**Проблемы и риски при разработке программного обеспечения**

Разработка программного обеспечения — это альтернатива выбору и покупке готовых программ. Часто выбор делается в пользу готового программного обеспечения из-за рисков и проблем, с которыми заказчик может потенциально столкнуться при собственной разработке (путем найма специалистов) или обращения в компанию при разработке программного обеспечения.

Существует несколько наиболее распространенных проблем, приводящих к тому, что либо проект не выходит в свет вовсе, либо заказчик получает продукт, которым невозможно пользоваться. Про эти проблемы, а также способах их избежать, пойдет речь в данной статье.

**Отсутствие четкого плана и технического задания**

Часто вопрос ставится так — нам необходимо автоматизировать компанию, для этого наймем программиста, пусть автоматизирует. Только что именно он будет делать? От формулирования бизнес-задачи, например “хотим чтобы коммерческие предложения выводились автоматически”, или “надо вести в системе заказы и должны быть звонки клиентов”, до даже начала программирования лежит огромная пропасть, процесс, называемый проектированием. Важно понимать, что это необходимо сделать. Надеяться, что программист выступит в роли бизнес-аналитика, проектировщика, системного архитектора и дизайнера и сделает все эти задачи хорошо — ну, можно попробовать. И некоторые программисты даже обещают это сделать. Есть даже такой термин — крюгеризм.

Нужен проект. И план. Проект — это и есть информационная система, программное обеспечение. Проект включает в себя полное техническое задание и изображения интерфейсов. Обе эти составляющие также могут быть разного качества.

Техническое задание часто путают с функциональным описанием. Когда заказчик говорит, что у него есть техническое задание, то хорошо, если это не просто на один лист набросанные названия разделов, а хотя бы несколько листов с описанием того, как он видит эту систему и что в ней должно быть. Но не стоит путать это с техническим заданием, которое должно содержать:

1. Цели и назначение информационной системы;
2. Требования к стеку технологий;
3. Требования к качеству программного кода и соблюдению стандартов разработки;
4. Структуру данных, полное описание будущей базы данных, всех таблиц и полей включая типы этих полей и связи между данными;
5. Текстовое описание интерфейсов, всех их элементов и логики работы, которое должно соответствовать прототипам и дизайну;
6. Описание логики работы в системе, что необходимо сделать по действиям для достижения результата работы функции системы.

Дизайн системы состоит из изображений или интерактивного макета, всех интерфейсов и их элементов. Дизайн-макет определяет, как будет выглядеть система и позволяет вам, как заказчику, до начала разработки точно понять не только как будет выглядеть, но и как будет работать будущая система. Можно пройтись по интерфейсам представляя действия в системе. Для разработчика интерфейсов — это четкий макет, по которому делается интерфейс. Если оставить внешний вид на милость программисту, то у вас будет одна большая кнопка на весь экран. И это будет соответствовать техническому заданию, ведь в нем сказано, что должна быть кнопка.

Имея техническое задание и прототипы, вы уже можете или нанимать программистов, или обращаться в IT-компании для сбора предложений стоимости реализации, в общем принимать решения.

Для того, чтобы составить такой проект, вы также можете обратиться в IT-компанию. Составить проект самостоятельно достаточно сложно без специальных навыков и знаний, как и найти одного человека который будет и аналитиком и дизайнером — это тоже задача практически невыполнимая.

В данном случае разработку программного обеспечения можно сравнить с постройкой дома. Если вы надеетесь нанять программиста, еще и недорого, чтобы он всё сделал, то это как нанять специалиста, специализирующегося по кладке кирпича и ожидать от него построенный дом. А отсутствие проекта — это сказать мастеру, что дом надо 8 на 10 пожалуйста, желательно двухэтажный.

По причине того, что уже используемые системы являются шаблонными, не поддаются изменениям или эти изменения очень трудоемки, выбирается вариант заказать информационную систему для решения локальной задачи или конкретного бизнес-процесса.

**Устаревший или редкий стек технологий**

Это можно назвать — что умею, то и делаю. И тут тоже происходит отсылка к проекту, где должны быть указаны технологии.

Очень частая проблема, когда проект или функционал подстраивается под специалиста, который умеет что-то делать на своём языке программирования. При этом одиночные специалисты, как правило, не имеют объективного понимания применимости тех или иных технологий к задачам. Они предлагают написать проект на том, чем умеют и считают лучшим, но лучшим для себя, а не для проекта. Кроме того, в современном мире технологии вообще разделены на технологии интерфейса и серверной части, а для этого нужны разные люди с разными навыками. Если это делает один человек, то скорее всего это будет система без разделения на интерфейсную и серверную часть, а устаревший метод формирования интерфейса на сервере и отдачи браузеру. Варианты написания на разных C# или Delphi вообще не рассматриваются.

Часто бывает так, что программист умеет писать на Python и естественно считает его самым лучшим языком. На Java, Delphi или десятках других. И убеждает заказчика в преимуществах. Это не значит, что нет задач, которые лучше выполнять на тех или иных языках программирования, это действительно так. Но не нужно подгонять проект под язык, который знает тот программист, что попался под руку.

Довольно часто к нам обращаются клиенты, которым какой-то программист написал систему на Java/Python/Delphi и просят эти системы доработать. К сожалению, им будет трудно найти тех, кто согласится это делать. Иногда проще сделать заново, с нуля, на адекватном современном стеке технологий.

Чтобы проверить, будет ли вам легко найти специалиста по тому или иному языку программирования, сравните количество резюме на HeadHunter введя название языка программирования в поисковую строку.

**Качество разработки**

Качество определяется двумя критериями — архитектура будущего продукта и качество программного кода.

Приведем пример. Первая информационная система написана на шаблонном движке (один из коробочных вариантов распространения, который дорабатывается под нужды клиента). Для того, чтобы получить какие-то данные на экран отправляется запрос на сервер, на сервере включается весь механизм движка, собираются данные для ответа, формируется и выдается браузеру новый интерфейс, который появляется у вас в браузере, когда перезагружается страница. Чтобы что-то доработать, программный код пишется как бы дополняя код движка. Как следствие — система обрастает кодом с каждой доработкой, при этом всякий, кто пишет код, пишет его так, как считает нужным, меняя систему и внося изменения туда, куда считает нужным. Второй пример, сервер и интерфейсная часть системы — это полностью отдельные системы. Интерфейс не перезагружается, а каждый блок информации в интерфейсе, будь то таблица или панель, живут отдельной жизнью. Обмен данными каждый блок интерфейса с сервером выполняет самостоятельно, но при этом этот обмен четко структурирован в соответствии с архитектурой обмена данными между сервером и интерфейсом. Если меняются какие-то данные на сервере, то чтобы они были показаны пользователю, перезагружать страницу не нужно — сервер сам сообщит нужному элементу интерфейса, что данные изменились и передаст их для показа пользователю. Обмен данными производится по определенному протоколу и в соответствии с заранее определенной структурой. На сервере подобные запросы принимаются также в соответствии с архитектурой в определенных точках и обрабатываются по заложенной в системе логике. В такой реализации “наколхозить” или сделать в системе при ее доработке что-то неправильно уже не получится, необходимо писать в соответствии с заложенной архитектурой.

Второй компонент — качество программного кода. Законом программный код приравнен к литературным произведениям (в плане авторства). И как литературное произведение программный код можно написать так, что никто ничего не поймет, а можно логично и стройно. Для каждого языка программирования, да и вообще в программировании, есть общепринятые нормы и стандарты, как для интерфейсных частей, так и для серверных. Например, для языка программирования есть общепринятые мировые стандарты, именуемые PSR (PHP Standart Recommendations). Их много, часть из них принята сообществом к использованию, часть нет. Но суть в том, что профессиональный программист, соблюдающий стандарты и рекомендации, садясь за код, написанный другим человеком может спокойно его поддерживать. Существуют средства разработки, позволяющие проверять код на соответствие стандартам и даже на наличие ошибок (средства автоматизации для разработчиков). Важность соблюдения стандартов переоценить трудно, или это набор непонятного когда, в котором никто не разберется, или это логичный код с понятной архитектурой.

Таким образом, если вы фиксируете в техническом задании стандарты разработки и архитектуру, стек технологий, то снижаете риски последующей полной неподдерживаемости проекта.

**Вывод**

В индивидуальной разработке программного обеспечения много подводных камней. Выше перечислены только некоторые распространенные риски и проблемы, но следование рекомендациям позволит их существенно сократить.

В первую очередь необходимо четкое подробное техническое задание. В нем должны быть зафиксированы стандарты и подходы. Проект должен быть детальным образом расписан, каждый элемент системы, каждое вычисление. Должен быть дизайн интерфейса. Только после этого можно выбирать вариант, как будет разрабатываться информационная система и кем.

На нашей практике часто встречались случаи, когда заказчику разные специалисты предлагали сделать информационную систему (на основе функционального описания просто с перечислением разделов) за стоимость пары месяцев зарплаты среднего программиста, что сразу вызывает подозрения. После нашего проектирования и просьбы заказчика оценить стоимость разработки по проекту у различных подрядчиков эта стоимость возрастает на порядки, потому что при правильном проектировании и учете в проекте всех нюансов уже не сделать что-то некачественное, упрощенное, а придется делать именно то, что в проекте и соблюдать критерии качества.

**Проблемы разработки сложных программных систем**

Сложные ПО:

1. сложность формального определения требования к программным системам

2. коллективная разработка

3. необходимость увеличения степени повторяемости кодов

4. отсутствие удовлетворительных средств формального описания поведения дискретных систем.

I.

1. При определении требований необходимо учесть большое количество различных факторов.
2. разработчики не являются специалистами предметной области и не могут сформулировать проблему в нужном ракурсе.
3. В процессе создания программных систем используют языки сравнительно низкого уровня, это приводит к крайней детализации операций и увеличивает объём описаний разрабатываемых программ. Средств позволяющих детально описывать поведение сложных систем на более высоком уровне, чем язык программирования не существует
4. Коллективная разработка

Необходимость увеличения и степень кодов. Для повышения производительности труда компании стремятся к созданию библиотек компонентов которые можно использовать в дальнейшей разработки. При этом компоненты стараются делать более универсальные, что в итоге приводит к увеличению сложности разработки.

**Каскадная.**

Такая модель предполагает переход на следующую стадию только после полного завершения предыдущей стадии.

Достоинства:

1) получение в конце каждой стадии законченного набора проектирования документации

2) простота планирования процесса разработки.

Недостатки:

1) при неточных спецификациях приводят к пересмотру уже принятых решений.

2) Изменение требования заказа непосредственно в процессе разработки.

**Модель с промежуточным контролем.**

В данной схеме можно вернуться на любой уровень.

**Спиральная модель.**

В соответствии с этой схемой программное обеспечение создается не сразу, а итерационно с использованием метода прототипирования. Прототипом называют действительный программный продукт, реализовывающий отдельные функции и внешние интерфейсы разрабатываемого программного обеспечения. На итерации как правило специфицируют, проектируют, реализуют и тестируют интерфейс пользователя. На 2) добавляют некоторые ограниченный набор функций, на последних этапах этот набор расширяет возможности разрабатываемого программного продукта. Основным достоинством является начин-ся с некоторой итерации, на которой обеспечена определяемая функциональная полнота, продукт можно предложить пользователю, что позволяет:

- сократить время до появления новых версий продуктов

- заинтересовать большое количество пользователей за счет продвижения следующих версий.

- ускорить формирование и уточнение спецификаций за счет практики использования продукта

- уменьшить вероятность морального устарения системы за время разработки.

Разработка спиральной модели жизненного цикла программы и КЕЙС технологии позволили сформулировать условия выполнения которых сокращает сроки создания ПО. Современные технологии проектирования, разработки и сопровождения ПО должна отвечать следующим требованиям:

1. Поддержка полного жизненного цикла ПО

2. Гарантированное достижения цели разработки с заданным качеством и в установленное время.

3. Возможность выполнения крупных проектов виде подсистем, разрабатываемые группой производителей (3-7 человек) с последующей интеграцией составных частей и координации ведения общего проекта.

4. Минимальное время получения работоспособной системы

5. Возможность управления конфигурации проекта, ведения версий проекта и автоматического корректного выпуска документации по каждой версии.

6. Независимость выполняемых проектных решений от средств реализации.

Этим требованиям отвечает технология RAD (быстрая разработка приложений). Эта технология проектирования созданная для достижения высокой скорости, высокого качества и низкой стоимости разработки ПО. В этой технологии процесс делится на следующие этапы:

1. Анализ и планирование требования пользователя

2. Проектирование

3. Реализация

4. Внедрение

На 1м этапе определяются приоритетные требования и направления ограничивающие масштаб проекта. На 2м этапе детально описываются процессы системы, требования к данным и доступа к ним и определяют состав необходимой документации. По результатам анализа этих процессов определяют количество функциональных точек и принимают решение о количестве подсистем и команд участвующих в разработке. Под функциональной точкой в технологии RAD понимают любой из функциональных элементов разрабатываемых системой, к которым относится: входной элемент приложения, выходной элемент приложения, запрос, совокупность записей, интерфейс приложения.

Нормы определяющие количество человек для разработки человек: менее 1000 функциональных точек – 1 человек, от 1-4000 точек – команда (3-7 человек). На каждые 4000 точек – 1 команда.

На этапе реализации выполняется интерактивное построение реальной системы. На этом этапе активно работают с заказчиками.

На этапе внедрения проводят обучение пользователей и осуществляя постепенный переход на новую систему. При этом эксплуатация старой версии продолжается до полного внедрения.

**Понятие технологичности ПО.**

Под технологичностью понимают качество проекта программного продукта, от которого зависят трудовые и материальные затраты на реализацию и последующие модификации. Технологичностью ПО определяется проработанность его модулей, уровнем независимости модулей, стилем программирования и степенью повторного использования кодов. Чем выше независимость модулей, тем легче понять, реализовать, модифицировать и находить ошибки.

При проектировании достаточно сложного ПО выполняют его декомпозицию на модули. Модуль – автономно компилируемая программная единица. К модулям предъявляются следующие требования:

1. 1 точка входа

2. 1 точка выхода.

3. Отдельная компиляция

4. Соответствие принципу вертикального управления

5. Выполнение одной функции

6. Небольшой размер (50-70 операторов)

7. Возможность вызова других модулей

Уменьшение зависимости модулей улучшает технологичность проекта. Степень независимости модулей (подпрограмм и библиотек) оценивают 2мя критериями: сцеплением и связанностью. Сцеплением является мера взаимозависимости модулей, которая определяет на сколько хороши модули отделены друг от друга. Модули не зависимы если каждый из них не содержит о другом никакой информации.

Существует 5 типов сцепления модулей:

1. модули обмениваются данными. При небольшом количестве обмениваемых данных этот тип сцепления обеспечивает наилучшие технологические характеристики ПО.

2. сцепление по образцу – модули обмениваются данными, объединённые в некоторые структуры (элементы массива)

3. Сцепление по управлению – когда 1 модуль посылает другому некоторой информационный объект, предназначенный для управления внутренней логикой модуля.

4. Сцепление по общей области данных – предполагается, что разные модули работают с одними данными. Такой вид сцепления считается недопустимым.

5. Сцепление по содержимому – один модуль содержит обращение к внутренним компонентам другого. Такой вид сцепления тоже не допустим.

Связанность модулей – мера прочности соединения функциональных и информационных объектов внутри одного модуля. Если сцепление характеризует качество деления модуля, то связанность характеризует степень взаимосвязанности элементов реализуемым одним модулем. Различают следующие виды связанностей, перечисляются в порядке убывания уровней:

1. Функциональная

2. Последовательная

3. Информационная (коммуникативная)

4. Процедурная

5. Временная

6. Логическая

7. Случайная

Функциональная – все объекты модуля предназначены для выполнения одной функции.

Последовательная – выход одной функции служит исходными данными для другой функции.

Информационная – считается функция обрабатывающая одни и те же данные.

Процедурно связанные – функции или данные, которые являются частями одного процесса.

Временно связанные – функции выполняются параллельно или в течении определённого периода времени.

Логическая связь – функции базируются на объединении данных или функций в одну логическую группу.

Случайная – связь между элементами мала или отсутствует.

При хорошо продуманной декомпозиции модули верхних уровней иерархии имеют функциональную или последовательную связанность функции данных. Например модули содержащие описании класса при ООП характеризуется информационной связанностью методов и функциональной связанностью данных. Кроме этого для улучшения технологических характеристик используют разделение тела модуля на интерфейсную часть и область реализации. Интерфейсная часть содержит совокупность объявления ресурсов (заголовков подпрограмм, имён переменных, типов, классов и т.д.) который данный модуль предоставляет другим модулям. Ресурсы объявления которых в интерфейсных частях отсутствуют из вне не доступны. Область реализации содержит тела подпрограмм и внутренние ресурсы используемые этими подпрограммами. При такой организации модулей изменение реализации не затрагивает интерфейса, что улучшает технологические характеристики модуля.