Infrastructure

Как проводить переезды, не привлекая внимания пользователей

Виктор Азаубаев

Разработчик системы сборки
в Yandex Platform Engeneering

Сегодня расскажу

Как управлять большими миграциями и переездами:

1

минимально ломая пользовательские сценарии

2

не замедляя разработку сильнее чем необходимо

3

в условиях ограничения ресурсов

4

предвосхищая возможные проблемы

Мой опыт

1

Организовал и активно участвовал в миграции 180_000 строк кода с Python 2 на Python 2/3, затем (почти) на Python 3.

2

Внедрял новые фичи в систему сборки с DAU > 10_000 и 1_000_000 запусков в день

3

Собрал и объединил опыт переездов коллег в условиях монорепозитория

Проблематика

- Миграция — процесс замены подсистем (реализацию) на новые с отказом от старых
- Чем больше система, тем сложнее процесс
- При неправильном подходе миграции могут принести больше проблем, чем пользы

Цели

Что мы хотим:

Заменить старую реализацию на новую:

- функция, библиотека, api, фреймворк, подсистема, язык

Что хочет пользователь:

- Продукт должен продолжать решать задачу
- Ни единого разрыва!
- Незаметный переход*
- Минимальные затраты на переход

Что хочет бизнес:

Нужны:

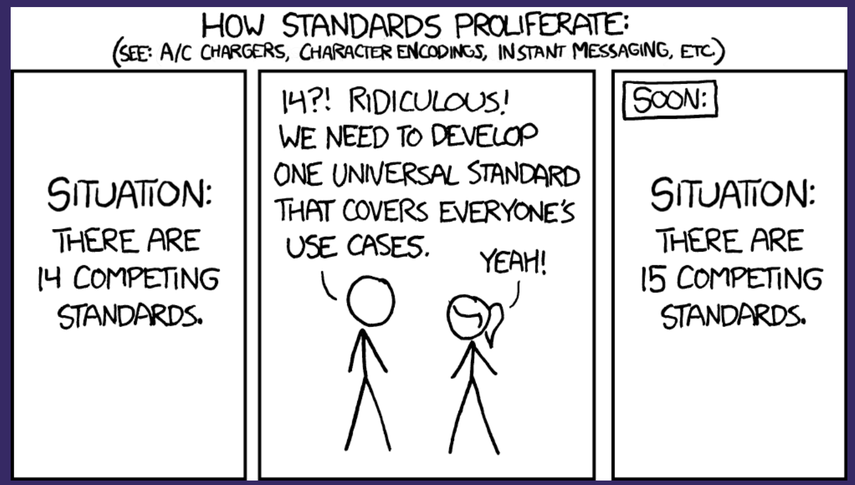
- Фичи, фичи, больше фич!
- Высокая скорость доставки
- Доступность / надёжность

Примеры

Переход с Python 2 на Python 3

Переход на другую базу данных

Унификация существующих решений



Обновление версии библиотеки

Добавление новой версии в API

Как мы хотим

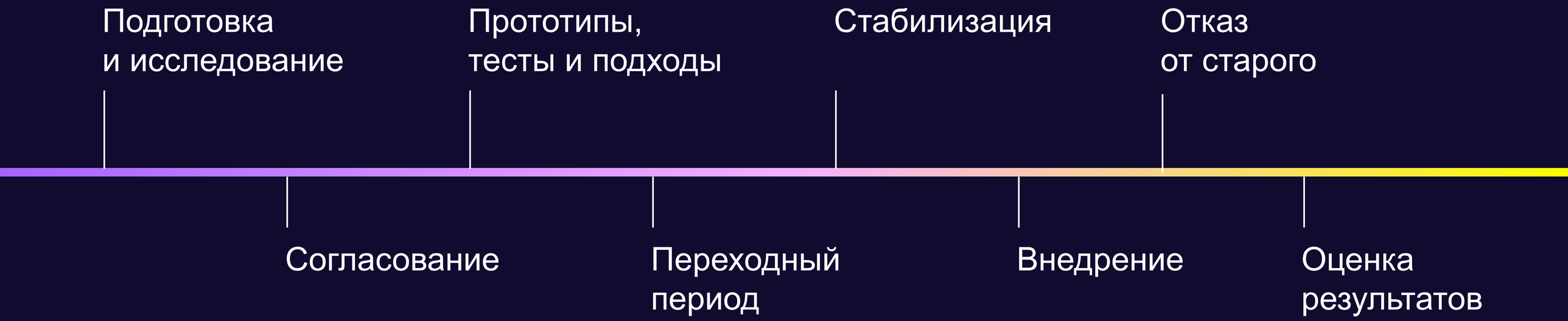
Ожидание:

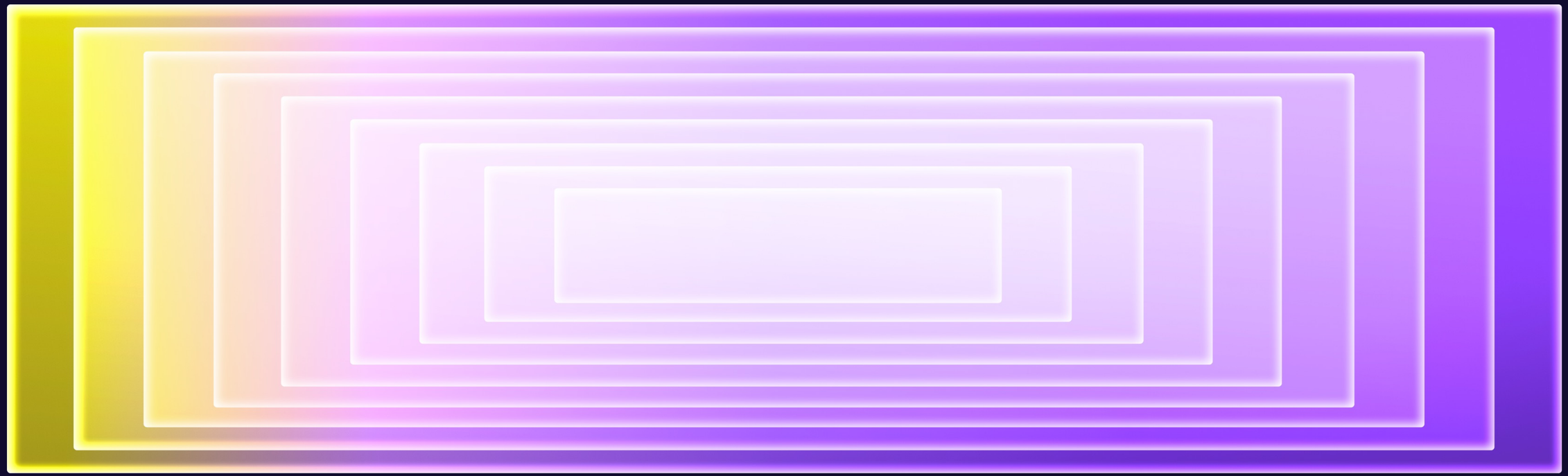
- Переписать, переключить, выбросить старое и идти в светлое будущее

Реальность:

- Сложно обосновать необходимость
- Много неопределённости, чего-то не учли, не успеваем в сроки
- Неприятный баг замедлил переезд

Обзорный план





Подготовка
и исследование

Ответить на вопрос «нужно ли?»

Когда не нужно:

- Проект на поддержке,
  не развивается или будет переписан
- Небольшой impact
- Проект большой, рук мало
- Разработчики не хотят
- Впереди большие изменения
- Всё просто работает, есть
  пить не просит

Ответить на вопрос «нужно ли?»

Когда не нужно:

- Проект на поддержке,
  не развивается или будет переписан
- Небольшой impact
- Проект большой, рук мало
- Разработчики не хотят
- Впереди большие изменения
- Всё просто работает, есть
  пить не просит

Когда нужно:

- Проект активно развивается
- Миграция решит важную
  проблему
- Объективные причины
- Команда готова

Ответить на вопрос «зачем?»

- Что-то станет лучше
  - ?
  - ?
  - ?

- Какие появятся проблемы
  - ?
  - ?
  - ?

при

- Отказе от старого
  - ?
  - ?
  - ?

- Переходе к новому
  - ?
  - ?
  - ?

скорость разработки

потребление ресурсов

производительность

найм

безопасность

Определиться с форматом

Постепенный
(старый и новый вместе)

- проект большой и важный, активно развивается
- есть возможность переключать по частям
- есть возможность обеспечить одновременную работу

Моментальный
(старый и новый отдельно)

- проект небольшой, не сильно быстро развивается, много тестов, легко переключиться
- нет возможности либо слишком дорого делать постепенно

Оценка

Попробовать мигрировать небольшую часть проекта

(если есть возможность)

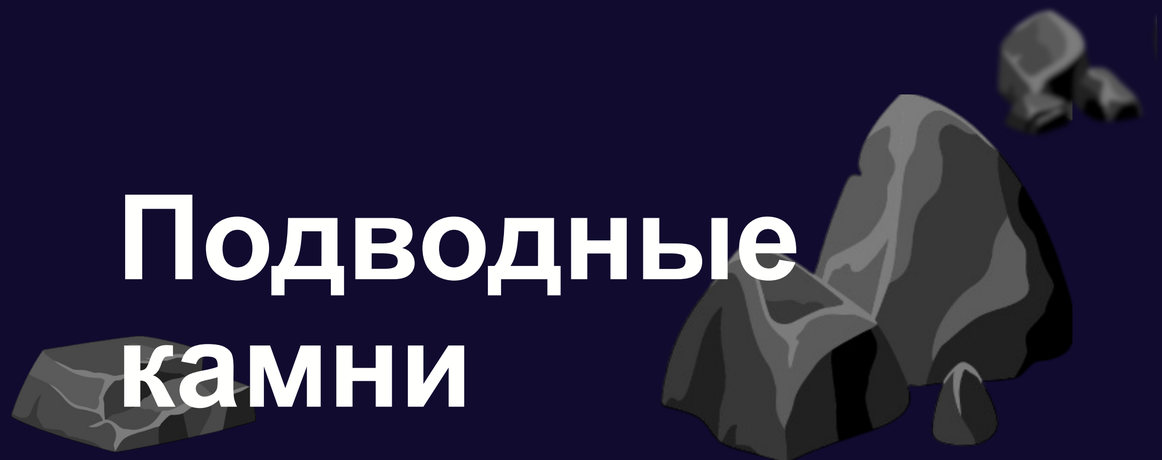
Составить план для оценки масштаба работ

(понять функционал заменяемой части)

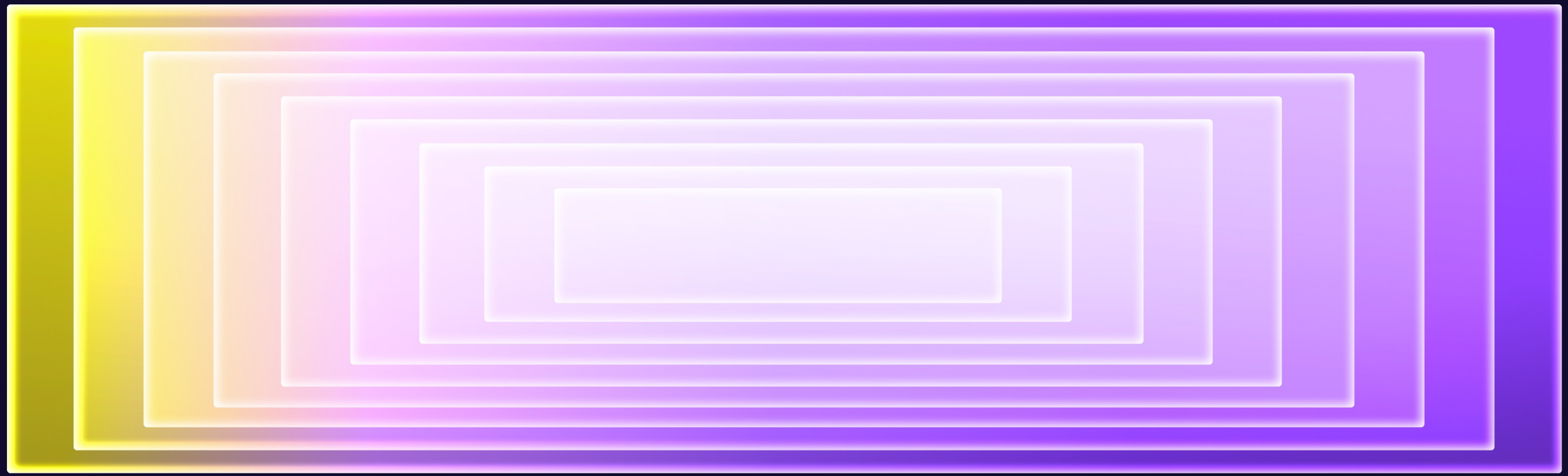
- ⊕ Пользовательский путь 1
- ⊕ Часть приложения X
- ⊕ Библиотека Y
- ⊕ Эксперименты

- ⊖ Можем замедлиться
- ⊖ Придётся переучиваться
- ⊖ Будет тяжело в переходный период
- ⊖ Есть новая мажорная предрелизная версия с большим количеством изменений

Подводные камни

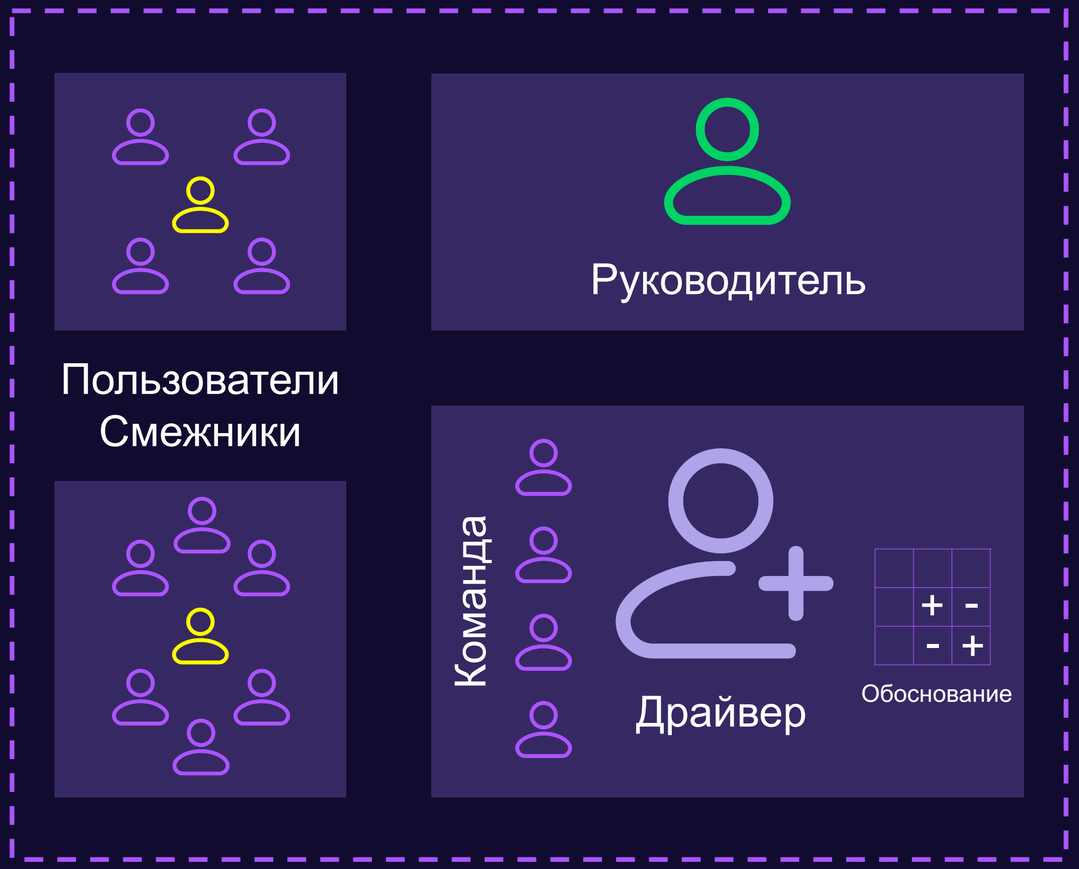


- Наличие тестов не даёт 100% гарантий
- Рутина и выгорание
- Что-то забыли при переезде
- Увеличение когнитивной нагрузки
- Миграция потребляет ресурсы
- Миграция ВСЕГДА требует больше ресурсов чем ожидается

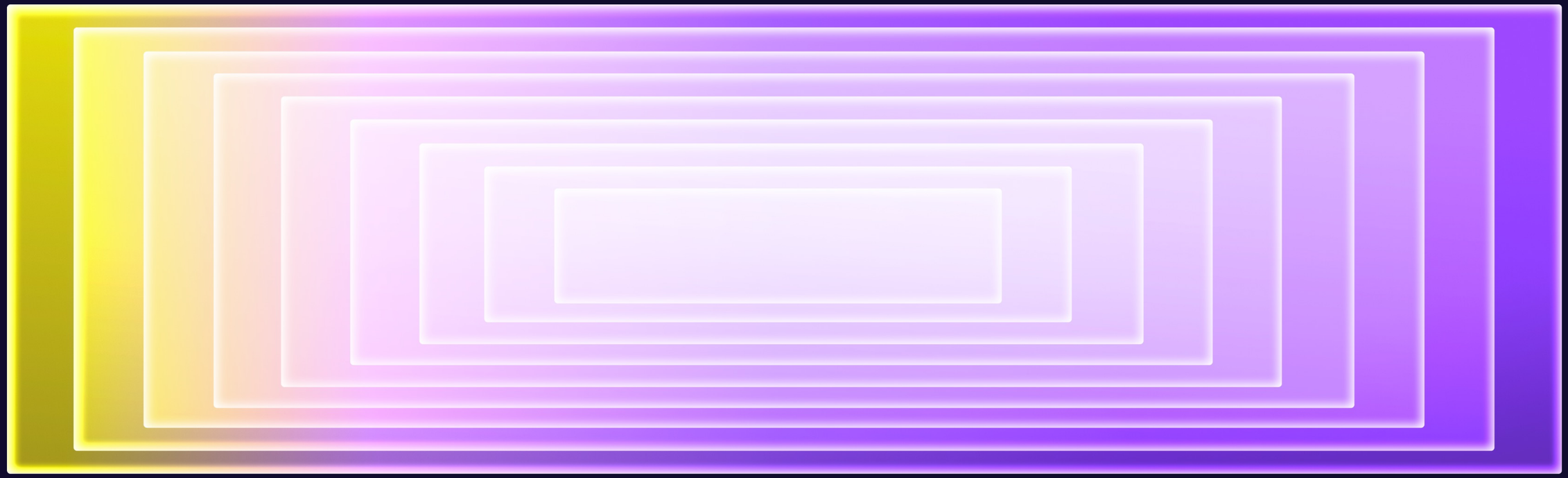


Согласование

Выделить ресурсы



- Найти драйвера изменений
- Собрать план
- Заручиться поддержкой команды
- Заручиться поддержкой руководителей/бизнеса/пользователей/смежников
- Быть честным и прозрачным

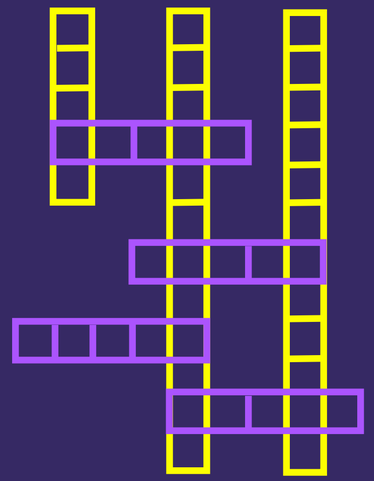


Прототипы, тесты, подходы

Выделение use-cases и функциональных частей

Части:

- Пользовательский путь
- Компонент

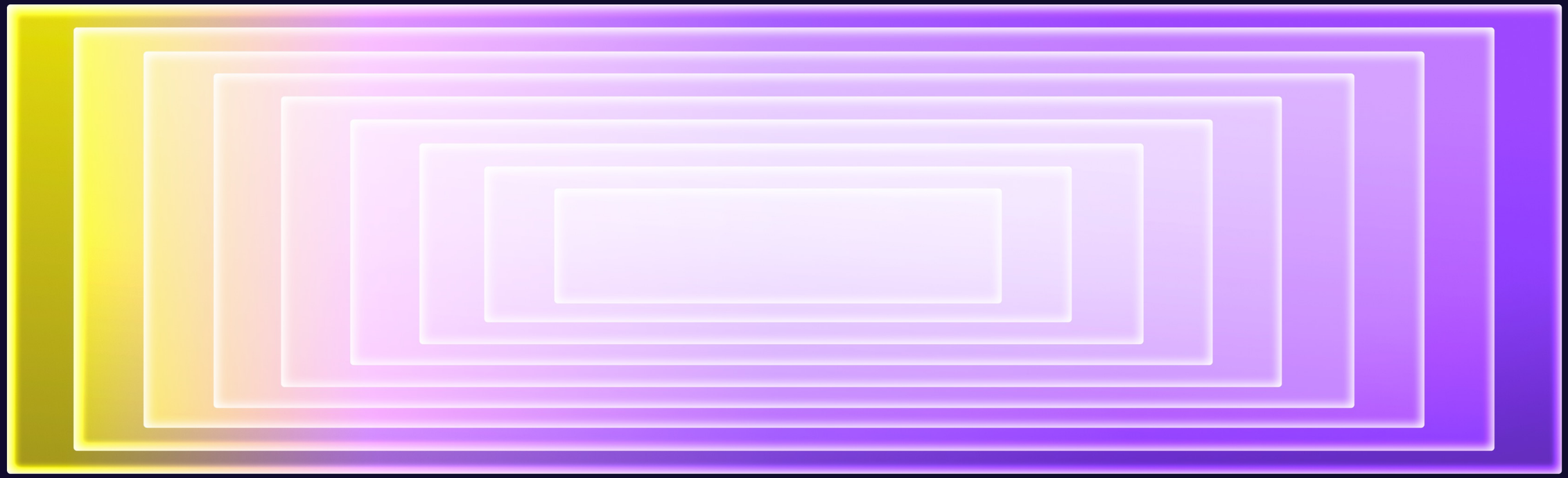


Условно-независимые части:

- Сложные и частые помогут пройти путь миграции, осознать основные проблемы и начать внедрять лояльным пользователям / окружениям
- Простые и редко используемые помогут отработать механику переключения

Тесты


Тесты помогают
не ломать пользовательские сценарии
не допускать регрессий
можно считать покрытие
дописывать тесты на базе поломок



Переходный период

Переходный период — инструменты и подходы

Инфраструктура для одновременной работы / тестов:

1

- Совместимое подмножество

2

- testing/preprod окружения

3

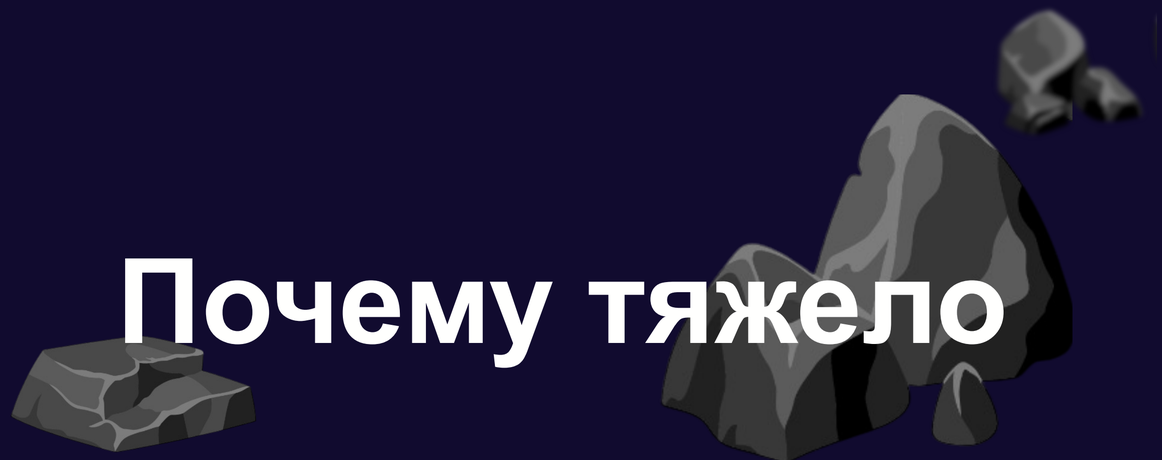
- Экспериментальные ветки

4

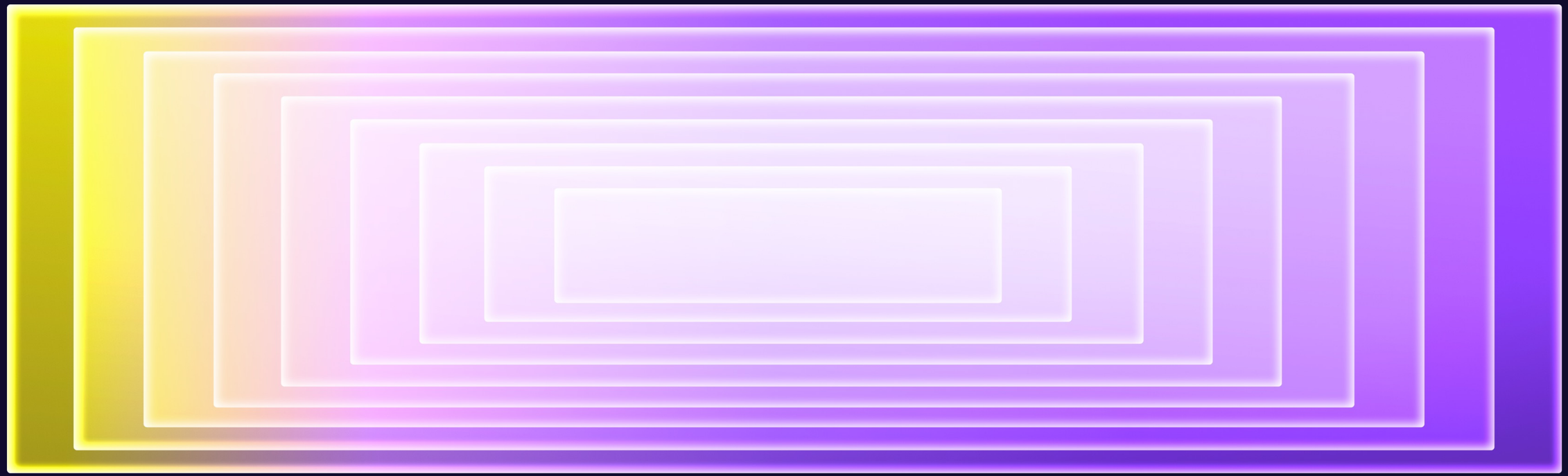
Proxy

- Выбор поведения исходя из версии (if)
- Откат на старое поведение в случае проблем (try)

Почему тяжело



- Нужно помнить о двух реализациях сразу
- В случае экспериментальных веток есть риски конфликтов
- Внешние потребители могут страдать и заставлять страдать вас



Стабилизация

Правило Парето 20-80

20% работы

дают

80% результата

Что и где искать

Небольшая часть приложения, которая делает значительную часть работы

Задача выделить:

1

Воспроизводимый артефакт

2

Слабая связь через интерфейс

3

Нет комбинаторного взрыва (проклятья размерности):

- Мало входных данных
- Небольшое количество возможных значений
- Небольшое количество вариантов использования

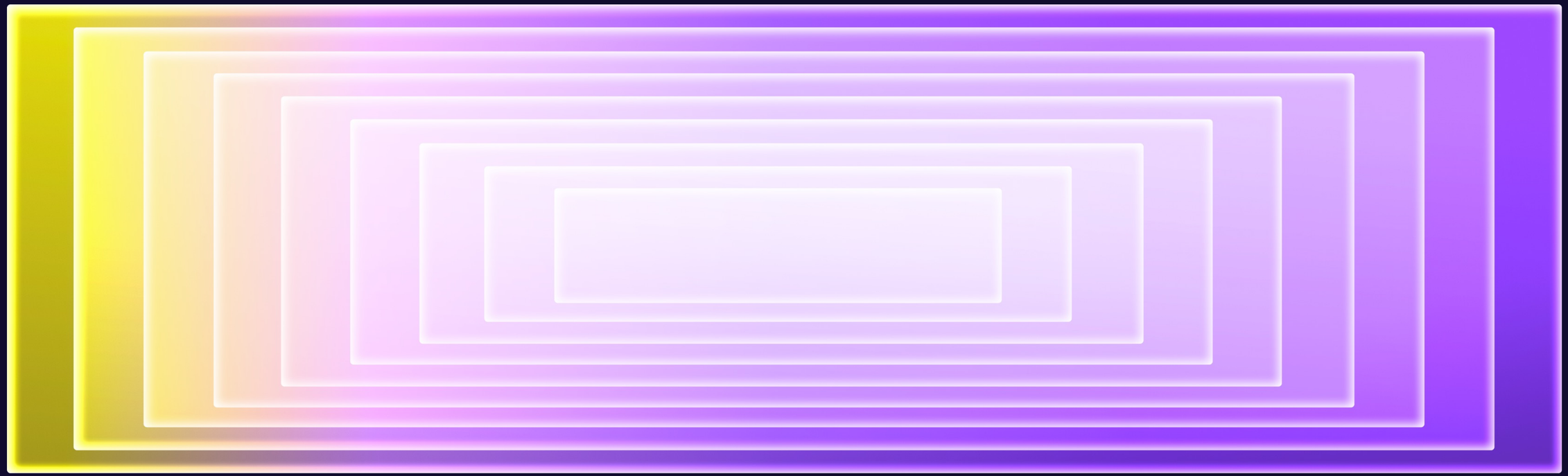
4

Потребляемые ресурсы (время, cpu, память)

Как пользоваться

- Воспроизводимый артефакт
- Поверх самых частых пользовательских сценариев
- Поверх части приложения с небольшой вариативностью
- Поверх части приложения с небольшим потреблением ресурсов
- Тестируем сами, остальное тестируем на пользователях
- 20% приложения на 20% частых кейсов при 20% потребления ресурсов:

  0,8% работы при 51% результата

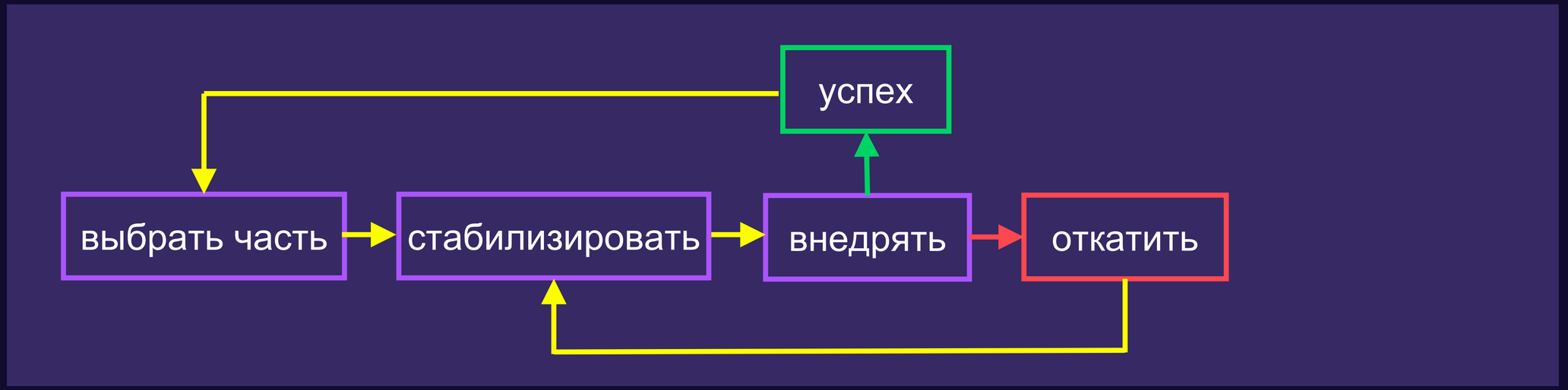


Внедрение

Внедрение

1. Пользователи ничего не должны заметить
2. Постепенное включение по частям
3. Тестирование на лояльных пользователях
4. Тестирование на prestable
5. Постепенное включение на пользователей
6. Переключать не в прайм-тайм, не ночью, не в пятницу
7. Переключение без простоя
8. Возможность быстрого отката
9. Мониторинги и обратная связь

Много рутины



- хахатоны
- делегирование
- популяризация инструментов
- автоматизация
- обучение команды

Автоматизация рутинных действий

Меньше вероятность допустить ошибку

Не нужно тратить рабочую неделю на автоматизацию того, что можно сделать за рабочий день руками

1 день

можно потратить на то, что сэкономит **месяц** работы другим людям

Можно потратить час на автоматизацию того, что может пригодиться людям (или составить инструкцию)

Можно автоматизировать часть, допишут те, кому надо

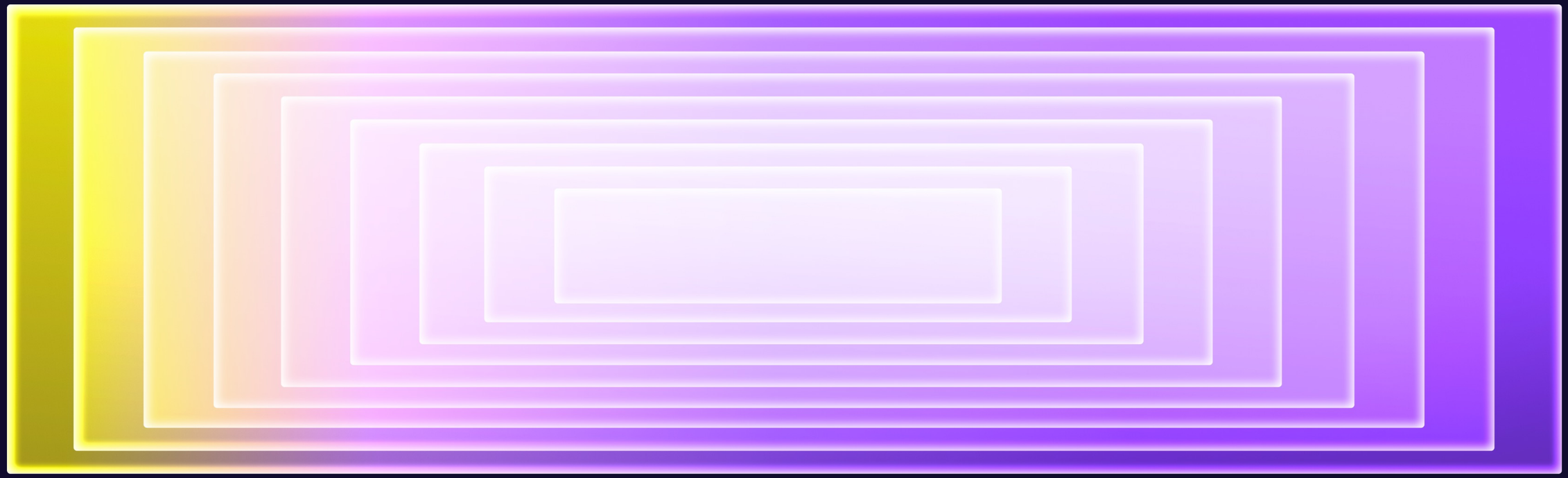
Готовимся к тому, что:

- Проблемы обнаружатся уже во время переезда
- Обязательно потребуется больше времени и ресурсов
- Чем больше и сложнее проект — тем более масштабные проблемы возникнут

Сложности при внедрении новых частей

Как избежать

- заранее исследовать интерфейсы и способы взаимодействия
- сделать тестовое внедрение и поэксперементировать на реальных примерах

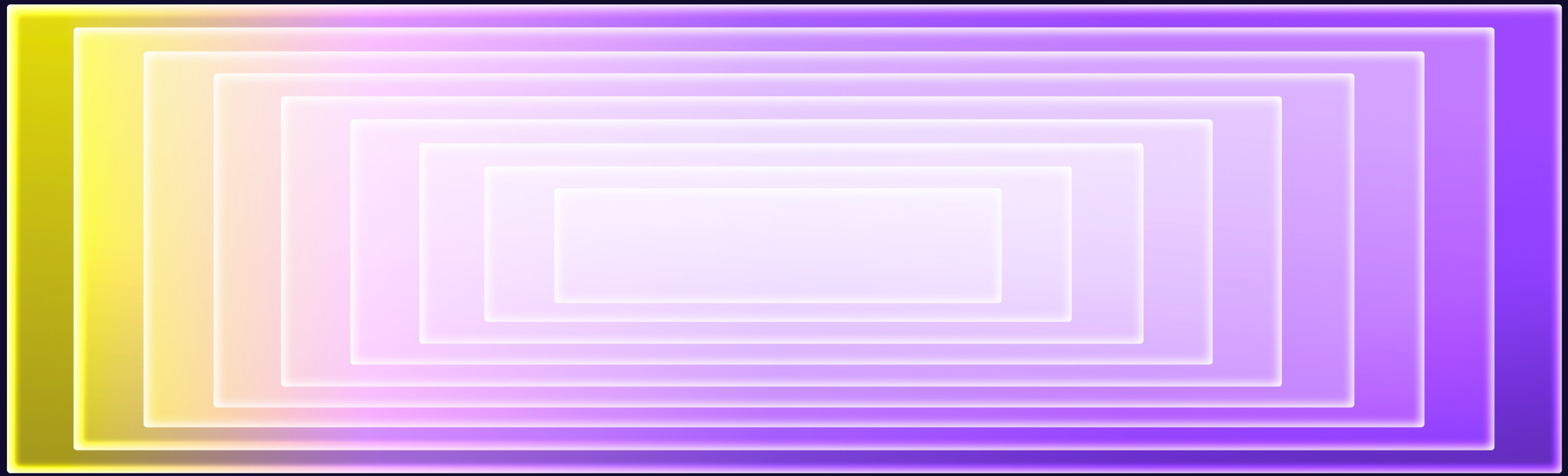


Отказ от старого

Отказ от старых частей

Причины

- не реализовали важный функционал
- нет ресурсов мигрировать смежникам
- внезапная ошибка в неожиданном месте
- недостаточная глубина проработки



Оценка результата

Оценка результата

1

Достигли
ли того,
что хотели?

2

Стало
ли быстрее/
удобней/
легче?

3

Рассказать
и вдохновить

4

Наградить
причастных

Выводы

- Мигрировать нужно не всегда
- Если нужно — то стараться делать это оптимально и умеренно быстро
- Миграция связана с увеличением сложности, потребления ресурсов, когнитивной нагрузки
- Идти в разных направлениях
- Не забывать про командную работу, процессы, автоматику
- Делиться опытом
- Продумывать все риски и неопределённости как можно раньше → Акцентировать внимание на рисках и неопределённостях
- Помнить, что всё пойдёт не по позитивному сценарию
- Заранее заложить больше ресурсов

Спасибо
за внимание

Infrastructure

Вопросы и наброс

Виктор Азаубаев
Разработчик системы сборки
в Yandex Platform Engeneering