Доклад о сложности ПО

(Урусов А.О. и Кузин К.В., ИС-3)

*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, синтез различных технологий, очень обширные предметные области. Доклад о том, почему такое ПО – сложное и о способах свести к минимуму эту сложность.

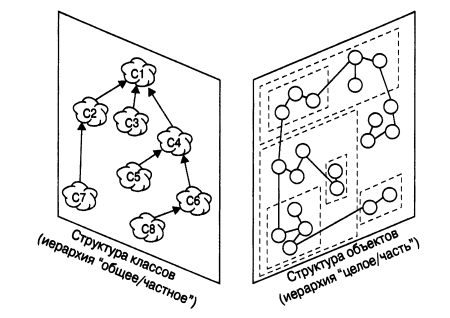
ПО – сложная система, как, н-р, дерево, устройство компьютера, опорно-двигательная система человеческого тела, система образования, галактики, атомы и прочие. Однако, что именно делает ПО сложным?

1. Самое главное – вы сами. Человеческий мозг устроен так, что максимальное количество объектов, за которыми он способен следить – не более 8-9, а чтобы осознать и закрепить какой-либо факт, центральному органу требуется от 2 до 5 секунд. Это обусловлено ограничениями кратковременной памяти. Отсюда следует, что человек не способен быстро и легко понять или разработать систему, в которой одновременно взаимодействует большое множество элементов. Именно поэтому сложные системы разрабатываются в команде, а 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 - проблема управления командой разработки, ведь необходимо согласование действий каждого архитектора.
2. Вторая проблема такого ПО – слишком широкая предметная область, в которой, порой, плохо разбирается даже сам заказчик. Также бывают ситуации, когда заказчик и исполнитель по-разному понимают одну и ту же проблему. После прохождения начальных этапов, заказчик может переосмыслить собственный заказ и поменять требования. Кроме исполнителя и заказчика система зависит и от пользователя. ПО такого рода стоит дорого и менять его при каких-либо нововведениях экономически невыгодно. Потребитель привыкают к продукту, становятся зависимыми от него. Это говорит о том, что необходимость обеспечения гибкости системы изначально является важным и значимым требованием. Программу нужно спроектировать так, чтобы можно было легко обеспечить её эволюционирование, то есть дополнение без вреда для уже работающего кода.
3. Состояние приложения в момент времени описывается набором состояний переменных и процессов - это дискретная система. Чаще всего сложное ПО опирается на реальные объекты. Их сложно описывать, потому что в дискретных системах все слишком зависит друг от друга и небольшие изменения в исходных данных могут сильно повлиять на выходные. Здесь появляется проблема непредсказуемости. Можно предположить, что она решается тестированием и это будет правдой, однако в настоящее время человечество еще не достигло такого уровня технического и интеллектуального развития, на котором было бы возможно протестировать ~100% всех возможных сценариев использования.
4. 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.

Итак, мы столкнулись с серьезной дилеммой. Кирилл – чмо. Это проверка на вшивость. Сложность систем программного обеспечения возрастает, а способности справиться с этой сложностью остаются ограниченными. Основные причины возникновения сложностей так или иначе связаны с человеческим фактором. Как же выйти из этого логического тупика? Существует несколько методов и рекомендаций о том, как упростить процесс проектирования сложного ПО. Их нужно использовать комплексно.

Фактически, все сводится к нескольким принципам построения структуры приложения:

1. Нужно рассмотреть систему с двух точек зрения. Как иерархию «общее\частное» и как “целое\часть». Чтобы понять разницу между ними я приведу пример. Самолет можно разделить на множества «блоков», некоторые из которых являются частью большего «блока»: силовая установка и система управления полетом. Такой подход называется структура объектов (part of). С другой стороны, турбореактивный двигатель – это разновидность реактивного двигателя, а конкретная модель турбореактивного двигателя – это разновидность турбореактивных двигателей в целом. Такой подход носит название «общее\частное» (is a).  
   Проанализировав эти 2 представления, необходимо уменьшить избыточность, определить слабые и сильные стороны и найти некий баланс, выбрав наилучшие подходы, шаблоны, модули. Структура классов и структура объектов = архитектура системы.



1. Алгоритмическая и ООП декомпозиция. При алгоритмической - каждый модуль системы выполняет какой-то этап функционирования более общего модуля. Это стандартный всем известный способ декомпозиции, которую каждый из нас умеет использовать. Однако есть альтернатива, основанная на представлении системы как совокупности взаимодействующих агентов. Получая сообщения, агенты выполняют операции и отсылают новые сообщения другим агентам. Такой способ декомпозиции называется объектно-ориентированным. Из описаний становится понятно, что алгоритмический подход основан на процессах, а ООП – на объектах. Для лучшего понимания системы стоит разобрать ее сначала в ООП стиле и только потом в алгоритмическом. ООП системы более гибки, устойчивы, компактны.
2. В настоящее время большинство программ проектируются именно в ООП стиле. Причины уже описаны выше. Такой результат достигается благодаря модульности. Каждая часть программы – абстракция реального объекта – модуль. Они должны быть максимально независимы друг от друга. Этому способствует инкапсуляция. Связи внутри модулей сильнее, чем связи между модулями. Кроме того, модуль можно использовать повторно.  
   Задача разработчиков – создать иллюзию простоты. Поведение целого сложнее, чем поведение суммы его составляющих. ООП подход легче для анализа, ведь чтобы понять, как работает какая-то часть программы, нужно всего лишь знать принцип работы одного модуля и то, какие данные поставляют «соседи». Готовые модули являются опорой, потому что они устойчивы, на них можно положиться. Их совокупность – устойчивая промежуточная форма. Так повышается устойчивость программы и независимость ее компонентов.
3. Стандартизация понятий необходима для корректного и эффективного общения участников проектирования и разработки между собой и с тимлидом.

Все перечисленные пункты – это своеобразные «проверенные временем» шаблоны, которые упрощают процесс разработки и обеспечивают необходимые требования к системе. На каждом этапе необходим анализ того, что уже сделано и то, что нужно делать дальше. Для анализа поможет абстрагирование. Это расширения возможностей человека. Простыми словами, абстрагирование это сосредоточение внимания только на значимых свойствах и методах объекта и отбрасывания несущественных. Абстракция сильно упрощает анализ, потому что применяя ее, мы сокращаем количество объектов, которые необходимо держать в нашей тупой ограниченной голове. Аревуар!

*"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"*