Сложность

Врач, строитель и программистка спорили о том, чья профессия древнее. Врач заметил: "В Библии сказано, что Бог сотворил Еву из ребра Адама. Такая операция может быть проведена только хирургом, поэтому я по праву могу утверждать, что моя профессия самая древняя в мире". Тут вмешался строитель и сказал: "Но еще раньше в Книге Бытия сказано, что Бог сотворил из хаоса небо и землю. Это было первое и, несомненно, наиболее выдающееся строительство. Поэтому, дорогой доктор, вы не правы. Моя профессия самая древняя в мире". Программистка при этих словах откинулась в кресле и с улыбкой произнесла: "А кто же по-вашему сотворил хаос?"

1. Сложность, присущая программному обеспечению

1.1. Простые и сложные программные системы

Звезда в преддверии коллапса; ребенок, который учится читать; клетки крови, атакующие вирус, - это только некоторые из потрясающе сложных объектов физического мира. Компьютерные программы тоже бывают сложными, однако их сложность совершенно другого рода. Брукс пишет: "Эйнштейн утверждал, что должны существовать простые объяснения природных процессов, так как Бог не действует из каприза или по произволу. У программиста нет такого утешения: сложность, с которой он должен справиться, лежит в самой природе системы"[1].

Мы знаем, что не все программные системы сложны. Существует множество программ, которые задумываются, разрабатываются, сопровождаются и используются одним и тем же человеком. Обычно это начинающий программист или профессионал, работающий изолированно. Мы не хотим сказать, что все такие системы плохо сделаны или, тем более, усомниться в квалификации их создателей. Но такие системы, как правило, имеют очень ограниченную область применения и короткое время жизни. Обычно их лучше заменить новыми, чем пытаться повторно использовать, переделывать или расширять. Разработка подобных программ скорее утомительна, чем сложна, так что изучение этого процесса нас не интересует.

Нас интересует разработка того, что мы будем называть *промышленными программными продуктами.* Они применяются для решения самых разных задач, таких, например, как системы с обратной связью, которые управляют или сами управляются событиями физического мира и для которых ресурсы времени и памяти ограничены; задачи поддержания целостности информации объемом в сотни тысяч записей при параллельном доступе к ней с обновлениями и запросами; системы управления и контроля за реальными процессами (например, диспетчеризация воздушного или железнодорожного транспорта). Системы подобного типа обычно имеют большое время жизни, и большое количество пользователей оказывается в зависимости от их нормального функционирования. В мире промышленных программ мы также встречаем среды разработки,

которые упрощают создание приложений в конкретных областях, и программы, которые имитируют определенные стороны человеческого интеллекта.

Существенная черта промышленной программы - уровень сложности: один разработчик практически не в состоянии охватить все аспекты такой системы. Грубо говоря, сложность промышленных программ превышает возможности человеческого интеллекта. Увы, но сложность, о которой мы говорим, по-видимому, присуща всем большим программных системам. Говоря *"присуща* мы имеем в виду, что эта сложность здесь неизбежна: с ней можно справиться, но избавиться от нее нельзя.

Конечно, среди нас всегда есть гении, которые в одиночку могут выполнить работу группы обычных людей-разработчиков и добиться в своей области успеха, сравнимого с достижениями Франка Ллойда Райта или Леонардо да Винчи. Такие люди нам нужны как архитекторы, которые изобретают новые идиомы, механизмы и основные идеи, используемые затем при разработк е других систем. Однако, как замечает Петерс: "В мире очень мало гениев, и не надо думать, будто в среде программистов их доля выше средней"[2]. Несмотря на то, что все мы чуточку гениальны, в промышленном программировании нельзя постоянно полагаться на божественное вдохновение, которое обязательно поможет нам. Поэтому мы должны рассмотреть более надежные способы конструирования сложных систем. Для лучшего понимания того, чем мы собираемся управлять, сначала ответим на вопрос: почему сложность присуща всем большим программным системам?

1.2. Почему программному обеспечению присуща сложность?

Как говорит Брукс, "сложность программного обеспечения - отнюдь не случайное его свойство"[3]. Сложность вызывается четырьмя основными причинами:

- сложностью реальной предметной области, из которой исходит заказ на разработку;
- трудностью управления процессом разработки;
- необходимостью обеспечить достаточную гибкость программы;
- неудовлетворительными способами описания поведения больших дискретных систем.

Сложность реального мира. Проблемы, которые мы пытаемся решить с помощью программного обеспечения, часто неизбежно содержат сложные элементы, а к соответствующим программам предъявляется множество различных, порой взаимоисключающих требований. Рассмотрим необходимые характеристики электронной системы многомоторного самолета, сотовой телефонной коммутаторной системы и робота. Достаточно трудно понять, даже в общих чертах, как работает каждая такая система. Теперь прибавьте к этому дополнительные требования (часто не формулируемые явно), такие как удобство, производительность, стоимость, выживаемость и надежность! Сложность задачи и порождает ту сложность программного продукта, о которой пишет Брукс.

Эта внешняя сложность обычно возникает из-за "нестыковки"между пользователями системы и ее разработчиками: пользователи с трудом могут объяснить в форме, понятной разработчикам, что на самом деле нужно сделать. Бывают случаи, когда пользователь лишь смутно представляет, что ему нужно от будущей программной системы. Это в основном происходит не из-за ошибок с той или иной стороны; просто каждая из групп специализируется в своей области, и ей недостает знаний партнера. У пользователей и разработчиков разные взгляды на сущность проблемы, и они делают различные выводы о возможных путях ее решения. На самом деле, даже если пользователь точно знает, что ему нужно, мы с трудом можем однозначно

зафиксировать все его требования. Обычно они отражены на многих страницах текста, "разбавленных"немногими рисунками. Такие документы трудно поддаются пониманию, они открыты для различных интерпретаций и часто содержат элементы, относящиеся скорее к дизайну, чем к необходимым требованиям разработки.

Дополнительные сложности возникают в результате изменений требований к программной системе уже в процессе разработки. В основном требования корректируются из-за того, что само осуществление программного проекта часто изменяет проблему. Рассмотрение первых результатов - схем, прототипов, - и использование системы после того, как она разработана и установлена, заставляют пользователей лучше понять и отчетливей сформулировать то, что им действительно нужно. В то же время этот процесс повышает квалификацию разработчиков в предметной области и позволяет им задавать более осмысленные вопросы, которые проясняют темные места в проектируемой системе.

Большая программная система - это крупное капиталовложение, и мы не можем позволить себе выкидывать сделанное при каждом изменении внешних требований. Тем не менее даже большие системы имеют тенденцию к эволюции в процессе их использования: следовательно, встает задача о том, что часто неправильно называют *сопровождением программного обеспечения*. Чтобы быть более точными, введем несколько терминов:

- под *сопровождением* понимается устранение ошибок;
- под *эволюцией* - внесение изменений в систему в ответ на изменившиеся требования к ней;
- под *сохранением* - использование всех возможных и невозможных способов для поддержания жизни в дряхлой и распадающейся на части системе.

К сожалению, опыт показывает, что существенный процент затрат на разработку программных систем тратится именно на сохранение.

Трудности управления процессом разработки. Основная задача разработчиков состоит в создании иллюзии простоты, в защите пользователей от сложности описываемого предмета или процесса. Размер исходных текстов программной системы отнюдь не входит в число ее главных достоинств, поэтому мы стараемся делать исходные тексты более компактными, изобретая хитроумные и мощные методы, а также используя среды разработки уже существующих проектов и программ. Однако новые требования для каждой новой системы неизбежны, а они приводят к необходимости либо создавать много программ "с нуля либо пытаться по-новому использовать существующие. Всего 20 лет назад программы объемом в несколько тысяч строк на ассемблере выходили за пределы наших возможностей. Сегодня обычными стали программные системы, размер которых исчисляется десятками тысяч или даже миллионами строк на языках высокого уровня. Ни один человек никогда не сможет полностью понять такую систему. Даже если мы правильно разложим ее на составные части, мы все равно получим сотни, а иногда и тысячи отдельных модулей. Поэтому такой объем работ потребует привлечения команды разработчиков, в идеале как можно меньшей по численности. Но какой бы она ни была, всегда будут возникать значительные трудности, связанные с организацией коллективной разработки. Чем больше разработчиков, тем сложнее связи между ними и тем сложнее координация, особенно если участники работ географически удалены друг от друга, что типично в случае очень больших проектов. Таким образом, при коллективном выполнении проекта главной задачей руководства является поддержание единства и целостности разработки.

Гибкость программного обеспечения. Домостроительная компания обычно не имеет собственного лесхоза, который бы ей поставлял лес для пиломатериалов; совершенно необычно,

чтобы монтажная фирма соорудила свой завод для изготовления стальных балок под будущее здание. Однако в программной индустрии такая практика - дело обычное. Программирование обладает предельной гибкостью, и разработчик может сам обеспечить себя всеми необходимыми элементами, относящимися к любому уровню абстракции. Такая гибкость чрезвычайно соблазнительна. Она заставляет разработчика создавать своими силами все базовые строительные блоки будущей конструкции, из которых составляются элементы более высоких уровней абстракции. В отличие от строительной индустрии, где существуют единые стандарты на многие конструктивные элементы и качество материалов, в программной индустрии таких стандартов почти нет. Поэтому программные разработки остаются очень трудоемким делом.

Проблема описания поведения больших дискретных систем. Когда мы кидаем вверх мяч, мы можем достоверно предсказать его траекторию, потому что знаем, что в нормальных условиях здесь действуют известные физические законы. Мы бы очень удивились, если бы, кинув мяч с чуть большей скоростью, увидели, что он на середине пути неожиданно остановился и резко изменил направление движения. [Даже простые непрерывные системы могут иметь сложное поведение ввиду наличия хаоса. Хаос привносит случайность, исключающую точное предсказание будущего состояния системы. Например, зная начальное положение двух капель воды в потоке, мы не можем точно предсказать, на каком расстоянии друг от друга они окажутся по прошествии некоторого времени. Хаос проявляется в таких различных системах, как атмосферные процессы, химические реакции, биологические системы и даже компьютерные сети. К счастью, скрытый порядок, по-видимому, есть во всех хаотических системах, в виде так называемых *аттракторов*]. В недостаточно отлаженной программе моделирования полета мяча такая ситуация легко может возникнуть.

1.3. Последствия неограниченной сложности

"Чем сложнее система, тем легче ее полностью развалить"[5]. Строитель едва ли согласится расширить фундамент уже построенного 100-этажного здания. Это не просто дорого: делать такие вещи значит напрашиваться на неприятности. Но что удивительно, пользователи программных систем, не задумываясь, ставят подобные задачи перед разработчиками. Это, утверждают они, всего лишь технический вопрос для программистов.

Наше неумение создавать сложные программные системы проявляется в проектах, которые выходят за рамки установленных сроков и бюджетов и к тому же не соответствуют начальным требованиям. Мы часто называем это *кризисом программного обеспечения,* но, честно говоря, недомогание, которое тянется так долго, становится нормой. К сожалению, этот кризис приводит к разбазариванию человеческих ресурсов - самого драгоценного товара - и к существенному ограничению возможностей создания новых продуктов. Сейчас просто не хватает хороших программистов, чтобы обеспечить всех пользователей нужными программами. Более того, существенный процент персонала, занятого разработками, в любой организации часто должен заниматься сопровождением и сохранением устаревших программ. С учетом прямого и косвенного вклада индустрии программного обеспечения в развитие экономики большинства ведущих стран, нельзя позволить, чтобы существующая ситуация осталась без изменений.

Как мы можем изменить положение дел? Так как проблема возникает в результате сложности структуры программных продуктов, мы предлагаем сначала рассмотреть способы работы со сложными структурами в других областях. В самом деле, можно привести множество примеров успешно функционирующих сложных систем. Некоторые из них созданы человеком, например: космический челнок Space Shuttle, туннель под Ла-Маншем, большие фирмы типа Microsoft или General Electric. В природе существуют еще более сложные системы, например система кровообращения у человека или растение.

2. Структура сложных систем

2.1. Примеры сложных систем

Структура персонального компьютера.Персональный компьютер (ПК) - прибор умеренной сложности. Большинство ПК состоит из одних и тех же основных элементов: системной платы, монитора, клавиатуры и устройства внешней памяти какого-либо типа (гибкого или жесткого диска). Мы можем взять любую из этих частей и разложить ее в свою очередь на составляющие. Системная плата, например, содержит оперативную память, центральный процессор (ЦП) и шину, к которой подключены периферийные устройства. Каждую из этих частей можно также разложить на составляющие: ЦП состоит из регистров и схем управления, которые сами состоят из еще более простых деталей: диодов, транзисторов и т.д.

Это пример сложной иерархической системы. Персональный компьютер нормально работает благодаря четкому совместному функционированию всех его составных частей. Вместе эти части образуют логическое целое. Мы можем понять, как работает компьютер, только потому, что можем рассматривать отдельно каждую его составляющую. Таким образом, можно изучать устройства монитора и жесткого диска независимо друг от друга. Аналогично можно изучать арифметическую часть ЦП, не рассматривая при этом подсистему памяти.

Дело не только в том, что сложная система ПК иерархична, но в том, что уровни этой иерархии представляют различные уровни абстракции, причем один надстроен над другим и каждый может быть рассмотрен (понят) отдельно. На каждом уровне абстракции мы находим набор устройств, которые совместно обеспечивают некоторые функции более высокого уровня, и выбираем уровень абстракции, исходя из наших специфических потребностей. Например, пытаясь исследовать проблему синхронизации обращений к памяти, можно оставаться на уровн е логических элементов компьютера, но этот уровень абстракции не подходит при поиске ошибки в прикладной программе, работающей с электронными таблицами.

Структура растений и животных. Ботаник пытается понять сходство и различия растений, изучая их морфологию, то есть форму и структуру. Растения - это сложные многоклеточные организмы. В результате совместной деятельности различных органов растений происходят такие сложные типы поведения, как фотосинтез и всасывание влаги.

Растение состоит из трех основных частей: корни, стебли и листья. Каждая из них имеет свою особую структуру. Корень, например, состоит из корневых отростков, корневых волосков, верхушки корня и т.д. Рассматривая срез листа, мы видим его эпидермис, мезофилл и сосудистую ткань. Каждая из этих структур, в свою очередь, представляет собой набор клеток. Внутри каждой клетки можно выделить следующий уровень, который включает хлоропласт, ядро и т.д. Так же, как у компьютера, части растения образуют иерархию, каждый уровень которой обладает собственной независимой сложностью.

Все части на одном уровне абстракции взаимодействуют вполне определенным образом. Например, на высшем уровне абстракции, корни отвечают за поглощение из почвы воды и минеральных веществ. Корни взаимодействуют со стеблями, которые передают эти вещества листьям. Листья в свою очередь используют воду и минеральные вещества, доставляемые стеблями, и производят при помощи фотосинтеза необходимые элементы.

Для каждого уровня абстракции всегда четко разграничено "внешнее"и "внутреннее". Например, можно установить, что части листа совместно обеспечивают функционирование листа в целом и очень слабо взаимодействуют или вообще прямо не взаимодействуют с элементами корней. Проще говоря, существует четкое разделение функций различных уровней абстракции.

В компьютере транзисторы используются как в схеме ЦП, так и жесткого диска. Аналогично этому большое число "унифицированных элементов"имеется во всех частях растения. Так Создатель достигал экономии средств выражения. Например, клетки служат основными строительными блоками всех структур растения; корни, стебли и листья растения состоят из клеток. И хотя любой из этих исходных элементов действительно является клеткой, существует огромное количество разнообразных клеток. Есть клетки, содержащие и не содержащие хлоропласт, клетки с оболочкой, проницаемой и непроницаемой для воды, и даже живые и умершие клетки.

При изучении морфологии растения мы не выделяем в нем отдельные части, отвечающие за отдельные фазы единого процесса, например, фотосинтеза. Фактически не существует централизованных частей, которые непосредственно координируют деятельность более низких уровней. Вместо этого мы находим отдельные части, которые действуют как независимые посредники, каждый из которых ведет себя достаточно сложно и при этом согласованно с более высокими уровнями. Только благодаря совместным действиям большого числа посредников образуется более высокий уровень функционирования растения. Наука о сложности называет это *возникающим поведением.* Поведение целого сложнее, чем поведение суммы его составляющих [6].

Обратимся к зоологии. Многоклеточные животные, как и растения, имеют иерархическую структуру: клетки формируют ткани, ткани работают вместе как органы, группы органов определяют систему (например, пищеварительную) и так далее. Мы снова вынуждены отметить присущую Создателю экономность выражения: основной строительный блок всех растений и животных - клетка. Естественно, между клетками растений и животных существуют различия. Клетки растения, например, заключены в жесткую целлюлозную оболочку в отличие от клеток животных. Но, несмотря на эти различия, обе указанные структуры, несомненно, являются клетками. Это пример общности в разных сферах.

Жизнь растений и животных поддерживает значительное число механизмов надклеточного уровня, то есть более высокого уровня абстракции. И растения, и животные используют сосудистую систему для транспортировки внутри организма питательных веществ. И у тех, и у других может существовать различие полов внутри одного вида.

2.2. Пять признаков сложной системы

Исходя из такого способа изучения, можно вывести пять общих признаков любой сложной системы. Основываясь на работе Саймона и Эндо, Куртуа предлагает следующее наблюдение [7]:

1. *"Сложные системы часто являются иерархическими и состоят из взаимозависимых подсистем, которые в свою очередь также могут быть разделены на подсистемы, и т.д., вплоть до самого низкого уровням."*

2. *Выбор, какие компоненты в данной системе считаются элементарными, относительно произволен и в большой степени оставляется на усмотрение исследователя.*

3. *"Внутрикомпонентная связь обычно сильнее, чем связь между компонентами. Это обстоятельство позволяет отделять "высокочастотные"взаимодействия внутри компонентов от "низкочастотной" динамики взаимодействия между компонентами"[10].*

4. *"Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных"[11].*

5. *"Любая работающая сложная система является результатом развития работавшей более простой системы... Сложная система, спроектированная "с нуля никогда не заработает.*
Следует начинать с работающей простой системы".

Список литературы

[1] Brooks, F. April 1987. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer vol.20(4), p.12.

[2] Peters, L. 1981. Software Design. New York, NY: Yourdon Press, p.22.

[3] Brooks. No Silver Bullet, p.11.

[4] Parnas, D. July 1985. Software Aspects ofStrategic Defense Systems. Victoria, Canada: University of Victoria. Report DCS-47-IR.

[5] Peter, L. 1986. The Peter Pyramid. New York, NY: William Morrow, p.153

[6] Waldrop, M. 1992. Complexity: The Emerging Science at the Edge of Order and Chaos. New-York, NY: Simon and Schuster.

[7] Courtois, P. June 1985. On Time and Space Decomposition of Complex Structures. Communications of the ACM vol.28(6), p.596.

[8] Simon, H. 1982. The Sciences of the Artificial. Cambridge, MA: The MIT Press, p.218

[9] Rechtin, E. October 1992. The Art of Systems Architecting. IEEE Spectrum, vol.29(10), p.66.

[10] Simon. Sciences, p.217.

[11] Ibid, p.221.