

hl⁺⁺

HighLoad⁺⁺

**О чем стоит подумать,
приступая к разработке
высоконагруженной
СИСТЕМЫ.**

Артем Вольфтруб

www.gramant.ru



hl⁺⁺

HighLoad⁺⁺

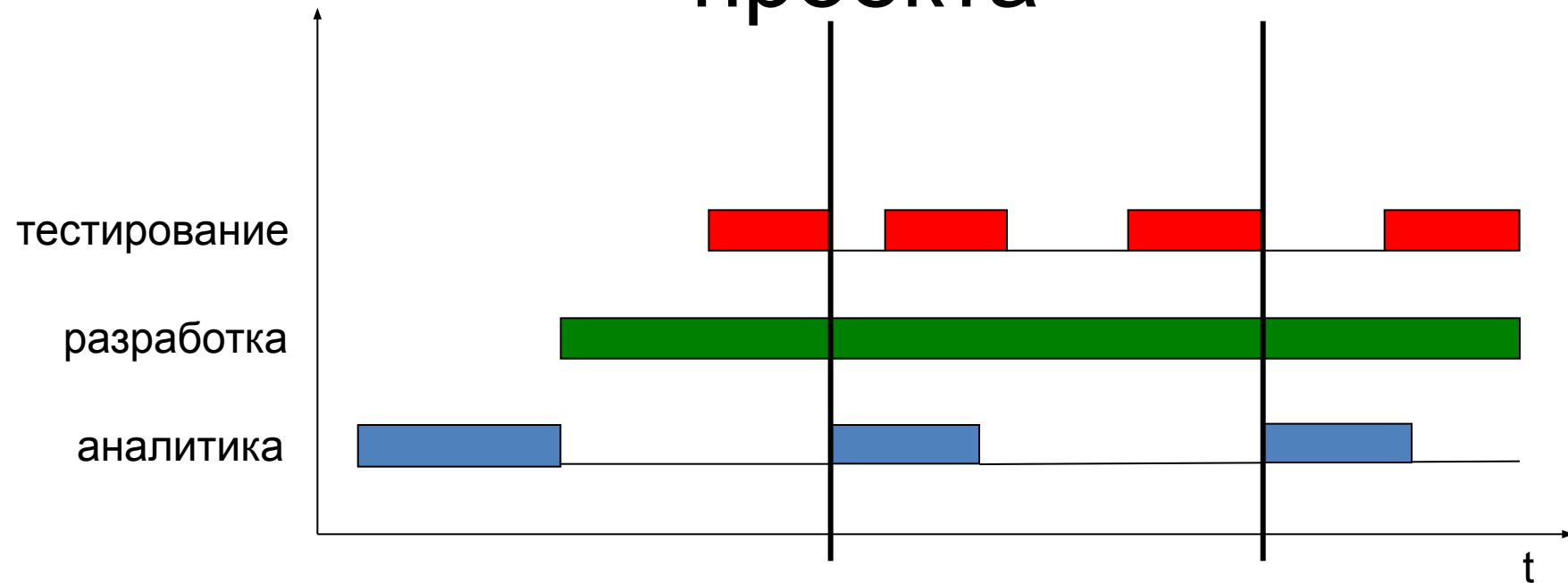
1

У нас есть своя IT команда, но она сильно загружена в ближайшие три месяца.

Мы рассчитываем, что за это время вы напишите первую версию системы, которую мы будем развивать своими силами.



Цикл разработки интернет-проекта



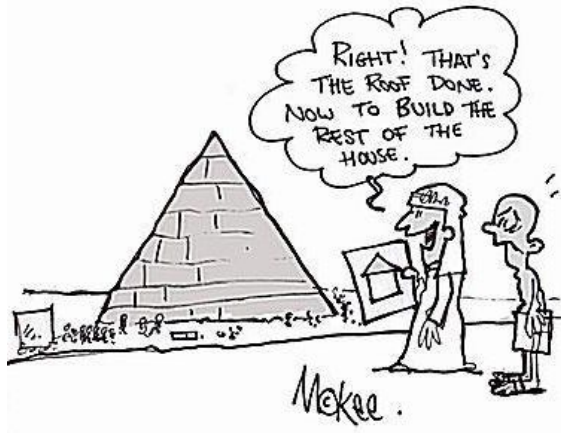
Важно понимать, что

- Три месяца – минимальный цикл разработки интернет системы
- К моменту релиза требования поменяются процентов на 40
- Если N разработчиков сделают систему за три месяца, то $2 * N$ разработчиков сделают систему за... **три месяца**
- Основная работа начинается после релиза



Передача проекта другой команде

- Передавать код другой команде сразу после релиза – **плохая идея**
- Передавать код в виде дампа svn - **очень плохая идея**
- Личное VS профессиональное



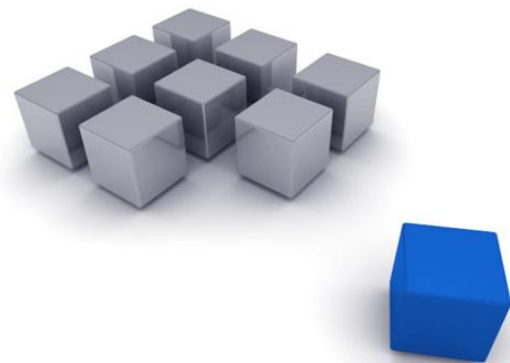
Чтобы не было мучительно больно

- Решение о передаче проекта не должно быть спонтанным
- Решение должно быть известно заранее
- Привлекайте разработчиков к процессу
- Приготовьтесь заплатить дважды



Способы облегчить процесс

- Совместная работа над проектом
- Постепенный ввод новой команды

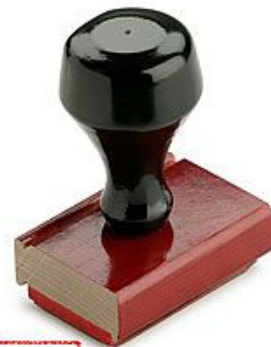


*В первую версию системы должно войти N фич.
У нас есть еще несколько минорных пожеланий, но их
можно будет реализовать после выпуска первой
версии.*



Формирование требований

- Анализ рынка
- Формирование ключевых пользовательских групп
- Формирование стратегии
- Интервьюирование ключевых пользователей
- Прототипирование
- Тестирование, получение обратной связи
- Коррекция



NO

ТАК НЕ БЫВАЕТ

Формирование требований

- Наличие аналогичного продукта или сервиса
- Видение системы, изложенное на листе А4
- Идея в голове начальника



YES

ТАК БЫВАЕТ

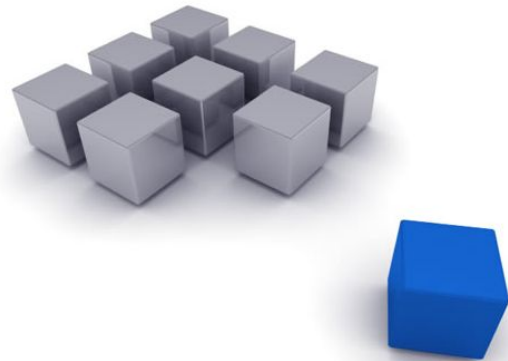
Из опыта

- На момент релиза, востребованными оказываются около 60% фич
- 40% фич, которые оказались не востребованными – самые сложные с точки зрения реализации
- Наиболее ценные фичи не попадают в первую версию

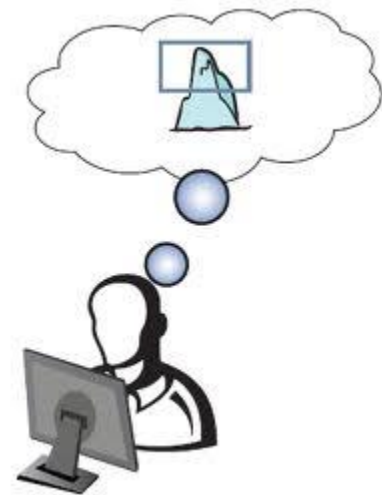


Рамки проекта

- Старайтесь включать в первую версию только то, без чего реально нельзя жить. Экономьте время!
- Основной источник требований – пользователь
- Бета-версия – главный инструмент аналитика
- Бета-версия – полностью функциональный продукт, а не «отмазка» для разработчиков
- Разработка не заканчивается релизом



Система должна быть масштабируемой. Нам нужен подробный план того, как мы будем справляться с нагрузками, когда система вырастет со 100 000 пользователей до 10 000 000.

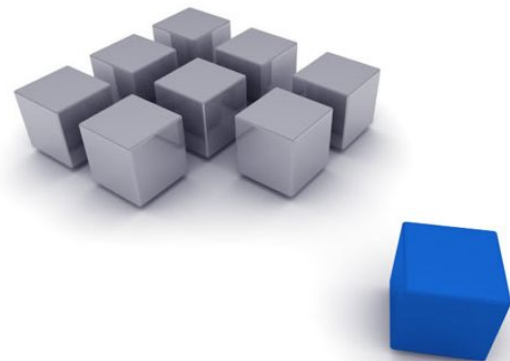


Цели планирования

- План для начальства или план для разработчиков
- Узкие места возникают совершенно не там, где это предполагалось
- А кто будет писать?

Анализ нагрузки

- Оцениваем трафик
- Оцениваем объем данных
- Фантазируем («если – то»)



Слайд не для менеджеров!

- У «Веселого фермера» тоже был первый пользователь
- Когда у вас будет 10 000 000 пользователей, у вас будут деньги, чтобы все переписать



Производительность системы будет проверяться проведением полноценного нагрузочного тестирования перед сдачей проекта.

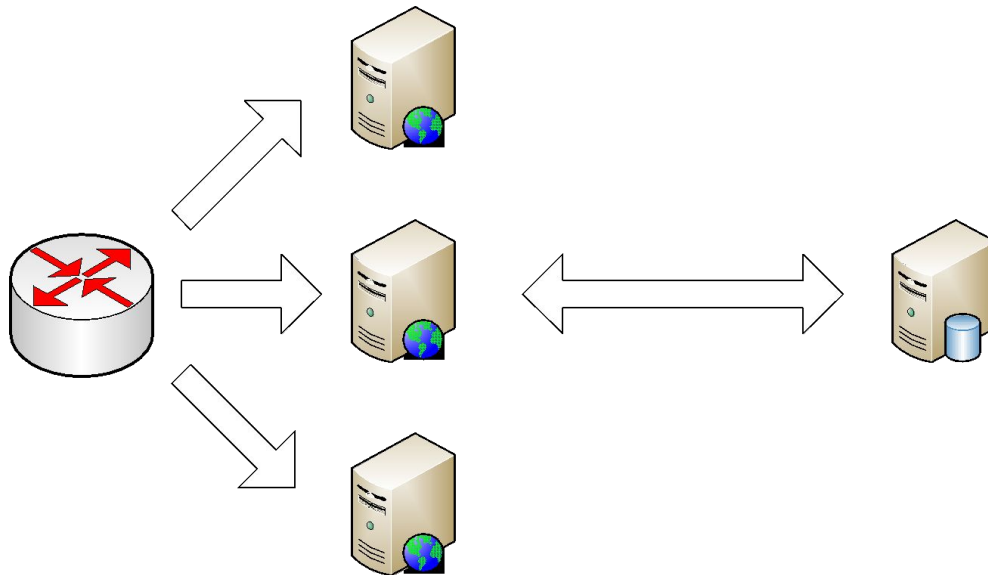


Проблемы нагрузочного тестирования

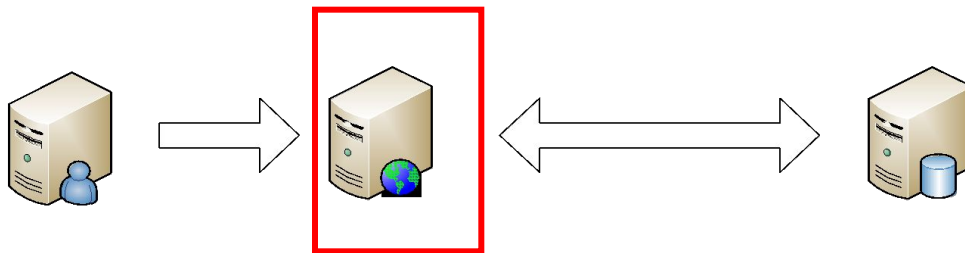
- Смоделировать реальные действия пользователя очень сложно
- Влияние внешних компонентов
- Тепличные условия тестирования
- Заказчик считает, что нагрузочное тестирование позволит оценить качество системы



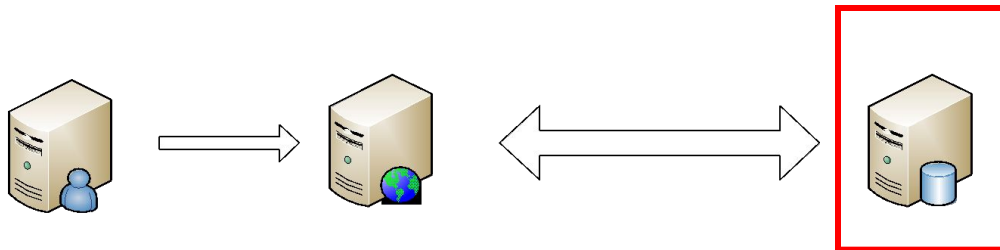
Хроники нагрузочного тестирования



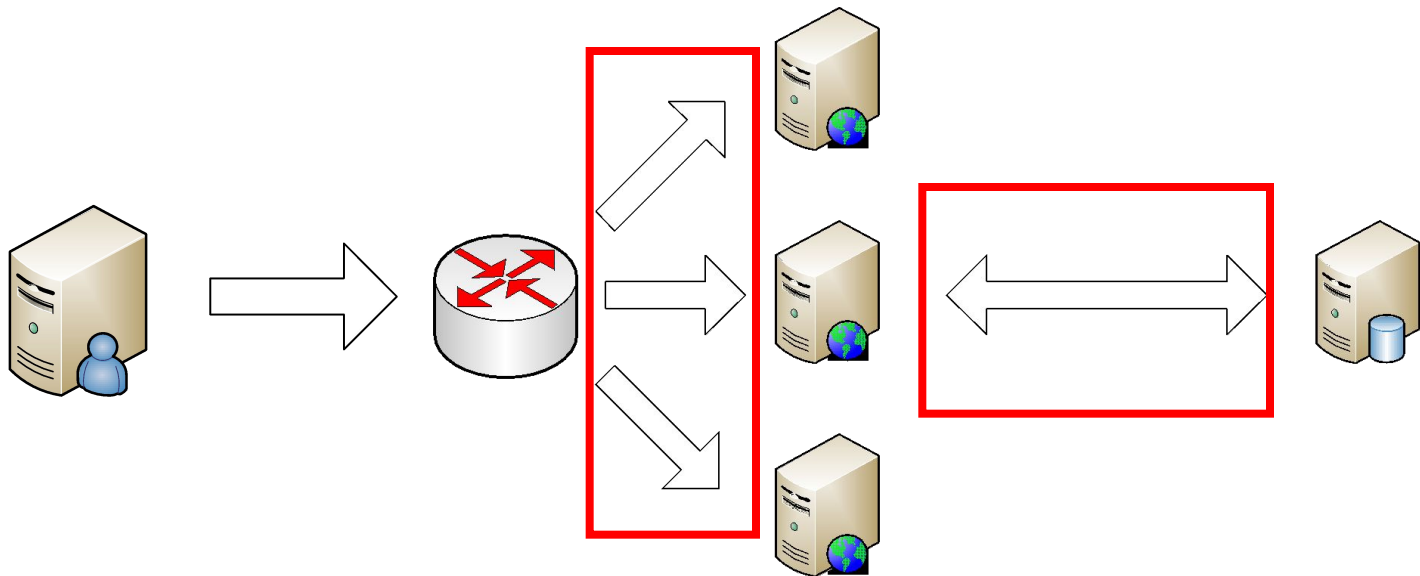
Хроники нагрузочного тестирования



Хроники нагрузочного тестирования



Хроники нагрузочного тестирования

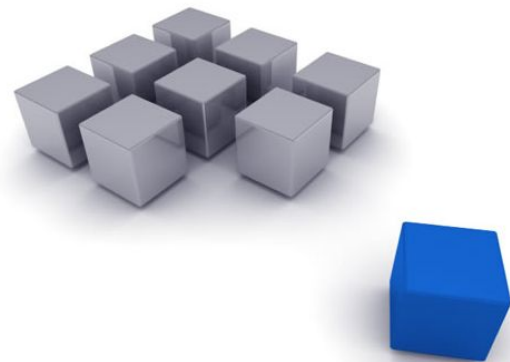


Обобщаем

- Другое железо
- Другой объем данных
- Другой канал
- Влияние окружения на работу приложения
- Интерполяция не работает

Выводы

- Нагрузочное тестирование инструмент анализа, а не критерий приемки
- Проверять лучше отдельные сценарии, а не полноценную работу приложения



*Что значит приемлемый уровень отказоустойчивости?
Система должна работать безотказно!*



Виды простоев

- Отказ в результате выхода из строя
- Остановка на плановое обслуживание



Оценка отказоустойчивости

- Внешние зависимости
- Прагматичный подход (99.99% - это 1 час в год)
- Выделение критических компонентов

Кому нужна отказоустойчивость

- Компоненты, которые используются внешними системами
- Компоненты, от которых зависит бизнес компании
- Компоненты, простой которых связан с репутационными потерями компании



Зачем нам система мониторинга? Если система сломается, это и так все увидят!



Проблемы

- Мониторинг не является частью проекта
- Систему мониторинга должен кто-то эксплуатировать



Запуск высоконагруженной системы без мониторинга не имеет смысла!

Что дает мониторинг

- Прогнозирование нагрузки
- Диагностика проблем на ранней стадии
- Выявление типовых проблем ⇒ разработка универсальных решений
- Может использоваться, как инструмент аналитика

Виды мониторинга

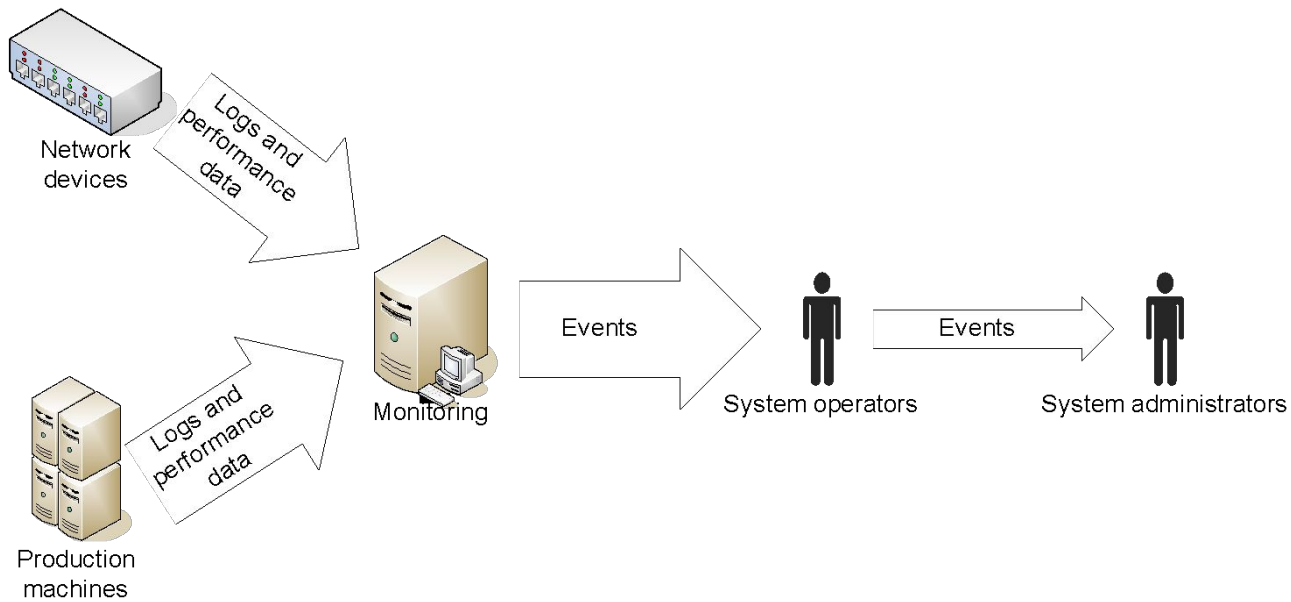
- Физический уровень
(*сеть, доступность сервера, CPU, место на диске, память, IO*)
- Уровень приложения
(*HTTP Errors, Response time, Exceptions*)
- Бизнес уровень
(основан на бизнес критериях)

Методы измерений

- Критериальная система
- Тренды



Система мониторинга



Согласно последним обзорам, производительность фреймворка XYZ выше, чем ZYX.

Давайте разрабатывать систему с использованием XYZ

Java



PHP



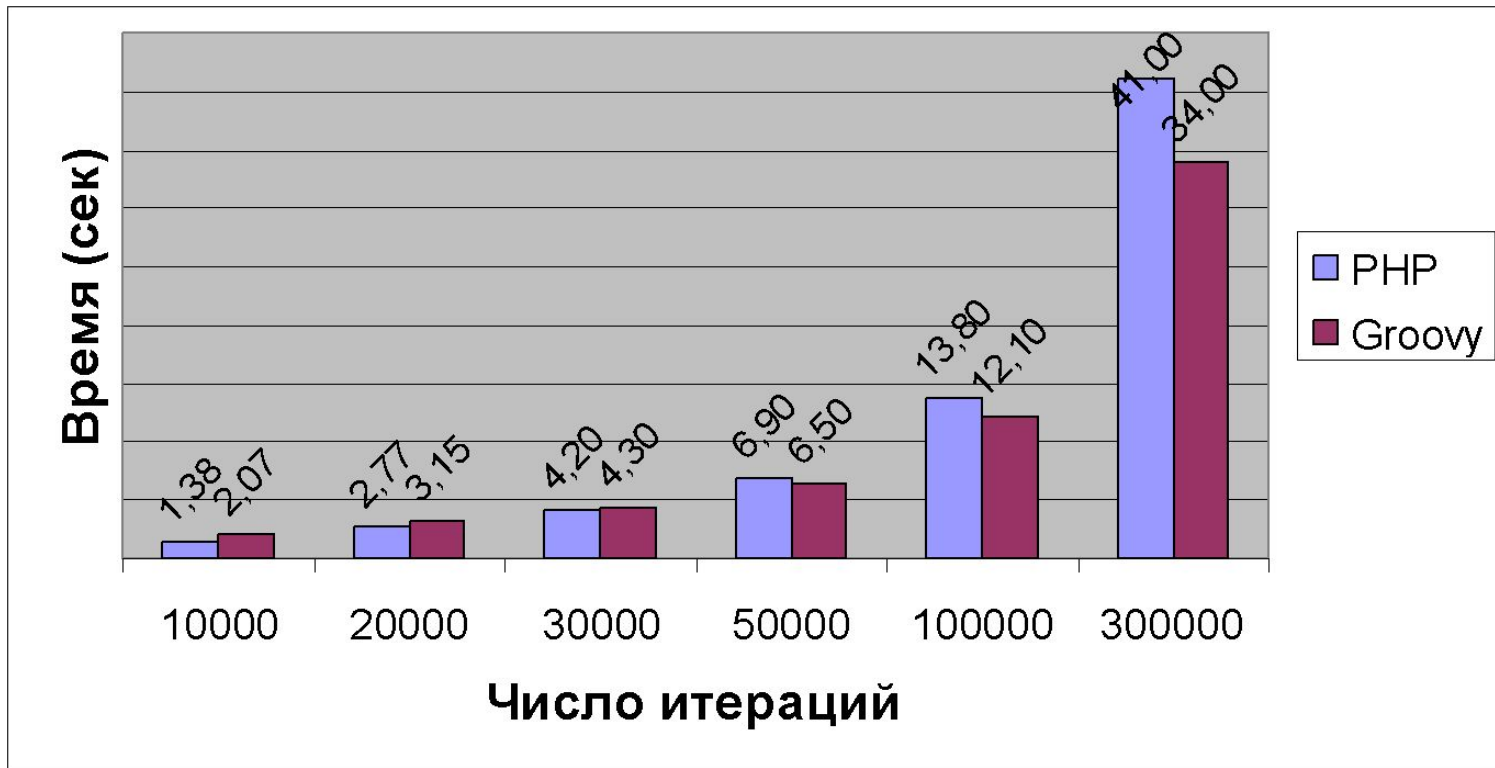
Причины ограничения выбора

- Корпоративный стандарт
- Расширения существующей системы
- Собственная команда разработчиков



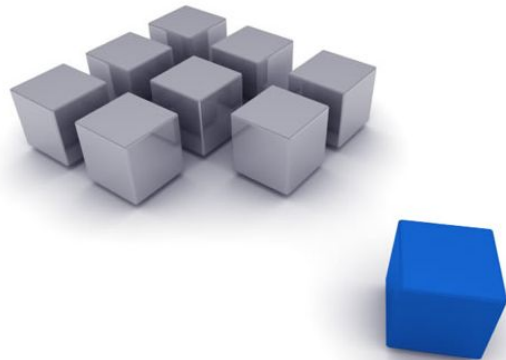
Сравнение фреймворков

- Самый быстрый фреймворк - это тот, которым умеют пользоваться разработчики
- Программа «Hello world» всегда работает быстро



Как выбирать

- Исходите из текущих задач и задач на ближайшую перспективу (время написания первой версии + поддержка)
- Смотреть на профиль команды



*Наши IT-шники не разбираются в вашей системе.
Напишите нам максимально подробную
пошаговую инструкцию, как ее устанавливать и
поддерживать.*



Откуда растут ноги

- Конфиденциальная информация
- Корпоративные стандарт безопасности
- Нежелание разбираться в новых системах



Разделение ответственности

- Человек, который отвечает за систему, должен иметь всю полноту власти
- Можно разделить роли, но не обязанности



Коллективной ответственности не бывает. Коллективной бывает только безответственность

(Валерий Лобановский)

Мы нашли баг в системе, вы можете прислать нам последнюю версию, мы выложим ее сегодня ночью



Очень важно

- Сложность исправления бага определяют разработчики
- Тестирование может занимать намного больше, чем сам фикс
- Может потребоваться значительное обновление системы



Обновление системы

- План работ
- Сценарий проверки
- **План В (если все плохо)**
- **Сценарий проверки плана В**

Формальные процедуры

- Версионность
- Разные ветки для активной разработки и релиза
- Разделение уровней допуска
- Процедуры утверждения релизов

Зачем переписывать код, который был написан всего пару месяцев назад. У нас еще куча фич, которые нужно реализовать.



Типичные ситуации

- Программисты всегда занимаются бессмысленным украшательством, система и так неплохо работает
- Улучшение кода это замечательно, но у нас пока есть более важная работа



Важно

- Расставить приоритеты на каждый этап проекта
- Убедиться, что все разработчики правильно понимают приоритеты каждого из этапов
- Понимать, что рефакторинг – неотъемлемая часть любой разработки
- **Доверять разработчикам**



hl⁺⁺

HighLoad⁺⁺



Вопросы?

artem@gramant.ru

www.gramant.ru

