

# Посткризисное управление жизненным циклом ПО

Сергей Зыков

к.т.н., доц. ГУ-ВШЭ



# О себе

- Зыков Сергей, к.т.н., доц.ГУ-ВШЭ (отделение ПИ факультета УрПО), Master CIW / CWP Designer, iCarnegie Certified Instructor (SSD1/9)
- Стаж в ИТ-отрасли – более 15 лет.
  - В 1996-2007 гг. работал в международной нефтегазовой группе «ИТЕРА» (в т.ч. зам. директора департамента ИТ).
  - Руководил проектами внедрения ERP-систем в компаниях НГК «ИТЕРА».
  - В 2002-2007 гг. руководил проектами создания Интернет- и Интранет-порталов НГК «ИТЕРА», ИПУ РАН и др. корпоративных структур.
- Автор 5 монографий, более 100 статей, ряда учебных курсов по ИТ и корпоративным программным комплексам (МИФИ, МФТИ, ГУ-ВШЭ, IBS, Microsoft Research, ЛАНИТ, ИНТУИТ, SoftLine, ТЕКАМА и др.).
- WWW: [www.hse.ru/org/persons/3468544](http://www.hse.ru/org/persons/3468544)
- E-mail: [SZykov@hotmail.com](mailto:SZykov@hotmail.com)

# Содержание

- Программная инженерия - кризис или депрессия?
- Ответ кризису - "тюнинг" ЖЦ
  - Проект или продукт?
  - Структура ЖЦ и оптимизация затрат
  - "Факторы роста": диалог, дисциплина, стандарты, CASE
- Модели ЖЦ ПО - "скелет" оптимизации
- ООП - возможности и действительность
- От проектирования до передачи
  - Прототип, реализация, сборка: цели, затраты, средства
  - Reuse - важность, артефакты, границы
- Главная цель ЖЦ - сопровождаемость
  - Состав продукта и роль документации
  - Сопровождение: виды, требования, средства, экономика
- ЖЦ ПО - как бороться с кризисом?

# Аналоги кризиса ПИ в экономике

- Кризис в экономике США, Великая депрессия
- Современный глобальный экономический кризис
- Этапы развития производства
  - Искусство (уникальные, неповторимые шедевры)
  - Ремесло («кустари-одиночки»)
  - Мануфактура (специализация, кооперирование, ручной труд)
  - Промышленность (автоматизированный конвейер)
- Выводы
  - Все аналогии не вполне верны  
(параллель с архитектурным строительством)
  - Нужна науч.-тех. дисциплина (программная инженерия)

# Кризис в программной индустрии

- Начало: 60-е гг. XX века
- Причины:
  - «Анархия» разработки (параметры «проектного треугольника» неуправляемы)
  - Непредсказуемый, безответственный, бессистемный ЖЦ
  - Примат АО над ПО (1950-е–60-е )
  - Одна методология – «проб и ошибок»
- Выводы
  - Нужна систематизация проектирования
  - Нужен рост ответственности за результаты труда

# Депрессия в программной индустрии

- Начало: 70-е гг. прошлого века
- Причины:
  - Искусство стало наукой, но не вполне - технологией
  - Созданы крупные центры научной разработки (CMU **SEI**)
  - Затяжной характер «кризиса» => депрессия
  - Рост значимости ПО (жизнеобеспечение, оборона и др.)
  - Отсутствие универсальной методологии
- Выводы:
  - Кризис не побежден (непобедим ???)
  - Депрессия продолжается <= «серебряной пули» нет
  - **Выход из кризиса – оптимизация ЖЦ**

# Зачем нужна программная инженерия?

- ПИ – комплексная ИТ-основа построения ПО
- Причины - нужна дисциплина для построения:
  - Больших систем (терабайты и петабайты данных)
  - Сложных систем (тысячи файлов, десятки модулей, сотни компонент, пример – ERP Oracle E-Business Suite)
  - Качественных систем (надежность, безопасность, отказоустойчивость, эргономика, многократное использование, документация, сопровождаемость, ...)
  - Систем сложной архитектуры (порталы, веб-сервисы, ...)
  - Гетерогенных систем (арх-ры, БД, структурированность)
- Вывод: **ПИ необходима**

# ПИ: проблемы и заблуждения

## Возникают вопросы: Почему...

- ...создание ПО так дорого?
- ... ПО так долго создается?
- ... сложно измерить продвижение проекта?
- ... нельзя обнаружить все ошибки до приемки?

## На эти вопросы сложно ответить:

- В создании ПО **участвует много сторон**: заказчики, разработчики, руководство
- У сторон-участников **разные цели, ожидания и ограничения**. Согласование разумных подходов может серьезно увеличить сроки и стоимость продукта

**При кризисе вывод качественного продукта на рынок в срок – жизненно важен для разработчика !!!**

# ПИ: определения «великих»

**ПИ** – дисциплина, цель которой - производство систем, не содержащих ошибок, отвечающих требованиям заказчика, срокам и бюджету *(по S.R.Schach)*

**ПИ** – комплекс задач, методов, средств и технологий создания (проектирования и реализации) сложных, расширяемых, тиражируемых, высококачественных программных систем (возможно, включающих БД) *(по В.В.Липаеву)*

**ПИ** – дисциплина, охватывающая все аспекты создания ПО от начальной стадии разработки требований до использования *(по И.Соммервиллу)*

**Вывод:** ПИ – комплекс мер по оптимизации ЖЦ ПО

# ПИ и архитектура (SEC, 1968)

(1)

## Аналогия ограничена:

- быстрый прототип ПО не обязан быть надежным
- ошибка в завершеном ПО – не катастрофа
- перезапуск программы – дешевле перестройки здания
- ошибки в ПО накапливаются со временем
- ошибки в ПО труднее выявлять
- строить безошибочное ПО нужно другими методами
- подход «повышенной прочности» неприемлем
- Вместо этого:
  - последовательность, проверка предположений
  - отчеты об ошибках, способы восстановления до сбоя

# ПИ и архитектура (SEC, 1968) (2)

## Аналогия ограничена:

При сопровождении:

- повторная разработка ПО инкрементальна;
- изменения в ПО более масштабны и радикальны (здание может служить 100 лет при min сопровождении);
- ЖЦ ПО гораздо короче;
- ПО быстро морально устаревает
- ПО становится дешевле заменить, чем сопровождать
- Накопленный опыт разработки не всегда ведет к лучшему ПО (быстро меняются сложные платформы)

# Кризис и экономика ЖЦ

## Разработка ПО – многофакторная оптимизация

Вариативность ПО с желаемым выходом по заданной спецификации бесконечна!

**Цель антикризисной разработки ПО:** многомерная оптимизация модели ЖЦ с учетом:

- сроков
- стоимости
- качества
- сопровождаемости

Приоритетность параметров оптимизации зависит от характера и масштаба ПО и др. кризисных факторов

Для оптимизации ЖЦ необходимы четкая декомпозиция на процессы, адекватный выбор методов и средств разработки ПО

# Кризис и экономика ЖЦ

## Стадии ЖЦ ПО, общие для всех моделей:

- Анализ требований
- Подготовка спецификаций
- Проектирование
- Реализация
- Интеграция
- Сопровождение
- Вывод из эксплуатации

**Оптимизация может упрощать (и даже исключать!) отдельные стадии ЖЦ**

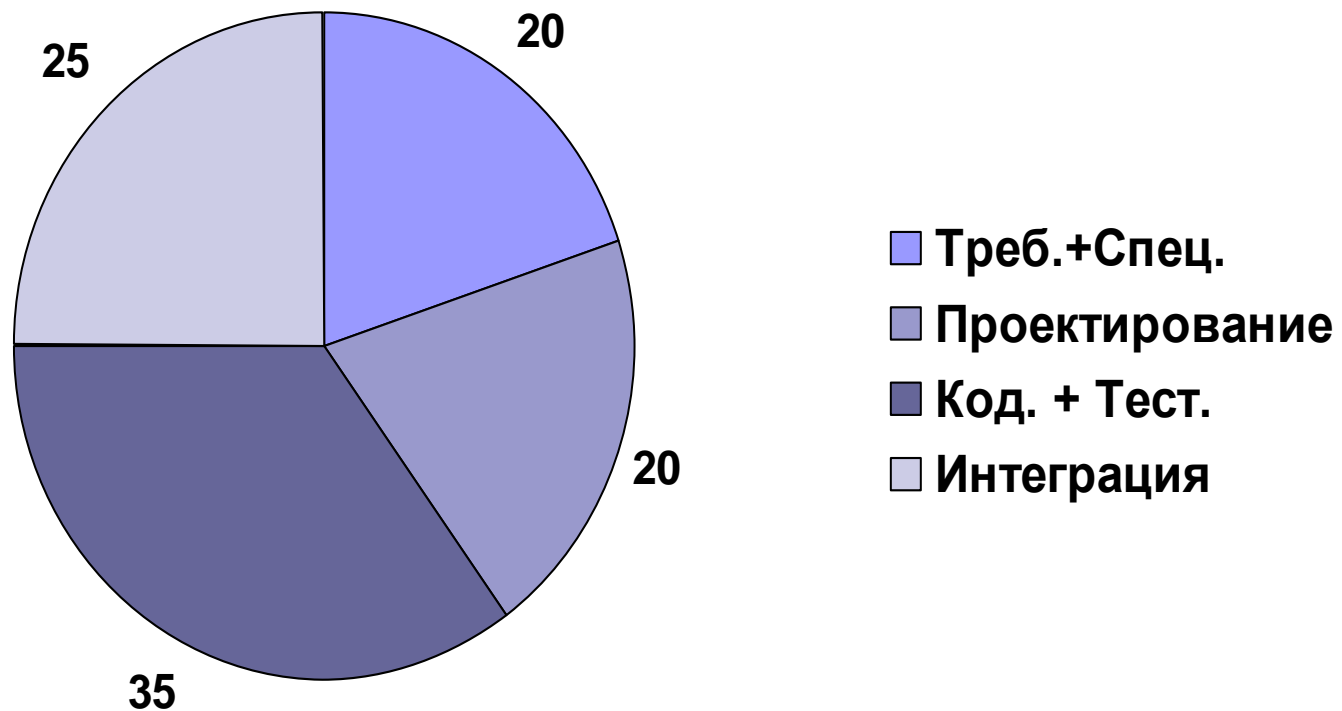
# Кризис и экономика ЖЦ

## Примерный вклад фаз ЖЦ в стоимость ПО, %



# Кризис и экономика ЖЦ

## Примерный вклад фаз ЖЦ в сроки проекта, %



# Кризис и экономика ЖЦ

- Основные затраты – на сопровождение (особенно для длительных проектов и сложных систем)
  - Средства, увеличивающие расширяемость ПО (и/или сроки обновления и тестирования) более эффективны, чем все ухищрения при кодировании
  - Фазы, предшествующие кодированию и следующие за ним, составляют 30% затрат (кодирование – 5%):
    - «обрамляющие» стадии улучшают ПО и ускоряют кодирование
  - Большинство ошибок – в ходе проектирования и спецификаций (нужны формальные методы анализа)
  - Цена поиска ошибок экспоненциально растет по мере продвижения проекта по ЖЦ
  - **Ошибки следует выявлять ASAP!**
- (иначе – сквозная трассировка **ВСЕХ** артефактов ПО)
- **Грамотная постановка, выбор модели/архитектуры, четкая ревизия проекта, дисциплина, стандарты, знание CASE, разумная достаточность документации – ключи к экономии**

# Модели ЖЦ ПО - "скелет" антикризисной оптимизации

- Модель «проб и ошибок» (build-and-fix)
- "Однопроходная" водопадная модель
- «Циклический» ЖЦ:
  - Модель быстрого прототипирования
  - Инкремент(аль)ная модель
  - Модель синхронизации и стабилизации
  - Спиральная модель
  - ОО-модель

# Модели ЖЦ ПО в сравнении

Модель ЖЦ	Преимущества	Недостатки
«Проб и ошибок»	Экономия для малых проектов с простыми требованиями/сопровождением	Неприменима для проектов от 1000 строк
Водопадная (один проход)	ПО за 1 проход, четкая дисциплина, строгая документация	Однопроходность не гарантирует соответствия ПО требованиям заказчика
Прототип	Экономичное уточнение функционала Техническая «неграмотность» клиента	Нетехнологичный код
Инкрементальная (эволюционная)	«Рабочее» ПО за 1 проход => Быстрый ROI плавная сопровождаемость	Нужна расширяемая архитектура, при «революции» требований может выродиться в «пробы и ошибки»
Синхронизации	Быстрый поиск ошибок, интероперабельность и интеграция	Масса специфических знаний (тесты и др.)
Спиральная	При прототипировании включает преимущества целого ряда моделей	Крупные внутренние проекты; затратное управление рисками
ОО-модель	Итеративная разработка с распараллеливанием	Без дисциплины, стандартов, знаний CASE может выродиться в «пробы и ошибки»

# Оптимизация объектной модели

- Структурный или объектный подход?
- Каковы принципы ООП?
- Идеология последовательной конкретизации - доколе?
- UML - зачем нам столько диаграмм?
- Прочие диаграммы (ERD, DFD и т.д.) - когда и в какой мере?
- Архитектурное проектирование - "скелет" продукта
- Эскизное и детальное проектирование - теория и практика
- Модели статики и динамики - подходы и ограничения
- Ревизия проекта - особенности и техника
- Выводы: ООП - как реализовать потенциал?

# Принципы проектирования

- **Модульность** – возможность декомпозиции продукта на на малые, самодостаточные компоненты-модули
- **Сцепление** – мера сходства или взаимодействия компонентов модуля
- **Связность** – мера взаимодействия модулей продукта
- **Преимущества модульности** – в легкости:
  - Восприятия - ПО структурировано, компоненты слабо независимы
  - Тестирования - локализация ошибок упрощается
  - Сопровождения – влияние модуля на другие сведено к минимуму

**Высокое сцепление, низкая связность – ПОЧЕМУ?**

# От проектирования – к реализации

## Пути экономии

**Выбор модели ЖЦ ПО** – критический фактор для сроков, стоимости и успеха внедрения:

- **«Однопроходная»** – параллелизм кодирования и тестирования – сборка – по их окончании
  - водопадная/каскадная
- **«Циклические»** – итеративность фаз для каждого релиза
  - «проб и ошибок»
  - инкрементальная
  - эволюционная
  - синхростабилизации
  - спиральная
- **«Параллельные»/«интегрированные»** – тесное взаимодействие процессов кодирования, тестирования и сборки
  - Объектно-ориентированная

**Критерий перспективности подхода - % reuse**

# Модули: с «чистого листа» или reuse?

**Reuse (многократное использование)** – использование модулей одного продукта при производстве другого, с иными функциональными требованиями

## **Виды reuse:**

- Непреднамеренное – допускается программистами
- Целенаправленное – модуль изначально создается с целью reuse

**Примеры:** MS .NET Framework, Windows Forms, Remoting, WCF, Enterprise Library, Office Extensions, DLL, API, GUI

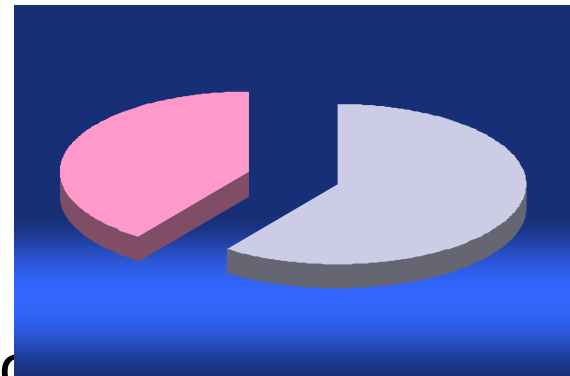
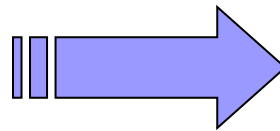
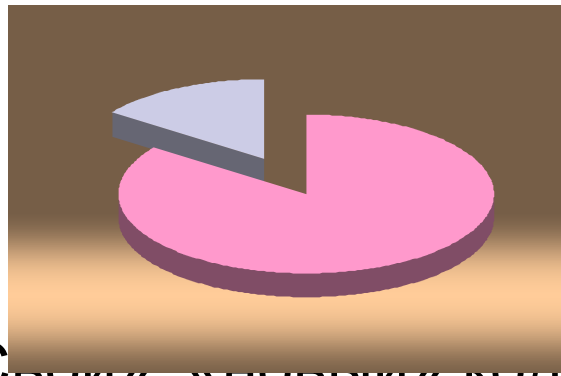
## **Целенаправленный reuse лучше и в кризис.** Условия:

- соблюдение стандартов
- документирование,
- тестирование (в т.ч. сборочное)
- дисциплина разработки

# Reuse: экономика и проблемы

## Проблемы:

- «Ножницы» между желаемым и действительным:
  - Потенциальный reuse - до 85% кода (инновации – около 15%)
  - Реальный reuse - до 40% кода



- «Свой», «новый» код лучше «чужого» (амбиции кодера)
- «Чужой» код «ненадежен» (сомнения в качестве)
- «Где, когда и зачем искать код?» (сложность поиска по БД)
- «Слишком дорого» (кодирование стоит +60%, а еще reuse)
- «Юридические казусы» (клиент получает права на ПО при контрактной работе)

# Юнит-тесты

- **Преимущественные** направления технологий:
  - Просмотр/инспекция – ошибки интерфейса
  - «Черный ящик» – ошибки логики управления
  - Док-ва корректности – ошибки редких и важных модулей
  - Анализ надежности – оценка оставшихся ошибок
- **Тип и рамки тестов – выбор РМ:**
  - Экономический анализ
  - прекращение тестов
  - Перекодирование
  - Порог ошибок/надежности
- **Технологии близки по эффективности !**
- **Технологии комбинируются с учетом:**
  - Характера/масштаба проекта,
  - знаний/зрелости команды,
  - экономики проекта (в т.ч. «кризисных» особенностей)

# Сборочные тесты

- **необходим план интеграции и тестирования**
  - общий порядок, технологии, процессы, ответственные, структура затрат
  - область ответственности группы контроля качества ПО
- **необходим гибкий выбор логических/операционных модулей**
- сборка выявляет ошибки кодирования
- **для интерфейсных тестов нужны CASE-средства**
- **следует совмещать модульное и сборочное тестирование**

# Внедрение и сопровождение

- Продукт – это только код?
- Что мы передаем заказчику?
- Документация к ПО - роль, состав, границы?
- Главная цель разработки? – Сопровождение!
- Как достичь "бесшовного" сопровождения?
- Сопровождение и документирование – что нам поможет?

# Продукт – не только код!

## Документация по фазам ЖЦ ПО

ФАЗА ЖЦ	ДОКУМЕНТАЦИЯ
Анализ требований	Документ требований / прототип продукта
Спецификация	Документ спецификации (деревья решений, неформальные спецификации, спецификации ввода-вывода, ER- и DF- диаграммы, ...)
Проектирование	Архитектурный, эскизный, рабочий проект (а также записи обсуждений и т.д.)
Реализация	Построчная, модульная, модульного тестирования (наборы, рез-ты)
Интеграция	Сборочного тестирования (наборы, результаты), по продукту для заказчика
Сопровождение	Обновление документации (требования, спецификации, проект, тесты, модули)
Вывод из эксплуатации	Документация не производится

# Документация к ПО: зачем?

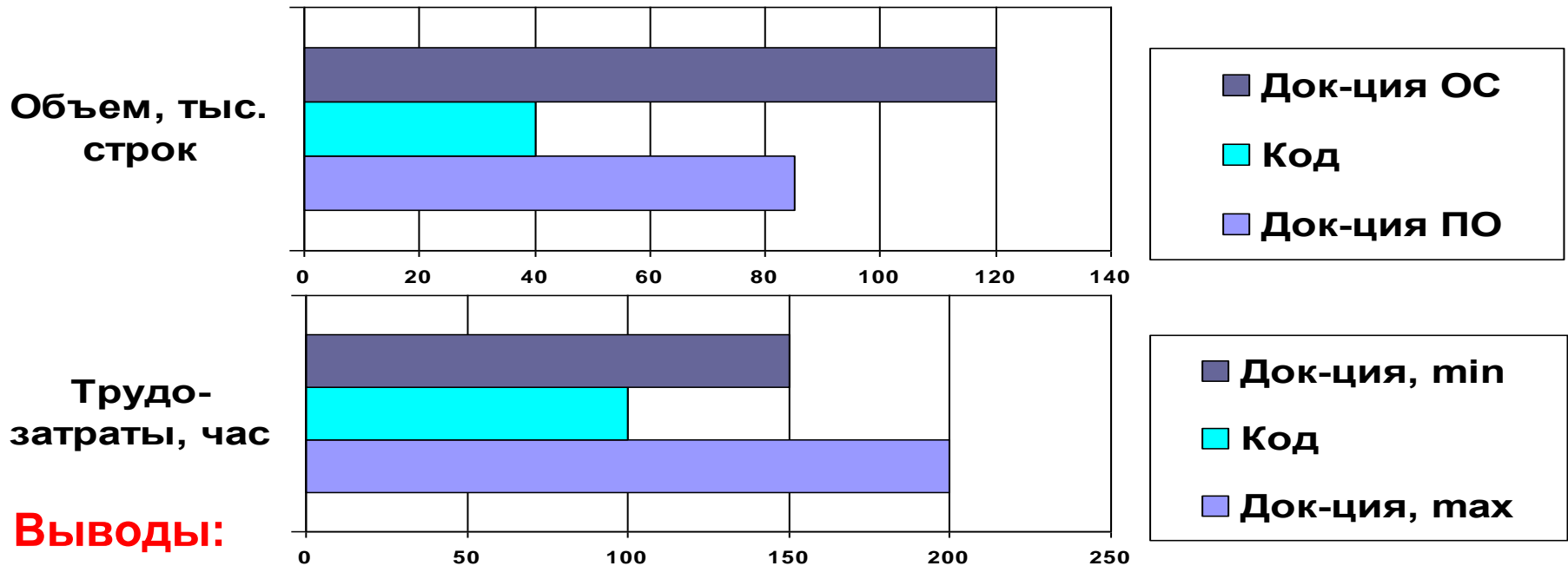
## Причины необходимости:

- Текучесть/мобильность кадров
- Сжатые сроки
- Коррекция проекта на фазе реализации
- Ошибки/проблемы сопровождения

## Выводы:

- Продукт – не только код
- Код НЕ документирует себя сам!
- План документирования – часть плана проекта
- Весь ЖЦ нужно тщательно документировать
- Документацию готовят участники процесса
- Итоговая проверка документации до передачи:
  - Внутренняя корректность (полнота, однозначность, непротиворечивость)
  - соответствие стандартам и коду
- Порядок документирования зависит от модели ЖЦ (каскадная – завершение документации перед переходом к след. фазе)

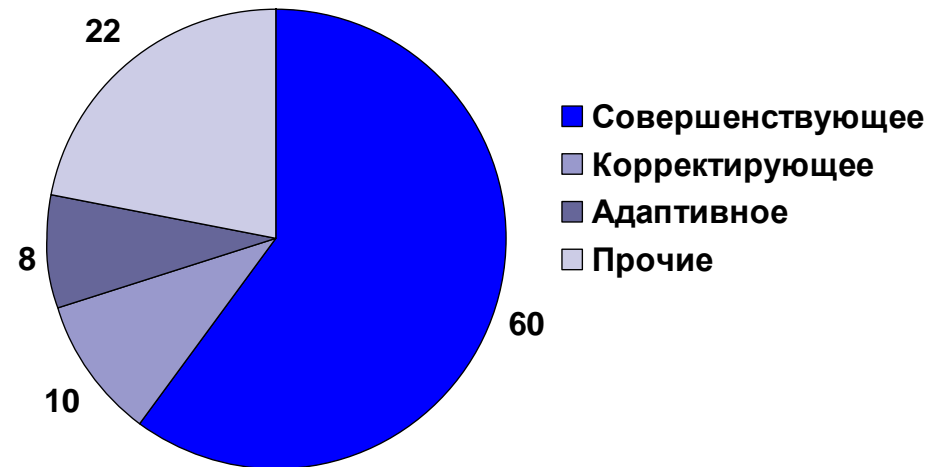
# Документация: что нам поможет?



## Выводы:

- Документация – объемная и затратная часть ЖЦ ПО
- Документация – неотъемлемая и необходимая часть ЖЦ ПО
- Пути роста снижения трудозатрат на документирование:
  - соответствие характеру и масштабу проекта
  - применение CASE
  - стандарты (международные, федеральные, отраслевые, корпоративные)

# Экономика сопровождения



## Типы:

- Совершенствующее – рост производительности / функциональности
- Корректирующее – устранение в продукте остаточных ошибок
- Адаптивное – коррекция продукта с учетом нового АО и ПО заказчика

## **Выводы:**

- Крайне сложная, многоаспектная и **самая затратная** часть ЖЦ ПО
- Не применять в новом релизе более одного вида сопровождения

# Выход из кризиса – оптимизация ЖЦ

ПИ – комплекс мер по оптимизации ЖЦ ПО с учетом характера и масштаба  
Оптимизация может упрощать (и даже исключать!) отдельные стадии ЖЦ

## Ключи к экономии (и успеху):

- раннее выявление ошибок
- грамотная постановка,
- выбор модели/архитектуры, процессов/методов/средств
- выбор логических/операционных модулей
- четкая ревизия проекта,
- знания/зрелость (дисциплина, стандарты, CASE)
- разумная достаточность этапов (метрики)
- модульность, высокое сцепление, низкая связность
- комбинирование технологий
- планирование и управление (PM)
- человеческий фактор

## Критерии перспективности подхода:

- % reuse
- сопровождаемость



# Благодарю за внимание!

## Вопросы?

***Сергей Зыков (ГУ-ВШЭ)***

SZykov@HSE.Ru

SZykov@hotmail.com

<http://www.hse.ru/org/persons/3468544/index.html>

<http://zykov.altweb.ru>

***Careerlab: 127055***, г.Москва, ул.Образцова д.14/2

***Телефон: + 7 (495) 775-1543***