Доклад о сложности ПО

(Урусов А.О. и Кузин К.В.)

*Врач, строитель и программист поспорили о том, чья профессия древнее.*

*Врач заметил: "В Библии сказано, что Бог создал Еву из ребра Адама. Это мог*

*сделать только хирург, поэтому моя профессия самая древняя в мире". Его*

*перебил строитель: "Однако, как сказано в Книге Бытия, еще раньше Бог сотворил*

*из хаоса сначала небо, а потом — землю. Это было первое и, несомненно,*

*наиболее впечатляющее строительство. Следовательно, дорогой доктор, вы ошибаетесь.*

*Именно моя профессия самая древняя в мире". Услышав это, программист*

*откинулся на спинку кресла, улыбнулся и спросил доверительным тоном: "Ну а кто*

*же, по-вашему, создал хаос?"*

В нашем докладе речь пойдет о сложных программных продуктах в промышленном масштабе. Это тысячи и десятки тысяч строк кода, огромное количество взаимодействующих элементов, скрещивание различных технологий, очень обширные предметные области. Доклад о том, почему такое ПО – сложное и о способах эту сложность свести к минимуму.

ПО – это сложная система, как, н-р, дерево, устройство компьютера, опорно-двигательная система человеческого тела, система образования и прочие. Однако что делает ПО сложным?

1. Самое главное – вы сами. Человеческий мозг устроен так, что максимальное количество вещей, за которыми он способен следить – не более 8-9, а чтобы осознать и закрепить какой-либо факт, мозгу требуется от 2 до 5 секунд. Это обусловлено ограничениями кратковременной памяти. Отсюда вытекают, что человек не способен быстро и легко понять или разработать систему, в которой одновременно взаимодействует множество элементов. Именно поэтому сложные системы разрабаотываются в команде, в процессе проектирования принимают участие несколько архитекторов, которые могут иметь разные точки зрения на то, как должна быть построена та или иная часть системы. Отсюда появляется новая проблема управления командой разработки: необходимо согласование.
2. Вторая проблема такого ПО – слишком широкая предметная область, в которой, порой, плохо разбирается сам заказчик. Также бывают ситуации, когда заказчик и исполнитель по-разному понимают одну и ту же вещь. Кроме исполнителя и заказчика есть еще пользователи. В процессе написания ТЗ и составления первых прототипов заказчик может осознать что-то и поменять свои требования.
3. ПО такого рода стоит дорого и менять его при каких-либо нововведениях экономически невыгодно. ПО стоит долго и юзеры привыкают к нему, становятся зависимыми от этого ПО. Это говорит о том, что необходимо обеспечить гибкось системы изначально и спроектировать ее так, чтобы ей легко было эволюционировать, дополняться без вреда для уже работающего кода.
4. Чаще всего сложное ПО опирается на реальные объекты. Совокупность этих объектов – это дискретная система, а не непрерывная. Их сложно описывать, потому что в дискретных системах все очень закисит друг от друга и небольшие изменения в исходных данных могут сильно повлиять на выходные данные. Состояние приложение описывается набором состояний переменных, а это детерминированная система.
5. Сложному ПО свойственна иерархическая структура с множеством уровней абстракции, каждый из которых в свою очередь может представлять собой другую систему, тоже сложную. Здесь очень важно изначально определить элементарные компоненты, на которых будет работать нижний уровень абстракции. Проблема в том, что выбор этих компонентов – штука субъективная. Наблюдатель произвольно выбирает, какие элементы считать основными.
6. Итак, мы столкнулись с серьезной дилеммой. Сложность систем программного
7. обеспечения возрастает, а способности справиться с этой сложностью остаются
8. ограниченными. Как же выйти из этого логического тупика?

Мы убедились, что ПО – это сложная система. Как же свести сложность к минимуму? Существует несколько методов и рекомендаций о том, как упростить процесс проектирования сложного ПО. Их нужно использовать комплексно.

Фактически все сводится к нескольком принципам построения структуры приложения:

1. Нужно рассмотреть систему с двух точек зрения. Как part-of и как is-a. Проанализироваав эти 2 представления, необходимо уменьшить избыточность, определить слабые и сильные стороны и уделять особое внимания им. Определить наилучший подход.
2. Алгоритмическая и ООП декомпозиция.
3. Модульность,сокрытие информации. Связи внутри коммпонентов сильнее. Общая структура – повторное использование.

Для анализа поможет абстрагирование. Это расширения возможностей человека. Устойчивые промежуточные формы. Поведение целого сложнее, чем поведение суммы его составляющих

Подходы:

Структурный подход сверху-вниз композиционное проектирование

Тем не менее структурный подход не

решает проблем, связанных с абстрагированием данных и сокрытием информации,

а также не обеспечивает средств для организации взаимной совместимости

объектов (concurrency). Структурный метод не допускает масштабирования для создания

очень сложных систем и, как правило, плохо сочетается с объектными и объектно-

ориентированными языками программирования.

Информационное программирование (data-driven design) яснее всего описано

в ранней работе Джексона (Jackson) [31, 32], а также Орра [33]. В этом методе

система рассматривается как отображение входных данных в выходные. Это

обстоятельство непосредственно влияет на структуру программного обеспечения. Как

и структурное проектирование, информационное программирование успешно

применялось при решении многих сложных задач, в частности, для реализации систем

управления информацией, в которых существуют прямые связи между входными

и выходными данными системы и не возникают проблемы, связанные с

быстродействием.

Основной принцип объектно-ориентированного анализа заключается в том, что

модель системы программного обеспечения рассматривается как совокупность

взаимодействующих объектов, а каждый объект — как экземпляр определенного

класса, входящего в определенную иерархию. Объектно-ориентированный анализ

и проектирование отражает топологию языков программирования высокого уровня,

таких как Smalltalk, Object Pascal, C++, Common Lisp Object System (CLOS), Ada,

Eiffel, Python, Visual C# и Java.

"Для

проектирования новой структуры необходим полет фантазии, а также синтез

опыта и знаний, которые художник воплощает на холсте или бумаге. Сначала инженер

должен стать художником и сформулировать свой замысел, а затем

превратиться в ученого и тщательно проанализировать проект, применяя строгий научный

метод"

Задача разработчиков программных систем — создать иллюзию простоты.