Восьмая независимая
научно-практическая конференция
«Разработка ПО 2012»
1 - 2 ноября, Москва

Обуздание Legacy-систем (из личного опыта)

Александр Мармузевич

РОНД (СТ Группа), Минск,
Беларусь

Система

Legacy система заказчика, это то, что:

- Кто-то когда-то **разработал**.
- Кто-то когда-то **внедрил**.
- И с тех пор заказчик её **эксплуатирует**.

И наверное она уже устарела…

Входные условия

- Система работает, и менять её на что-то новое нежелательно или пока затруднительно.
- Есть потребность в оптимизации, обновлении платформ, интеграции с другими системами, решении других задач.

- Разработчик объявил систему «sunset» продуктом и не горит желанием продолжать поддержку.
- Заказчик выкупил исходные коды и договорился с разработчиком о передаче знаний по системе.

План действий

- Заказчик ищет IT-компанию и создает на её основе центр компетенции для последующей поддержки системы.
- Разработчик проводит «knowledge transfer» от специалистов разработчика к специалистам центра компетенции.
- Разработчик по согласованию с заказчиком передает систему на поддержку специалистам центра компетенции.

Реализация плана действий

- Заказчик нашел близкую по специализации IT-компанию.
- В IT-компании сформировали группу (ядро) будущего центра компетенции и приступили к изучению документации.
- Разработчики за 2 месяца провели теоретические тренинги «Hi-Level overview» и «Intermediate course» и «Workshops»
- Заказчик «передал» центру компетенции «какие-то исходники» системы и приступил к развертыванию среды разработки и тестирования.

«Все хорошо, падаю...»

Характеристики системы:

- Автоматизированная банковская система, написанная на смеси COBOL + C + VB, разрабатываемая с 1996 по 2004 год разными группами разработчиков.
- Около 150m исходных кодов, порядка 10000 модулей.
- Есть высокоуровневые спецификации.
- Отладчиков нет, привычных IDE нет, автоматических тестов нет, полное тестирование системы никогда не проводили (по словам разработчиков), базы знаний по системе нет.

Наши действия

- Разбиваем систему на основные процессы, пытаемся задокументировать (в Wiki) как это все работает в привязке к исходным кодам.
- Пересобираем систему из переданных заказчиком исходных кодов.
- «Выбиваем» из заказчика проект по реализации API для вызова функций системы извне.
- Ввязываемся в анализ практических «кейсов».

Промежуточные итоги

- Идея с Wiki оказалась неудачной. Что-бы что-то найти, надо знать, что искать.
- Систему пересобрали, но с большим трудом и стойким ощущением, что нас на самом деле учили чему-то не тому.
- Первые функции API все-таки сделали, «кейсы» проанализировали, но внутренние трудозатраты повергли в уныние.
- Возникло стойкое ощущение, что что-то делаем не так.

Приближается Epic Fail

Анализ ситуации

- Исходный план построения центра не сработал (не те условия).
- Чтобы удовлетворить заказчика нам потребуется или расширить штат в 5 раз от запланированного, или потратить на изучение системы 5 лет.
- Можно попробовать купить специалистов разработчика, но это маловероятно и вне бюджета.

Формулировка проблемы

- Мы должны знать систему не хуже лучше специалистов разработчика.
- У нас на изучение системы нет ни ресурсов ни времени.

- Где и как по-быстрому получить нужные знания?

Решение

- Отказываемся от изучения всей системы, но изучаем только то, что нужно в данный момент.
- Выделяем « типовые процессы» решения задач центра.
- Разрабатываем «магический инструмент», который автоматизирует эти «типовые процессы».
- Обучаем специалистов работе с этим «инструментом».

Результат

На сегодняшний день мы при помощи «магического инструмента» в состоянии реагировать на обращения заказчика и решать поставленные задачи даже несколько более оперативно, чем разработчик.

Ингредиенты «магии»

- Исходный код – основной источник знаний.
- Парсер – инструмент извлечения знаний.
- Анализатор – инструмент использования знаний.

Парсер

- Строит синтаксические деревья по всему исходному коду и рассчитывает взаимосвязи между сущностями системы.
- Сохраняет информацию в базе данных.
- Поддерживает COBOL, C, Shell
- Учитывает всю специфику исходного кода поддерживаемой системы.

Анализатор

- Используя базу данных, созданную парсером:
  - Умеет искать информацию по базе данных
  - Позволяет осуществлять быструю навигацию по коду
  - Умеет извлекать и визуализировать информацию по алгоритмам, маршрутам обработки, стекам вызовов и многому другому.
- Кроме того:
  - Содержит модули для кодогенерации и маппинга переменных.
  - Имеет модуль справочной информации по кодам ошибок и т.д..

Как это все работает в связке?

- Заказчик
  - ставит задачу и предоставляет набор артефактов (скриншоты, логи, обрывки информации).
- Аналитик при помощи «инструмента»:
  - Быстро локализует область по артефактам, находит «workaround» либо готовит постановку задачи для новой разработки;
- Разработчик
  - Выполняет разработку по подготовленной постановке.
- Разработчик «инструмента»:
  - Анализирует действия аналитика и разработчика и, по возможности, улучшает инструмент.

Что в итоге?

- Нет необходимости тратить время и ресурсы на глубокое изучение системы, достаточно научиться пользоваться «магическим инструментом».
- Работы по локализации предметной области и постановке задачи на разработку может выполнять специалист со сравнительно невысокой квалификацией.
- Знания по работе системы принципиально не могут «уйти» с уходом специалистов.

Мы обуздали legacy-систему.

Наверное, нам просто повезло ☺