессе со множеством операций и участников, то данное требование де-факто является самым сложным. Методология UM решает этот вопрос через единую метамодель, обеспечивающую операции по управлению кодом, сборки, установки и поддержки сред (см. Введение)
- «Успешность или неудача внедрения релизов должны измеряться. Измерения должны включать в себя вызванные релизом инциденты. Анализ успешности релиза должен включать в себя оценку влияния на бизнес, эксплуатацию ИТ и ресурсы персонала поддержки, а также должен предоставлять входные данные для плана улучшения услуг.»
 - Это то, что происходит при закрытии релиза. (см. Операции - Установка релизов)

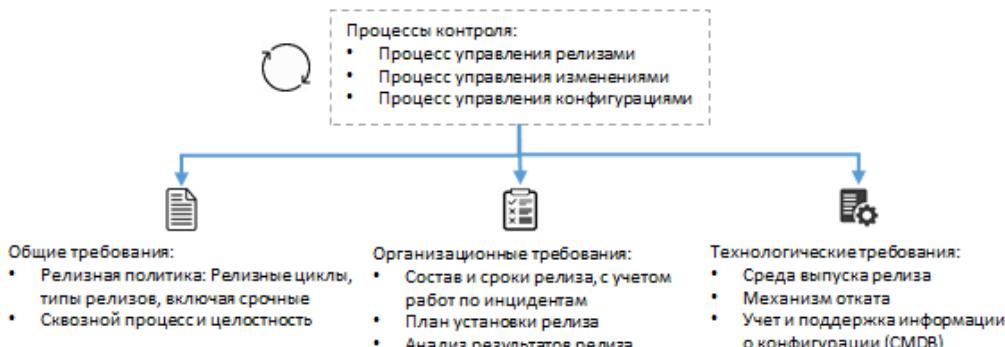


Рис. 284. Требования стандарта ISO 20000

В целом стандарт хорошо стыкуется с методологией UM, но в последней операции с релизами существенно более детально описаны, большее внимание уделяется программному коду, его структурированию на продукты, более конкретное описание сред и операционной модели в средах.

Релизы и изменения

Говоря про релизы, мы обычно имеем в виду некоторый дистрибутив, который выпускается и устанавливается. Говоря про изменения, мы имеем в виду либо изменения в поведении приложения, в его функциях, либо изменения объектов в среде промышленной эксплуатации. Все три объекта этих операций являются существенно разными. Для каждого из видов объектов можно составить свой алгоритм выполнения, прохождения жизненного цикла и таким образом это три разных активности, которые могут соответствовать трем разным процессам. Задача на изменение функции появляется тогда, когда речь не идет о конкретном релизе или изменении в среде, релиз начинают собирать и тестировать тоже могут задолго до того, когда в процессе управления изменениями появится какая-то информация об этом релизе, а управление изменениями может относиться к операции изменения среды, которая ни к какому релизу отношения не имеет.



Рис. 285. Взаимодействующие процессы

Таким образом, в общем случае речь идет о трех взаимодействующих процессах или активностях, операции которых порождают цепочку из трех звеньев - то, что называется технологической цепочкой: изменение продукта - выпуск релиза - изменение среды.

Роли в релизных процессах

Чтобы показать, как устроены участки релизных процессов, как выполняется взаимодействие, мы определим несколько ролей. Некоторые роли могут быть совмещены в одном человеке, но в некоторых случаях, указанных ниже, совмещение приведет к нарушению системы мотивации - конфликту целей, и, как следствие, нарушению даже правильно определенного процесса.

Итак, роли нам понадобятся следующие:

- **Разработчик** - тот, кто прорабатывает изменение, выполняет коммит в репозиторий кода, готовит информацию для плана установки релиза, обновляет документацию по продукту. Фактически это могут быть в организации совсем разные люди - технический писатель, программист, системный инженер, системный аналитик. Т.е. применительно к конкретной задаче или релизу это будет в общем случае не один человек, а целая группа лиц.
- **Тестировщик** - тот, кто тестирует изменение. Здесь важно сказать, что в команде все участвуют в тестировании, и в первую очередь программист. Однако тестирование, которое выполняет программист, является частью задачи кодирования и называется отладкой, при которой проверяются не только конечные требования, но и внутренние, связанные с дизайном. Отсутствие отладки практически неизбежно приведет к проблемам, к увеличению циклов исправление-тестирование и к понижению эффективности процесса разработки. Тестировщик же - это независимый от разработчика человек, который отдельно тестирует данную задачу,

причем в первую очередь с точки зрения конечных требований. Если отладка и тестирование были выполнены, выполняется то, что называется проверкой в 4 глаза (4-eye), при которой вероятность того, что оба ошиблись и не проверили одно и то же. Т.е. несмотря на то, что все люди ошибаются, при добросовестной работе два человека способны выпустить качественный продукт. Одним из вариантов является ситуация, когда роль тестировщика выполняет другой разработчик - не тот, что делал эту задачу. Т.е. важно не название позиции, а независимая повторная проверка. Тестировщик заинтересован в качестве среды тестирования и может заниматься ее контролем.

- **Релиз-инженер** - тот, кто обеспечивает наличие корректной среды тестирования и корректного дистрибутива, а также тот, кто отслеживает, что все задачи включенные в релиз, были установлены в среду тестирования и успешно протестированы. В случаях, когда это невозможно, решение принимает куратор - допустить ли отклонение или нет. Не желательно совмещать роль релиз-инженера и разработчика, но тестировщик вполне может выполнять эту роль.
- **Менеджер разработки** - тот, кто принимает решение об отклонениях со стороны разработки тогда, когда это еще не носит критического характера. В зависимости от того, насколько крупная и сложная команда участвует в выпуске, менеджером может быть ведущий разработчик или менеджер проекта.
- **Куратор** - тот, кто принимает решение об отклонениях со стороны разработки на поздних фазах релиза, когда риски, порождаемые отклонениями, особенно велики. Куратор, занимая более высокое положение, может признать проблемы разработки, за которую отвечает менеджер разработки и отказаться от выпуска рискованного или некачественного релиза. Для менеджера разработки это будет нарушением его плана, за который он отвечает, и он может пойти на неоправданный риск.
- **Менеджер сервиса** - тот, кто отвечает за уровень сервиса в среде промышленной эксплуатации, согласовывает установку релиза, и имеет право одобрить отклонение в процессе установки. В частности, когда никак не удается починить установленный релиз и нужно принять решение о прекращении попыток и откате к работоспособному состоянию.
- **Инженер эксплуатации** - лицо, выполняющее изменение среды промышленной эксплуатации. Подчиняется менеджеру сервиса по вопросам, связанным с установкой релиза.
- **Системный инженер** - администратор, отвечающий за оборудование, на котором функционирует та или иная среда.
- **Заказчик** - лицо, которое выступает от имени владельца среды промышленной эксплуатации, заинтересованного в использовании системы. Организация, которая отвечает за эксплуатацию, может не быть владельцем и как следствие, заказчиком. Функции заказчика, связанные с релизами, могут выполняться его представителем или могут быть делегированы бизнес-аналитику или куратору на стороне разработки, менеджеру сервиса на стороне эксплуатации. Заказчик определяет, когда и какие функции нужно выпускать в среду промышленной эксплуатации, а также в каком виде. Именно заказчик выступает со стороны пользователей. Со стороны разработки перед заказчиком отвечает менеджер разработки, а со стороны эксплуатации - менеджер сервиса.

Также, как уже было сказано, на практике может существовать выделенный релиз-менеджер, но его роль важна с точки зрения постановки и управления релизным процессом. Если релизный процесс идет не так как положено, то релиз-менеджер в этом принимает участие, но отклонения самого релизного процесса мы опустим, поскольку в них нет специальных правил, о которых стоило бы говорить. А вот отклонения выпуска или установки релиза, связанные с разработкой и эксплуатацией, являются типичной ситуацией в релизном процессе и их эффективная обработка является частью методов релизного процесса.

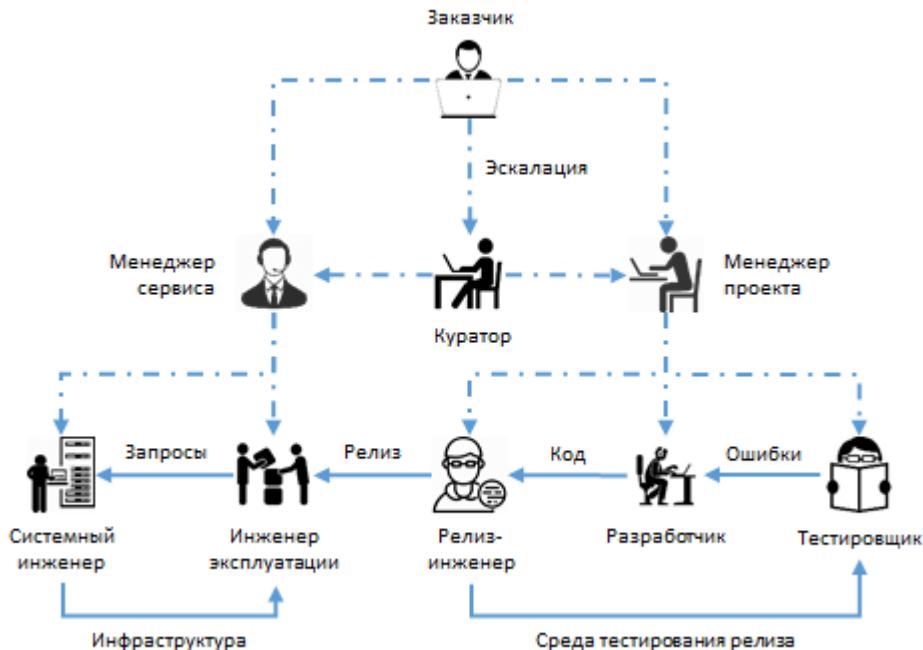


Рис. 286. Ролевая система для описания релизных процессов

Структура и связи процессов

Место релизных процессов в общей системе процессов и их фактическая структура зависит от организации и конкретного продукта. Тем не менее можно сформулировать рекомендации по составу активностей и их связям, оставляя на усмотрение организации, в состав каких процессов входят эти активности, и кто управляет этими процессами.

Внешняя среда для релизных процессов на верхнем уровне абстракции - это процессы разработки, которые берут на себя взаимодействие с заказчиком (в том числе и по поводу релизов) и коммерческую сторону вопроса, а также процессы эксплуатации, которые связаны как с заказчиком, так и с конечными пользователями, выполняющими операции в среде конечной эксплуатации. В реальной ситуации разработка, эксплуатация, заказчик, пользователи и лица, вовлеченные в релизные процессы, могут быть совершенно по-разному распределены по организациям и их подразделениям. Методология УМ не пытается определить данное распределение.

Все приведенные ниже активности релизных процессов связаны как организационной, так и с технической составляющей (как минимум с кодом или дистрибутивами), что обеспечивает отражение реально необходимых операций. На верхнем уровне релизные процессы представлены следующими активностями:

- **Инициация релиза** - начало жизненного цикла релиза, создание (пустого) дистрибутива
- **Управление составом релиза** - определение состава функциональных и нефункциональных изменений продукта, а также связанных с ними изменяемых объектов среды
- **Разработка задач релиза** - та часть разработки, которая идет под управлением релизных процессов
- **Интеграция программного кода** - то, что обеспечивает корректное состояние кода в условиях выпуска множества релизов и неизбежных проблем, связанных с разработкой программного кода
- **Сборка релиза** - создание производных объектов из исходного кода и помещение в одно место всех результирующих объектов релиза

- **Управление средами** - создание и поддержка сред для тестирования программного продукта, установка сборок релизов, контроль соответствия тестовых сред и сред промышленной эксплуатации
- **Тестирование релиза** - операции, которые обеспечивают качество программного продукта за счет поиска и исправления ошибок в программном коде
- **Финализация релиза** - операции, в результате которых появляется дистрибутив релиза, готовый к установке в среду промышленной эксплуатации
- **Подготовка к установке** - часть плана установки релиза, выполняемая заранее
- **Изменение среды промышленной эксплуатации** - операции, которые приводят к изменению алгоритмов конечных операций, что является целью релиза
- **Закрытие релиза** - операции подведения итогов релиза и необходимые для успешного выполнения последующих релизов

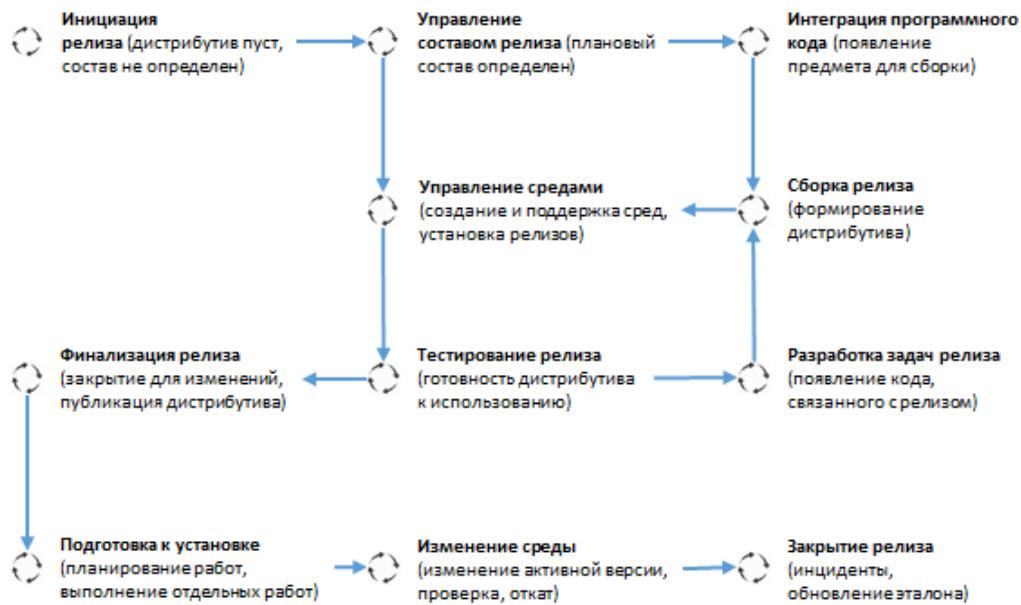


Рис. 287. Активности релизного процесса и связи между ними

Инициация релиза

Регулярный и нерегулярный плановый релиз, внеплановый релиз должны иметь определенный порядок инициации, который дает уверенность в том, что без необходимости релиз не выпускается, и среда промышленной эксплуатации не подвергается лишнему риску.

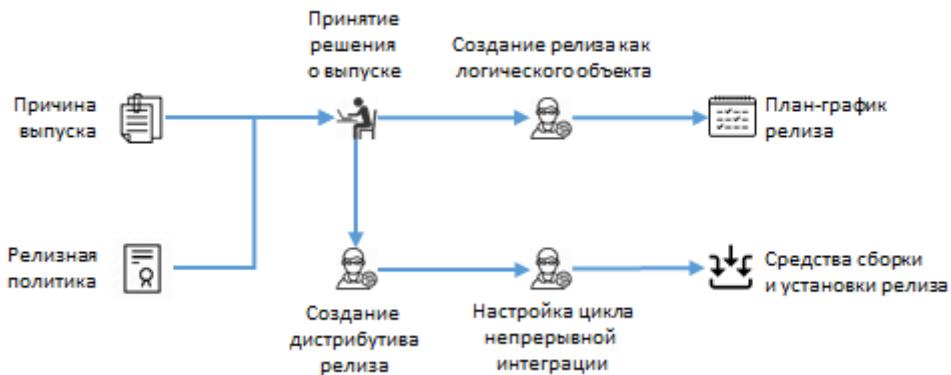


Рис. 288. Работы при инициации релиза

Управление составом релиза

Состав релиза определяется составом обновляемых элементов (для неполного, инкрементального релиза) и составом изменений в функциях и нефункциональных параметров (включая исправление ошибок, которые с ними связаны). Отсутствие контроля состава релиза приведет к рискам и увеличению затрат на выпуск релиза, и как следствие, сдвигу сроков и проблемам с качеством. На разных фазах релиза порядок внесения изменений в состав релиза должен быть разный.



Рис. 289. Работы при управлении составом релиза

Разработка задач релиза

В самом простом случае после включения в план релиза задачу могут начать и завершить в данном релизе. Но для регулярных циклов разработка по задаче может начаться в одном цикле, а

закончиться в другом. Если разработка носит достаточно неопределенный и рискованных характер, порождает существенные изменения в программном продукте, то ее зачастую планируют в терминах календарных сроков, а не в терминах релизов. В результате к моменту, когда формируется план релиза, конкретная задача уже может быть полностью или частично сделана. Однако к моменту, когда надо выпускать релиз, задача должна быть все-таки готова. Если же все-таки она не готова, задачу или исключают из релиза, или разбивают на части, выпуская только то, что уже готово. Неготовый код по причинам внутренней реализации может также остаться в выпущенном релизе, в случае, когда он физически присутствует, но не выполняется.



Рис. 290. Работы по разработке задач релиза

Интеграция программного кода

Активности по интеграции программного кода выполняются в связи с выпуском планового или внепланового релиза. Данные активности должны протекать согласованно, чтобы не потерять код и не внести изменения, которые не планировалось выполнять.

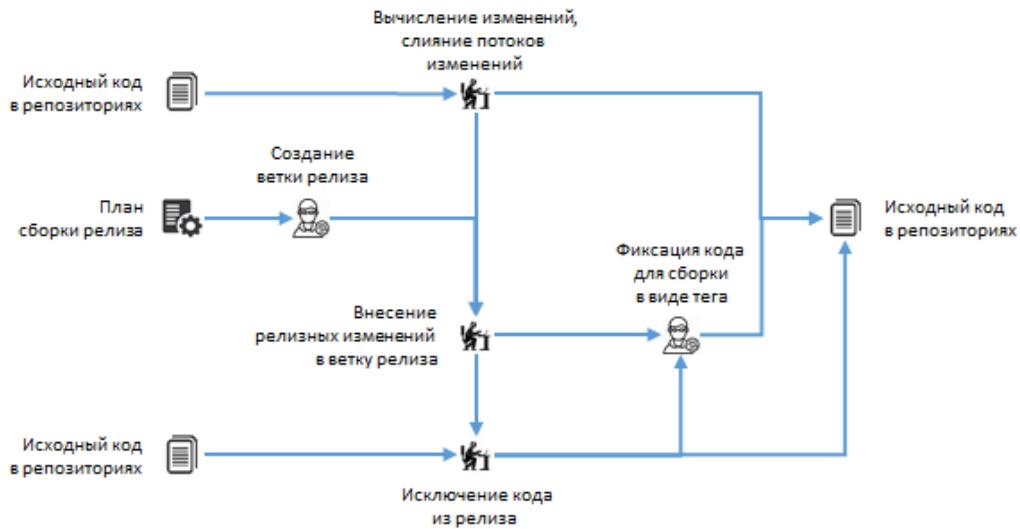


Рис. 291. Работы в части интеграции программного кода

Сборка дистрибутива

Сборка дистрибутива обеспечивает формирование итогового комплекта файлов и переносит все результаты разработки в одно место. Также сборка дистрибутива связана с созданием производных объектов и формированием сопроводительной документации к дистрибутиву. Производные объекты - это как элементарно результаты компиляции исходного кода, так и более сложные случаи, когда из исходных объектов порождаются составные в соответствии с какой-то технологией (например, веб-архивы) или просто архивы для упрощения перемещения и развертывания элементов дистрибутива.

Целью формирования сопроводительной документации к дистрибутиву обозначить состав и роль элементов дистрибутива, а также это может быть необходимость зафиксировать источники исходного кода и алгоритмы преобразования, что важно для обеспечения трассируемости, прослеживания, откуда взялся определенный элемент дистрибутива. Последнее обеспечивает воспроизведение сборки дистрибутива и техническую возможность локализовать и исправить проблему.

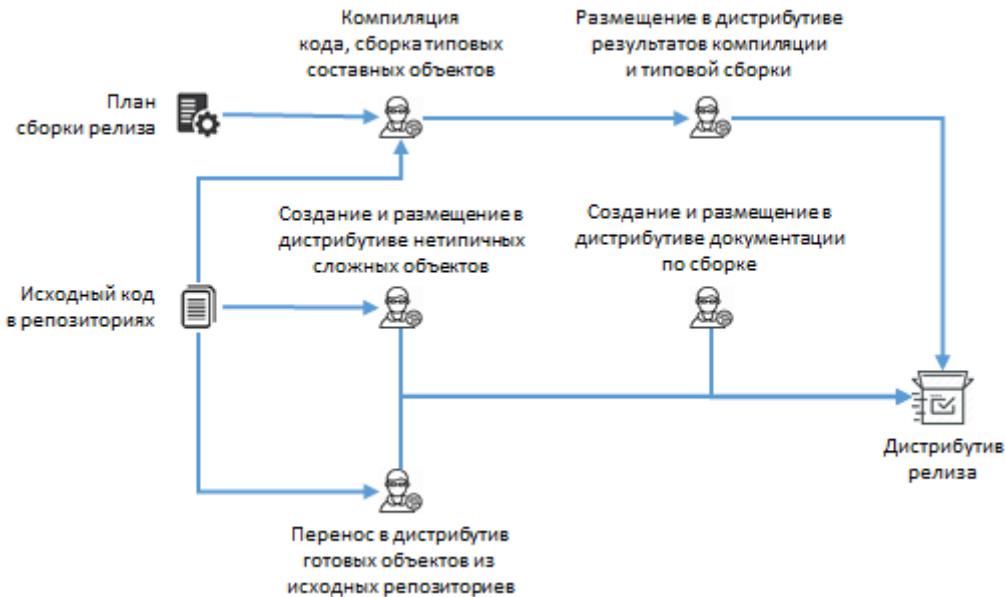


Рис. 292. Работы по сборке релиза

Управление средами

Чтобы протестировать релиз, необходимо создать специальную тестовую среду и установить туда сборку релиза. Поскольку релиз прикладного продукта - это всего лишь изменение его состояния и часть среды (еще есть неизменяемая релизом часть продукта, базовое и системное ПО, инфраструктура), то остальная часть среды должна быть в корректном состоянии. Тестирование в некорректной среде смысла не имеет, поскольку выявленные отклонения могут не являться ошибками, а настоящие ошибки могут быть не обнаружены. В идеале среда тестирования строго равна текущему состоянию среды промышленной эксплуатации плюс установленный туда релиз. Этого достигнуть сложно, поскольку в среде промышленной эксплуатации выполняются операции, там зачастую используются существенно большие ресурсы, применяются специальные технологии для обеспечения безопасности и т.д.

С учетом всех сложностей, данная активность должна в существующих условиях добиться создания среды, наиболее адекватной с точки зрения тестирования релиза. Если есть параллельные работы, требующие разной конфигурации, то таких сред может быть создано несколько. После создания в среде тестирования необходимо аналогичным образом повторять изменения среды промышленной эксплуатации, чтобы поддерживать данную среду в актуальном состоянии.

Также в ходе тестирования выявляются ошибки и сборка повторяется. Для установки сборки необходимо удалить (дезинсталлировать) предыдущий релиз и заново установить новый. Если есть инкрементальные операции и откат не подготовлен (а напомним, что речь идет про состояние разработки и нельзя рассчитывать на наличие и корректность процедуры отката), то придется либо выполнить частичный откат, либо формировать среду заново с удалением всего, установкой текущего состояния среды промышленной эксплуатации и установки новой сборки релиза. Что, разумеется может быть очень трудоемко и требовать много времени. Если нет автоматизации и нужной технологии, то это мешает выполнять достаточное количество итераций тестирования и неизбежно ведет к снижению качества выпущенного релиза.

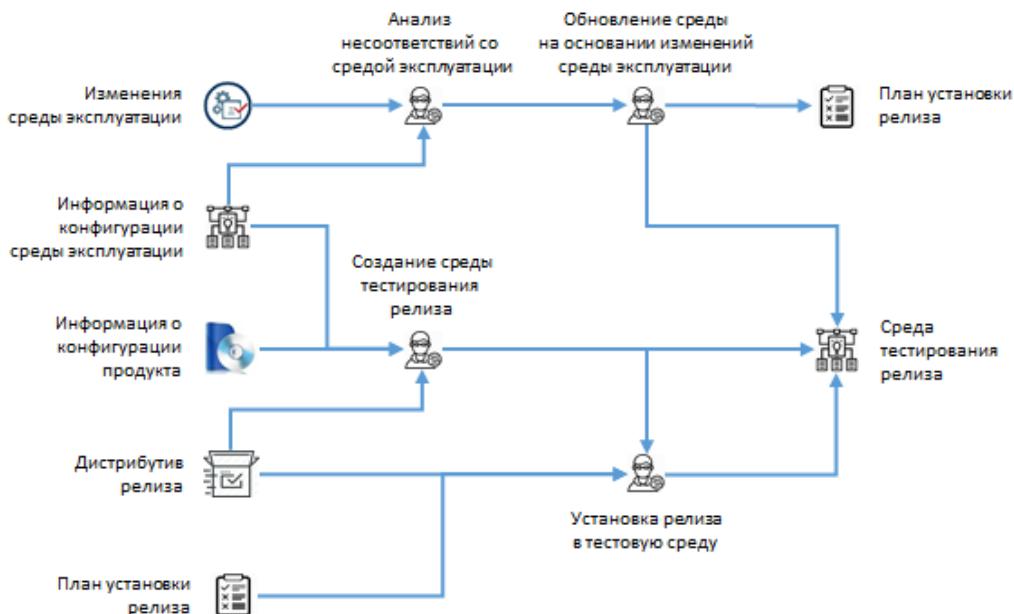


Рис. 293. Работы по управлению средами

Тестирование релиза

Поскольку релиз в общем случае состоит из нескольких задач, которые выполняются и тестируются в разное время, а в результате тестирования найденные ошибки и несоответствия требованиям устраняют, внося изменения в программный код, что порождает итерации. Но поскольку объектом сборки, установки и тестирования является не задача, а весь программный продукт, оптимальное тестирование релиза не является просто объединением отдельных операций тестирования задач. Итерации тестирования и исправления кода связаны с отдельными задачами, в то время как регрессионное тестирование всех или критических функций, тестирование производительности, отказоустойчивости относится ко всему релизу целиком.

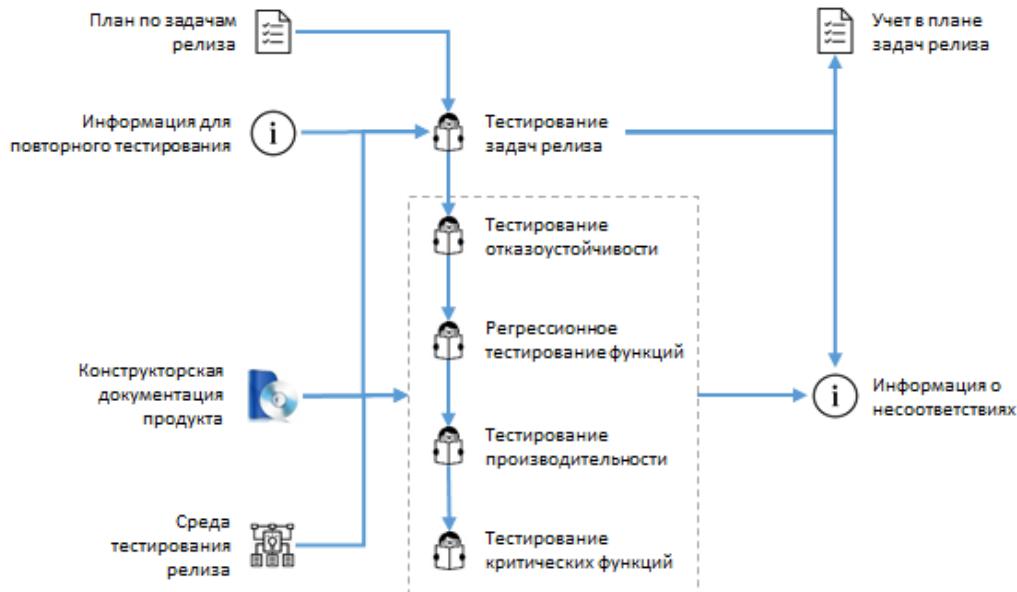


Рис. 294. Задачи тестирования при выпуске релиза

Финализация релиза

Перед тем как устанавливать релиз в среду промышленной эксплуатации, необходимо добиться его готовности к установке, что на практике означает наличие критериев готовности и тех последних недолгих, но важных операций на стадии выпуска, которые выполняют необходимый контроль и обеспечивают завершенность и взаимную согласованность дистрибутива, включая окончательный состав изменяемых элементов, состав изменений и план установки релиза. Отсутствие четкости и контроля на этом периоде неизбежно приведет к ошибкам, которые могут свести на нет все предыдущие усилия по выпуску релиза.



Рис. 295. Работы при финализации релиза

Подготовка к установке

Только в самых простых случаях установка релиза является операцией, которую начинают и быстро заканчивают. В общем случае план установки релиза состоит из подготовительных операций, которые могут выполняться весьма продолжительное время, поскольку не влияют на текущие операции в работающей системе, и операций обновления, которые в той или иной мере влияют на выполняемые процессы, создавая риски, ограничения по выполняемым функциям, производительности или просто остановку сервиса, что является наиболее часто применяемым методом, поскольку наименее рискованно и при этом просто с технической точки зрения.



Рис. 296. Работы в ходе подготовки к установке релиза

Изменение среды промышленной эксплуатации

Именно в этот момент начинается выполнение шагов, которые могут легко нарушить обязательства эксплуатации перед заказчиком и пользователями системы. Обстоятельства могут быть уникальными и непредвиденными, что при несоблюдении некоторых правил может обрушить всю выстроенную систему технической поддержки и привести к длительной неработоспособности продукта. Говоря о гарантированном уровне сервиса, эксплуатация должна не просто выделить ресурсы для решения возникающих проблем, но и обладать технологиями и инструментами, которые помогут оперативно справиться с этими проблемами (даже непредвиденными), или не допустить их.



Рис. 297. Работы при изменении среды промышленной эксплуатации

Закрытие релиза

Успешная установка релиза не является последним шагом в жизненном цикле релиза, поскольку необходимо обновить систему мониторинга, проинформировать пользователей о закрытии их инцидентов, обновить эталонный дистрибутив и в конце концов зафиксировать факт закрытия релиза в системе управления релизами. Именно факт закрытия релиза дает возможность окончательно уточнить план следующего релиза, и освободить ресурсы, связанные с данным релизом, если они были специально для него выделены.

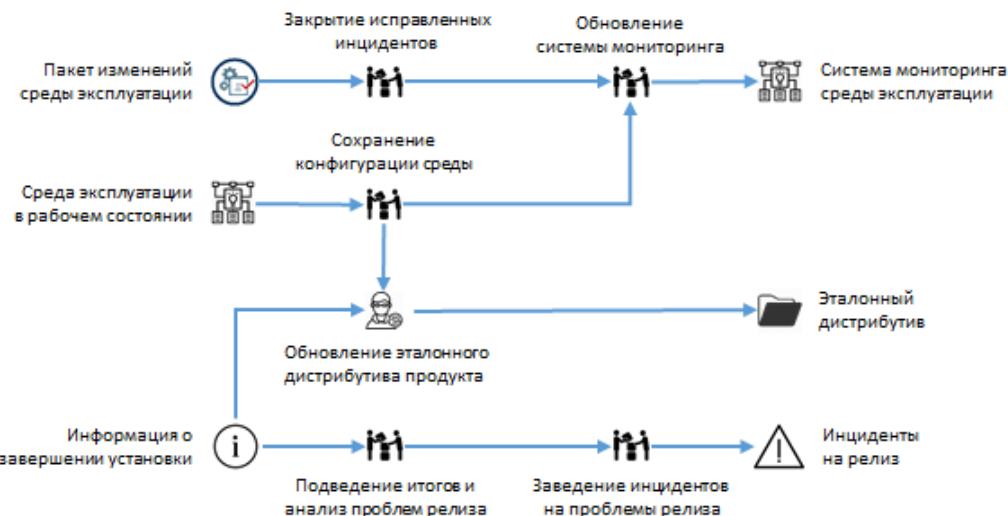


Рис. 298. Работы при закрытии релиза

База данных конфигурационного управления

ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ПРОДУКТЫ

ЭЛЕМЕНТЫ В СОСТАВЕ ПРОГРАММНЫХ ПРОДУКТОВ

СРЕДЫ ПРОГРАММНЫХ ПРОДУКТОВ

ЭЛЕМЕНТЫ ИНФРАСТРУКТУРЫ ДЛЯ ИНФОРМАЦИОННЫХ СИСТЕМ

КОНФИГУРАЦИОННЫЕ ФАЙЛЫ И ПАРАМЕТРЫ КОНФИГУРАЦИИ

В рамках данной главы описывается база данных конфигурационного управления и конкретизируются характеристики конфигурационных элементов. При создании элементов необходимо учитывать их характеристики, правильно и единообразно трактовать их смысл.

Информационные системы и программные продукты

В данном пункте определяется принципиальная структура элементов верхнего уровня конфигурации информационных систем. При этом все элементы относятся к единой базе данных конфигурационного управления (БДКУ) и одной организационной единице верхнего уровня, которая отвечает за ведение данной базы данных.

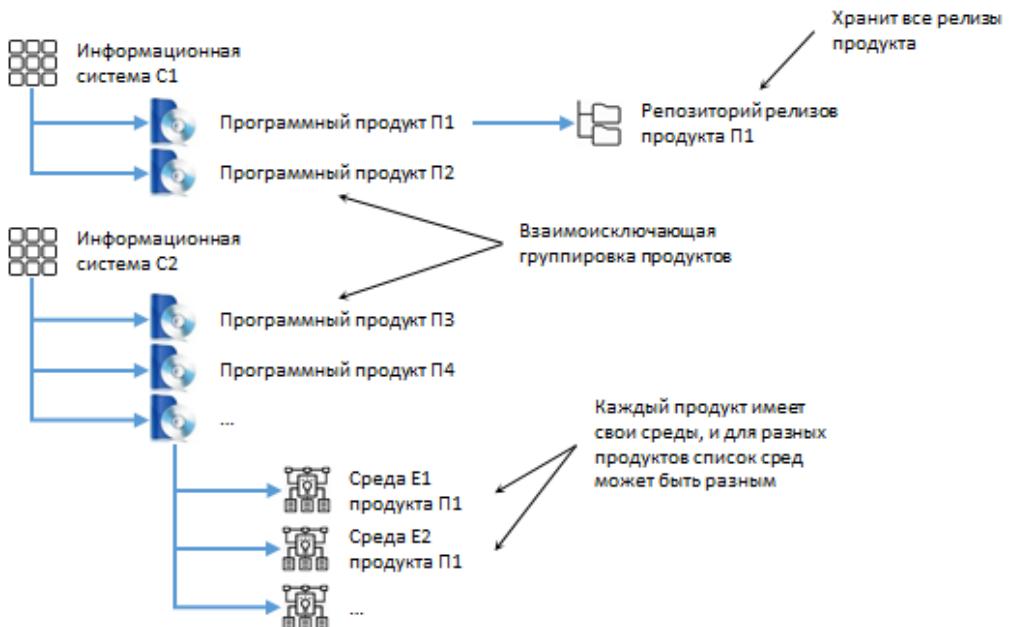


Рис. 299. Информационные системы и программные продукты

Тип объекта/атрибут	Семантика	Комментарий
Информационная система		См. Введение
Идентификатор	Способ идентифицировать элемент	Идентификация происходит на верхнем уровне БДКУ, отражает линию продуктов или портфель продуктов подчиненной организационной единицы
Логическое название	Группа продуктов	Область ответственности соответствующей организационной единицы или обобщенная функция линии продуктов
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. Если внутри системы программный продукт находится в процессе определения, то система тем не менее может считаться определенной. Если же действующей системе с существующими действующими продуктами присвоили статус неопределенной, то все ее продукты также считаются неопределенными. С неопределенными продуктами не могут выполняться никакие операции, кроме операций определения конфигурации.
Программный продукт		См. Введение.
Идентификатор	Способ идентифицировать	Идентификация происходит на верхнем уровне

	элемент	БДКУ, указание системы не должно быть обязательно
Тип продукта	Классификация	Прикладной продукт, платформенный продукт, технологический продукт
Логическое название	Функция продукта	Обобщенное описание функции продукта
Последний выпущенный релиз	Версия релиза	Последний плановый или внеплановый релиз со статусом полной готовности к установке
Установленный релиз	Версия релиза	Применимо, если нет свободной установки, обозначает релиз, процедура установки которого полностью завершена
Текущий плановый релиз	Версия релиза	Ближайший плановый релиз, который еще не готов к установке
Текущий внеплановый релиз	Версия релиза	Применимо, если возникла необходимость выпуска внепланового релиза, обозначает еще не готовый внеплановый релиз
Последний основной релиз	Версия релиза	Применимо, если используются основные релизы, обозначает последний выпущенный релиз с полным дистрибутивом
Следующий основной релиз	Версия релиза	Применимо, если используются основные релизы, обозначает ближайший запланированный релиз с полным дистрибутивом
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. С неопределенными продуктами не могут выполняться никакие операции, кроме операций определения конфигурации.

Элементы в составе программных продуктов

В данном пункте определяются подчиненные элементы программных продуктов, для управления конфигурацией при операциях сборки и выпуска релизов.

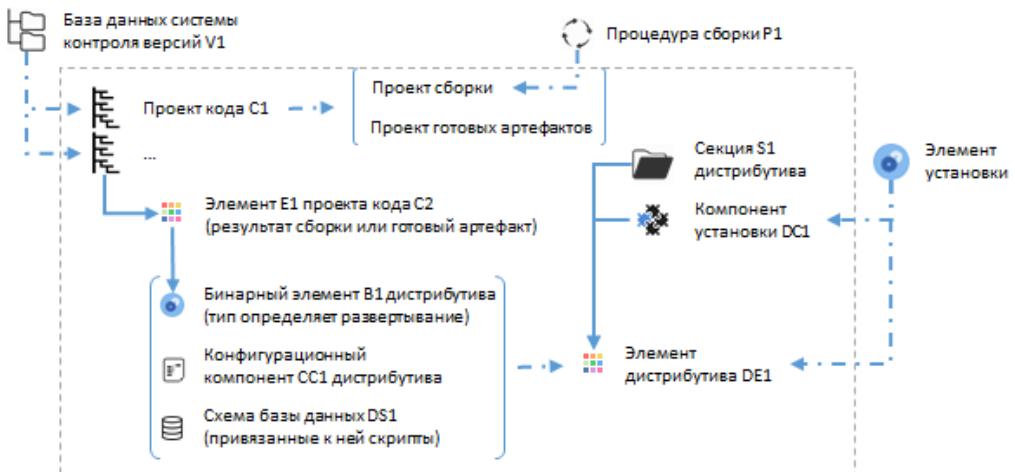


Рис. 300. Элементы в составе программных продуктов, 1, Элементы в составе программных продуктов

Тип объекта/атрибут	Семантика	Комментарий
Проект сборки программного продукта		См. Введение
Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Логическое название	Название	Описательная характеристика проекта
Информация о размещении	Способ найти и выгрузить программный код	Совокупность атрибутов, определяющая сервер системы контроля версий, репозиторий системы контроля версий, размещение проекта сборки в репозитории, доступ к файлам проекта сборки
Информация о сборке	Способ выполнить сборку программного кода	Совокупность атрибутов, определяющая алгоритм сборки, результатирующие артефакты, их местонахождение после выполнения сборки
Информация о зависимостях	От каких проектов зависит сборка	<p>Для успешной сборки проекта могут быть нужны результаты сборки других проектов данного или других продуктов. С другой стороны, сборка данного проекта в общем случае делает невалидными результаты сборки проектов, которые зависят от данного проекта.</p> <p>Это не так, если каждая сборка порождает новую версию, а зависимость идет от версий, однако в таком случае требуется явная (ручная) модификация зависимостей, что при выпуске релиза и множественных сборках неудобно.</p>
Проект готовых артефактов программного продукта		Вырожденный вид проекта сборки в случае, если как таковой сборки нет и в репозитории просто хранятся элементы, которые нужно поместить в дистрибутив или использовать при сборке других проектов. Еще более простым вариантом является вариант, при котором элементами проекта являются артефакты, не подлежащие изменению. В этом случае репозиторий может располагаться не в системе контроля версий.

Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Логическое название	Название	Описательная характеристика проекта
Информация о размещении	Способ найти и выгрузить артефакты	Совокупность атрибутов, определяющая расположение проекта готовых артефактов и доступ к его файлам
Элемент проекта программного продукта		То, что используется из результатов сборки проекта или его готовых элементов для помещения в дистрибутив или для сборки других проектов. Если связь одного проекта сборки с другим проектом настраивается на уровне стандартной автоматической сборки, например, через maven/nexus, то такие элементы не идентифицируются в БДКУ.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри проекта
Логическое название	Название	Логическое название элемента
Информация о хранении	Определение атрибутов физического хранения полученного элемента в исходном месте	Способ определить какой именно каталог или файл, с каким расширением, какого типа соответствует данному элементу в результатах сборки или в готовом виде в репозитории системы контроля версий. Название файла может определяться динамически на основании режима сборки, версии релиза, текущего основного релиза.
Признак промежуточного элемента	Признак	Указание на то, что элемент не попадает в дистрибутив продукта, но является результирующим элементом проекта, используемым при сборке другого проекта
Элемент дистрибутива	Идентификатор элемента дистрибутива	Если элемент входит в дистрибутив, то указание на идентификатор элемента дистрибутива
Элемент дистрибутива программного продукта для серверов приложений		Элемент, который может быть размещен в дистрибутиве программного продукта для серверов приложений. Если в дистрибутиве программного продукта отсутствует хотя один такой элемент, то дистрибутив является инкрементальным. К данному пункту не относятся файлы дистрибутива, служащие для изменения баз данных
Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Тип элемента	Классификация	Бинарный элемент (один файл и в дистрибутиве и в среде), архив (один файл в дистрибутиве и распакованные файлы архива в среде), конфигурационный компонент (один или несколько конфигурационных файлов)
Информация о хранении	Определение атрибутов физического хранения полученного элемента в	Способ определить какой именно каталог или файл, с каким расширением, какого типа соответствует данному элементу в результатах сборки или в готовом виде в дистрибутиве продукта. Название файла может определяться динамически на основании режима сборки, версии релиза, текущего основного релиза.

	дистрибутиве	
Секция дистрибутива	Идентификатор секции дистрибутива	Указание, в какую часть дистрибутива помещается данный элемент
Происхождение	Информация о происхождении элемента	Элемент может получен в результате сборки проекта сборки, выгрузки из проекта готовых артефактов, положен в дистрибутив вручную или получен определенным образом из других элементов дистрибутива.
Схема базы данных программного продукта		Логическая схема базы данных, которая в конкретной среде может быть физически размещена на одном или нескольких серверах баз данных внутри определенной физической схемы базы данных. Несколько логических схем могут быть размещены в одной физической схеме. К схеме привязываются файлы дистрибутива, связанные с изменением баз данных.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта.
Тип базы данных	Информация о типе базы данных	Атрибуты, определяющие тип СУБД, в которой может быть размещена данная схема, и соответственно формат файлов, содержащих изменения.
Секция дистрибутива программного продукта		Для формирования дистрибутивов сложных продуктов удобно делить дистрибутив на секции. Каждый элемент дистрибутива относится к одной определенной секции.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Информация о хранении	Способ указать на размещение секции в дистрибутиве	Например, каталог верхнего уровня
Информация о содержании	Способ указать на допустимые файлы внутри секции	К файлам относятся элементы дистрибутива для серверов приложений (каждый элемент попадает в одну конкретную секцию) и файлы для определенного для данной секции подмножества схем баз данных. Одна и та же схема базы данных может задействоваться в нескольких секциях дистрибутива, в том смысле что разные файлы разных изменений, располагающиеся в разных секциях дистрибутива, при этом могут относиться к одной и той же схеме баз данных. Можно представить, что одна секция дистрибутива связана с одним приложением. Несколько приложений работают с одними и теми же данными в общей базе данных, выполняя разные операции, и соответствующие изменения базы данных соответствуют изменениям самих приложений, которые идут в виде элементов дистрибутива для серверов приложений в той же самой секции.
Компонент установки программного продукта		Группа элементов дистрибутива для приложений и схем баз данных программного продукта. Используется для сокращения описания конфигурации среды программного продукта.

Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Информация о содержании	Способ указать на файлы дистрибутива, входящие в состав данного компонента	<p>К файлам относятся элементы дистрибутива для серверов приложений и файлы для определенного для данного компонента подмножества схем баз данных. Как элементы дистрибутива для серверов приложений, так и схемы баз данных могут входить в несколько компонентов.</p> <p>При этом если несколько компонентов указываются как размещаемые на конкретном сервере среди программного продукта, то это равнозначно указанию размещения на сервере подмножества элементов дистрибутива серверов приложений или схем баз данных, указанных в определении одного или нескольких из данных компонентов. Для изменений баз данных из конкретного дистрибутива это означает, что к данному серверу баз данных последовательно применяются существующие файлы изменений из всех секций дистрибутива, связанные с соответствующими схемами баз данных.</p>

Среды программных продуктов

В данном пункте определяются подчиненные элементы программных продуктов для управления конфигурацией сред программного продукта.

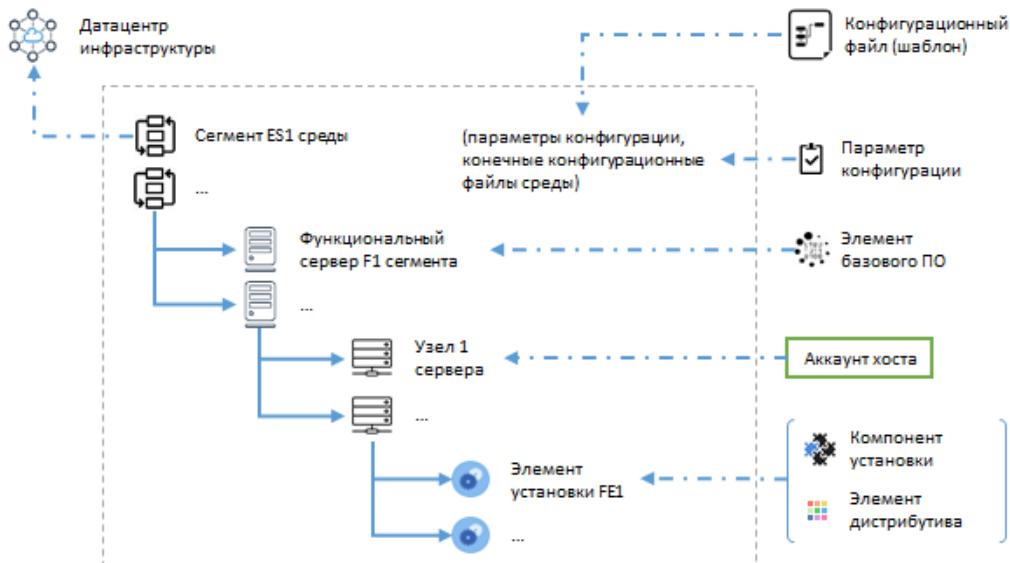


Рис. 301. Среда программного продукта

Тип объекта/атрибут	Семантика	Комментарий
Среда программного продукта		Группа элементов дистрибутива для приложений и схем баз данных программного продукта. Используется для сокращения описания конфигурации среды программного продукта.

Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Логическое название	Название	Логическое название элемента
Тип среды	Классификация	Определяет, относится ли среда к категории сред промышленной эксплуатации, предпродуктивных сред для контрольного тестирования (опытной эксплуатации) перед выпуском в промышленную эксплуатацию или сред разработки. Каждый тип определяет свою политику при выполнении операций со средами.
Референтная среда	Идентификатор среды	Идентификатор другой среды данного программного продукта. В случае когда существует несколько сред с разной конфигурацией, данный атрибут указывает, какая среда является эталонной для данной среды.
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. С неопределенными объектами не могут выполняться никакие операции, кроме операций определения конфигурации.
Сегмент среды программного продукта		Сегмент является типовым механизмом локализации изменений в среде. Сегментация сред может быть связана с функциональным делением, областями ответственности или физическим размещением оборудования.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри среды программного продукта
Логическое название	Название	Логическое название элемента. Определяет границы сегмента.
Датацентр	Идентификатор датацентра	Указание на размещение сегмента в конкретном датацентре. Два сегмента одной среды могут быть в одном датацентре, но один сегмент не может располагаться в нескольких датацентрах.
Референтный сегмент	Идентификатор референтного сегмента	Идентификатор сегмента референтной среды, если она указана, который указывает эталонный сегмент, с которым будет сопоставляться данный сегмент.
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. С неопределенными объектами не могут выполняться никакие операции, кроме операций определения конфигурации.
Функциональный сервер среды программного продукта		Набор процессов операционной системы, в общем случае работающих на нескольких хостах инфраструктуры, выполняющих идентичную функцию и имеющих одинаковую конфигурацию размещения элементов дистрибутива и схем баз данных. По сути представляет из себя кластер в общем случае из нескольких узлов. Два узла кластера могут размещаться на одном или двух хостах.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри сегмента среды программного продукта

Информация о виде процессов	Способ указать тип операционной системы и механизм контейнеризации процессов в операционной системе	Набор атрибутов, который определяет каким способом можно обнаружить процессы, относящиеся к одному узлу данного сервера среди всех процессов операционной системы и как выполнить типовые операции управления процессами - запуск, остановка, проверка состояния, обновление.
Информация о размещении файлов сервера	Расположение в файловой системе	Применимо только для сервера приложений, не для сервера баз данных. Набор атрибутов, который определяет где размещаются все элементы дистрибутива, относящиеся к серверу и средства управления сервером (скрипты). Методология UM обязывает осуществлять ту или иную контейнеризацию и локализацию объектов сервера приложений в операционной системе для повышения прозрачности конфигурации. Размещение может быть определено абсолютными путями или относительно домашнего каталога пользователя, под которым работает процесс.
Информация о составе элементов дистрибутива и их расположении	Перечень элементов дистрибутива для приложений и их расположение в файловой системе	Применимо только для сервера приложений, не для баз данных. Перечень и размещение элементов дистрибутива относительно общих путей размещения файлов сервера. Каждый элемент может иметь собственный относительный путь.
Информация о схемах баз данных	Перечень логических схем баз данных и их привязка к физическим схемам базы данных СУБД.	Применимо только для сервера баз данных, не для сервера приложений. Определяет какие логические схемы баз данных из определенных ранее присутствуют в базе данных данного сервера и в каких физических схемах они размещены, как авторизоваться для выполнения изменений в этих схемах.
Информация об элементах установки	Перечень элементов установки, размещенных на сервере	Определяет для сервера приложений элементы дистрибутива или зарегистрированные компоненты установки, а для сервера баз данных схемы баз данных.
Референтный сервер	Идентификатор референтного сервера	Идентификатор сервера в соответствующем референтном сегменте, если он указан. Обозначает, что данный сервер по функциям, составу устанавливаемых элементов и параметрам конфигурации соответствует референтному серверу.
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. С неопределенными объектами не могут выполняться никакие операции, кроме операций определения конфигурации.
Узел сервера среды программного продукта		Один или несколько процессов операционной системы, связанных с одной инсталляцией конфигурации элементов дистрибутива и схем баз данных, определенных для соответствующего функционального сервера. Является одним элементом кластера.
Номер узла в кластере	Порядковый номер узла	Узлы функционального сервера упорядочены и нумеруются от 1 до общего количества узлов. При удалении или добавлении узла выполняется перенумерация.

Тип узла	Классификация	Определяет роль узла в кластере. Кластер может быть активно-пассивный, где работает только активный узел, а пассивные осуществляют холодное или горячее резервирование (например, так работает сервер СУБД Postgres). Другим видом кластера является активно-активный, где все узлы попеременно, в соответствии с нагрузкой обслуживают внешние запросы. С другой стороны, кластер может иметь централизованное управление (например кластер серверов приложений на основе WebLogic) или являться набором независимых друг от друга узлов, объединяемых в кластер при помощи специальных средств маршрутизации запросов (например, pacemaker или nginx)
Информация о размещении в инфраструктуре	Аккаунт хоста в инфраструктуре	Указание на элемент инфраструктуры, где установлен данный узел функционального сервера.
Статус	Режим операций	Определяет, является ли объект действующим или находится в процессе определения. С неопределенными объектами не могут выполняться никакие операции, кроме операций определения конфигурации.

Элементы инфраструктуры для информационных систем

В данном пункте определяется принципиальная структура элементов конфигурации инфраструктуры, на которой размещены элементы информационных систем. Элементы одной иерархии размещаются на элементах другой иерархии, так, что на одном терминальном элементе инфраструктуры могут располагаться два терминальных элемента разных систем, но один терминальный элемент информационной системы располагается строго на одном терминальном элементе конфигурации инфраструктуры.

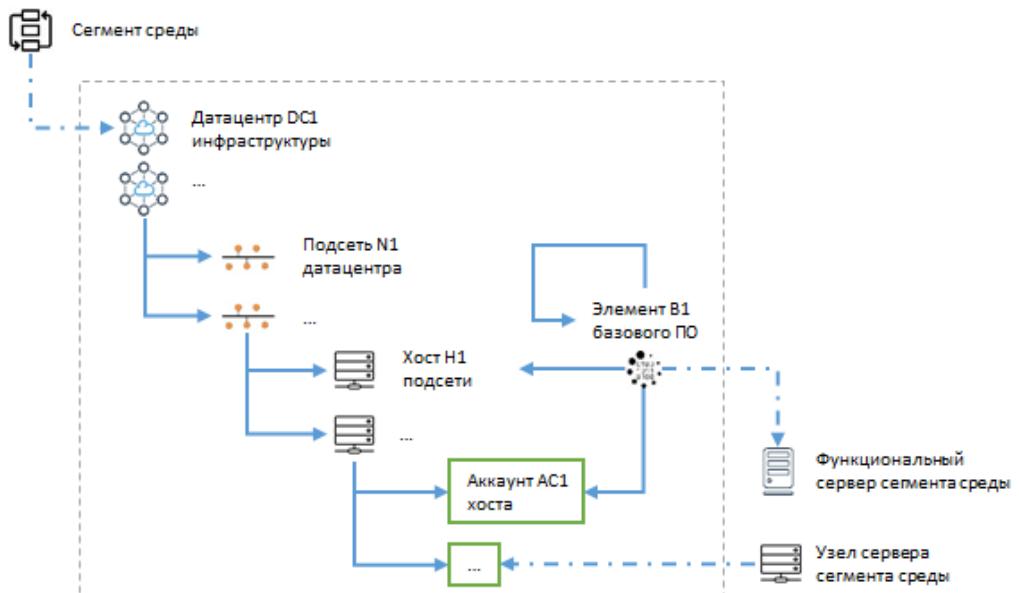


Рис. 302. Элементы инфраструктуры для информационных систем

Тип объекта/атрибут	Семантика	Комментарий
Датацентр		Физически существующий dataцентр или его логический сегмент. Деление на сегменты может быть вызвано областями ответственности или задачами разграничения доступа.
Идентификатор	Способ идентифицировать элемент	Идентификация происходит на верхнем уровне БДКУ, отражает информационные ресурсы соответствующей организационной единицы
Логическое название	Название	Логическое название элемента. Отражает размещение или границы сегмента.
Подсеть		Как правило в рамках dataцентра каждому хосту присваивается IP-адрес. Хосты объединяются в группы при помощи подсетей, которые также являются средством маршрутизации трафика, разграничения доступа через сетевое оборудование и определяют часть IP-адреса. Альтернативой является динамическое присвоение адреса, но как правило это используется только для персональных компьютеров, которые являются источником, а не адресатом соединений. Кроме адреса, у хоста всегда присутствует физическое имя, которое может быть сохранено даже если IP-адрес меняется. Имя прописывается в системе доменных имен и служит идентификатором элемента.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри dataцентра
Логическое название	Название	Логическое название элемента. Отражает назначение подсети
Маска подсети	Сетевая маска	Маска сети в виде X.X.X.X/Y. См. общедоступную информацию по теме.
Хост		Физически отдельный компьютер или виртуальный хост, созданный при помощи системы виртуализации
Идентификатор	hostname	Физическое имя хоста на уровне операционной системы.
IP-адрес	IP-адрес	Адрес хоста в данном dataцентре
Логическое название	Название	Название, отражающее назначение
Операционная система	Информация об операционной системе хоста	Обеспечивает возможность понять, каким образом и при помощи каких команд можно выполнять операции на данном хосте.
Аккаунт		Элемент операционной системы, посредством которого осуществляется разграничение доступа. В том числе суперпользователь, права которого не ограничиваются
Идентификатор	Username	Физическое название аккаунта на хосте, при помощи которого элемент информационной системы появляется на хосте и с чьими правами элемент выполняется.
Признак администратора	Признак	Указание на то, что данный пользователь обладает правами администратора и ему доступны привилегированные операции.
Информация об	Информация об	Группа атрибутов, обеспечивающая доступ к данному аккаунту. Это может быть пароль или закрытый

авторизации	авторизации	ключ.
-------------	-------------	-------

Конфигурационные файлы и параметры конфигурации

В данном пункте определяются объекты, составляющие конфигурационный компонент и указывается каким образом они влияют на содержание.

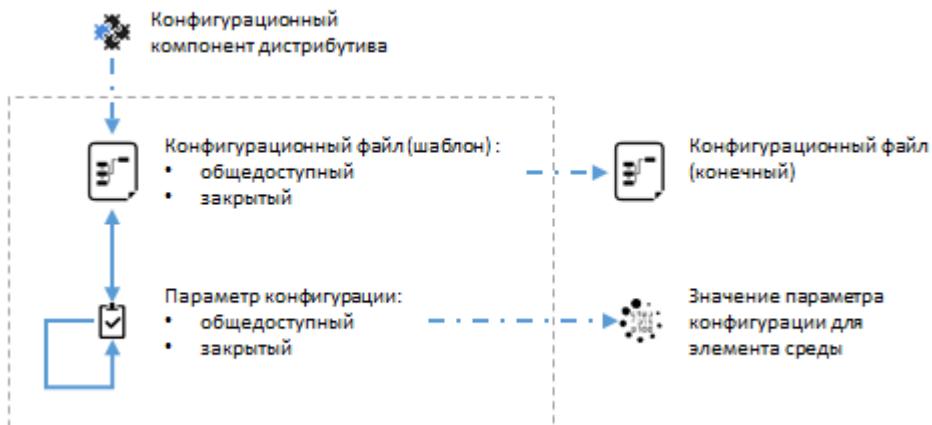


Рис. 3.03. Конфигурационные файлы и параметры конфигурации

Тип объекта/атрибут	Семантика	Комментарий
Конфигурационный файл		Файл в составе конфигурационного компонента, содержащий настройки конфигурации. Настройки конфигурации определяют данные или опции, изменяющие поведение программы или указывающие необходимые детали, которые не изменяются все время пока работает процесс приложения. Изменение настроек означает изменение приложения и соответствует некоторому релизу.
Файл	Текстовый файл	Файл в одном из текстовых форматов, позволяющему редактировать его обычными редакторами, без точной спецификации формата. Примерами таких форматов являются xml, yml, property-файл.
Параметр конфигурации		Настройки конфигурации могут быть недокументированными, внутренними, которые изменяются программистом, или документированными в руководстве по администрированию программного продукта. Документированная настройка конфигурации, занесенная в БДКУ, называется параметром конфигурации. В БДКУ имеет смысл заносить только те документированные настройки конфигурации, которые фактически используются для управления конфигурацией, например, отличны от значений по умолчанию.
Идентификатор	Способ идентифицировать элемент	Идентификация внутри продукта
Уровень конфигурации	Классификация	Параметр может иметь смысл, который привязывает его к определенному уровню конфигурации. Например, если в каждом сегменте среды есть одна база данных, то ее адрес является параметром

		конфигурации уровня сегмента среды - выше для он будет разный для разных сегментов, а ниже - будет тот же самый для всех серверов данного сегмента.
Значение	Данные параметра конфигурации	<p>Для одного или нескольких элементов своего уровня этот параметр может быть задан и иметь свое отдельное значение. Например, если среда состоит из двух сегментов с базами данных и одним сегментом для маршрутизации, где базы данных нет, то значение параметра адреса базы данных будет указано для двух из трех сегментов.</p> <p>Также значение параметра может быть указано для элементов более высокого уровня. В этом случае оно может иметь смысл как значение по умолчанию. Например, уровень детализации журнала уровня сервера может быть определен для среды разработки как TRACE, для среды промышленной эксплуатации как INFO, но для одного из функциональных серверов этой среды указан как DEBUG.</p>

Заключение

ЧТО ДАЛЬШЕ

Подведем итоги содержимому данной книги. О чем она рассказала и как применить ту информацию, что в ней описана.

Что дальше

Книга о конфигурации программных продуктов вводит основные интегральные понятия - программный продукт, среда программного продукта, релиз программного продукта и отслеживает все основные события "последней мили" в сфере информационных технологий от разработки релиза до постановки до мониторинга. В контексте этой цепочки описаны все основные операции, понятия и термины, зависимости и процессы. Именно разбиение на кубики позволяет в конкретном месте применения методов данной книги использовать эти кубики как конструктор, создав свой конечный вариант релизного процесса с учетом своих организационных и технологических особенностей, а также текущих целей данного процесса.

Созданную технологическую цепочку можно будет впоследствии улучшать, укреплять и дополнять, но самым трудным и главным шагом будет интеграция разрозненных операций и создание прозрачного и автоматизированного сквозного процесса. Только на интегрированной системе можно будет существенно повысить эффективность работ и добиться скорости прохождения по цепочке без снижения качества результата.

Сборку этого процесса можно делать так же как вы собираете сложный паззл - сначала выложить все элементы перед собой, рассортировать по областям, выбрать место конечной картины, с которого вы начнете собирать паззл - "точку кристаллизации процесса", и постепенно присоединять к нему все больше и больше существующих операций, до тех пор, пока не увидите полную картину от начала операций до конца цепочки. Собирание "паззла" означает не просто выписывание информации в некоторую записную книжку, а создание системы автоматизации, подключенной к единой CMDB, и выполнение с ее помощью включенных в паззл операций уже по выбранным правилам.

Количество стандартов и методологий, созданных на текущий момент не оставляет ни одного шанса, что термины данной книги совпадают с терминами существующих стандартов и методологий. Однако большая их часть также создана на основе опыта работы с реальными системами и по сути противоречий не должно быть. Надеюсь, что данная книга стала наиболее последовательным и четко структурированным источником информации о том, как организовать выпуск и обновление информационных систем, если их размер превысил некоторый предел, до которого можно оперативно найти удобное решение, без привлечения "тяжелой артиллерии" методов программной инженерии.

Содержание

Оглавление.....	3
Введение	5
Проблема последней мили	7
Релизные процессы.....	7
Парадигмы релизных процессов	9
Учетная система как инструмент.....	9
Единая метамодель.....	12
Идеология, методология и технология	14
Программные продукты	17
Понятие продукта	19
Поставка программных продуктов	21
Технологический стек.....	28
Информационная система как набор прикладных продуктов.....	29
Управление изменениями	32
Жизненный цикл продукта	33
Продукт и проект	33
Портфель и линия продуктов	35
Управление конфигурациями.....	36
Операции.....	39
Программный код	41
Исходный код и исходные материалы	43
Ресурс контроля версий и репозитории	43

Репозиторий исходного кода	44
Ветки и теги репозитория исходного кода	45
Компилируемый и некомпилируемый код.....	48
Код и библиотеки из сторонних продуктов.....	49
Код баз данных	50
Конфигурируемый программный код	52
Сборка программного кода	53
Что такое сборка и какова причина сборки.....	55
Исходная информация для сборки	57
Режимы сборки	59
Учет зависимостей	61
Средства сборки	63
Фреймворк сборки.....	64
Непрерывная интеграция.....	65
Выпуск релизов	67
Изменения	69
Потоки изменений	71
Потоки релизов	73
Виды релизов и сериализация	75
Дистрибутивы релизов	77
Способы дистрибуции релизов	79
Модели фиксированного объема и фиксированного окна	82
Релизный процесс и процесс управления задачами	84
Релизный процесс и релизный цикл.....	86
Фазовый контроль	88
Жизненный цикл релиза	90
Управление составом релиза.....	93

Отмена релиза	96
Непрерывная поставка.....	96
Создание и поддержка сред	99
Экземпляр продукта.....	101
Среда продукта	101
Конфигурация среды.....	103
Датацентры и облачная инфраструктура	104
Облака и сетевые ресурсы.....	106
Тип среды	107
Сегментация среды	110
Хосты и аккаунты	110
Функциональный сервер	114
Узлы сервера.....	117
Размещение элементов	119
Операции в среде.....	122
Тестирование релизов	123
Создание сред для тестирования.....	125
Закрытые данные и данные для тестирования	126
Установка изменений.....	129
Предмет и пакет изменений.....	131
Изменения в инфраструктуре.....	134
Изменения в операционной системе	136
Изменения в базовом ПО	137
Изменения в прикладном ПО.....	139
Процедуры запуска и остановки процессов	141
Время недоступности сервиса.....	142
Холодное и горячее обновление	144

Доведение обновления до конца	145
Обеспечение возможности отката изменений	147
Установка инкрементальных релизов	150
Изменение конфигурационных файлов в приложениях.....	153
Обновление закрытых данных в приложениях.....	155
Операции изменения баз данных	157
Изменение схем и экземпляров баз данных.....	159
Скрипты отката изменений баз данных.....	161
Изменение параметров конфигурации в базах данных.....	162
Мониторинг	163
Причина и предмет мониторинга.....	165
Время работы прикладного продукта.....	166
Активный мониторинг услуг.....	167
Ресурсный мониторинг.....	168
Мониторинг прикладных процессов.....	169
Мониторинг операций.....	170
Обработка событий мониторинга	171
Обновление системы мониторинга	172
Мониторинг и данные управления конфигурацией	174
Документация	175
Потребность в документации	177
Категории документации	178
Документация продукта	179
Документация релиза.....	184
Обновление документации	186
Форматы документации	187
Релизные процессы	191

	233
Потребность в процессах	193
Модель ITIL для релизных процессов.....	195
Релизы и изменения	199
Роли в релизных процессах	199
Структура и связи процессов	201
Инициация релиза.....	202
Управление составом релиза	203
Разработка задач релиза	203
Интеграция программного кода	204
Сборка дистрибутива	205
Управление средами.....	206
Тестирование релиза	207
Финализация релиза	208
Подготовка к установке.....	209
Изменение среды промышленной эксплуатации	209
Закрытие релиза	210
База данных конфигурационного управления.....	211
Информационные системы и программные продукты	213
Элементы в составе программных продуктов	214
Среды программных продуктов.....	218
Элементы инфраструктуры для информационных систем.....	221
Конфигурационные файлы и параметры конфигурации	223
Заключение	225
Что дальше	227
Содержание	229