ТЕХНОЛОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Тиунова Анна Евгеньевна

РЕИНЖИНИРИНГ

Реинжиниринг программного обеспечения – это процесс создания приложения на основе имеющейся работающей системы, но с изменением каких-либо ее характеристик: вычислительной платформы, базы данных, программной архитектуры и т. п.

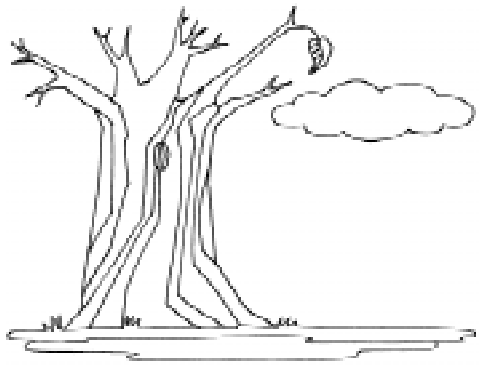
Зачем нужен реинжиниринг? Во многих крупных компаниях (таких, как банки, страховые компании, промышленные предприятия и т. д.) очень остро стоит проблема поддержки приложений, автоматизирующих их деятельность. Некоторые из таких приложений написаны более десяти лет назад, их авторов уже не найти, а документация или утеряна, или устарела. Многие из них написаны на устаревших языках программирования (Кобол, ПЛ/1 и др.), специалистов по которым найти трудно. Тем не менее, эти программы по-прежнему работают, и в них необходимо отражать изменения, происходящие в бизнес-процессах компании: иногда нужно сменить проценты, по которым рассчитывается страховая премия, иногда добавляются новые виды деятельности, иногда изменяется законодательство.

Как и с любой программой, существующей продолжительное время, возникает желание переписать все приложение, с использованием современных технологий программирования, чтобы существенно облегчить ее дальнейшее сопровождение. Однако написание новой системы «с нуля» требует огромных ресурсов, и редко является практически реальным. Реинжиниринг призван облегчить процесс создания новой системы за счет использования исходных текстов старой.

Мы занимаемся реинжинирингом уже более десяти лет. За это время появилось много разных направлений, выявились основные проблемы и сменился основной акцент.

Первоначально, основываясь на богатом опыте создания различных компиляторов и CASE-систем, использующих генерацию промежуточной программы на языке высокого уровня вместо прямой компиляции объектного кода, казалось что реинжиниринг можно полностью автоматизировать: пишем «компилятор», генерирующий по программе на старом языке такую же программу на современном языке, запускаем его на все старое приложение и все готово.

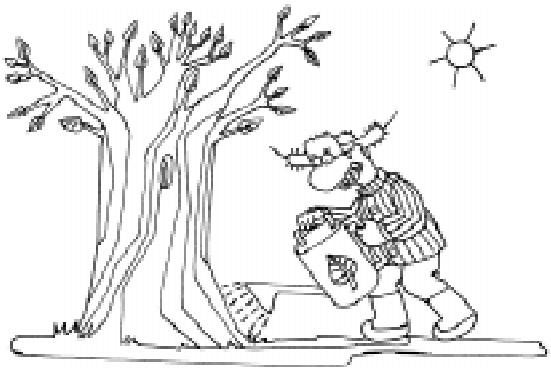
На практике все оказывается значительно сложнее. Языки программирования, да и вообще подход к программированию сильно изменились.



Некоторые из таких приложений написаны более десяти лет назад, их авторов уже не найти, а документация или утеряна, или устарела…

...возникает желание переписать все приложение, с использованием современных технологий программирования...

Обычная программа на Коболе – это «простыня» в несколько тысяч строк, с запутанной логикой исполнения, без явно выраженных процедур, в которой все переменные глобальны. Одна и та же программа может делать все сразу – выводить на терминал экранную форму, обрабатывать ввод пользователя, производить расчеты (скажем, все той же страховой премии на основе заполненной анкеты) и записывать все в базу данных. Обычная программа на Java – это объектно-ориентированная программа, в которой данные собраны в логически законченные классы, а обработка данных распределена между методами этих классов. При этом бизнес-логика отделена как от интерфейсной части программы, так и от подробностей организации базы данных (трехуровневая архитектура).



Реинжиниринг призван облегчить процесс создания новой системы за счет использования исходных текстов старой...

Возможности языков программирования различны – появились новые конструкции (не во всех языках были операторы ветвления типа switch и даже циклы), исчезли некоторые старые (сложные операции с массивами или строками, goto). Есть существенные отличия в типах данных и даже результатах арифметических операций (сумма 99 + 1 при определенных обстоятельствах может оказаться равной 0).

Из-за всех этих различий после прямолинейной генерации новой программы, которая в точности эмулирует поведение старой, получается код, разобраться в котором часто сложно, а сопровождать потом еще сложнее, чем старую программу.

Однако это не означает, что от идеи реинжиниринга нужно отказаться. Во-первых, можно совершенствовать генерацию кода, например, научить ее распознавать циклы, организованные с помощью goto, или локализовать хотя бы часть переменных. Во-вторых, можно привлечь человека, который может помочь сделать генерируемый код лучше, с учетом конкретных особенностей конкретной системы. Скажем, если эксперт знает, что ситуация 99 + 1 никогда в данной программе не наступает, потому что реальные значения переменной не превышают 50, то он может «подсказать» это системе, и тогда совершенно не нужно генерировать «защитный» код, а можно оставить обычное присваивание *A* + 1.

На самом деле это очень важный вывод из первых опытов реинжиниринга – реинжиниринг не может быть полностью автоматическим, участие человека в этом процессе обязательно. И задача автоматизированных средств реинжиниринга – минимизировать степень этого участия.

Кроме усовершенствования самой генерации текста программ, стали придумывать способы сменить технологию реинжиниринга. Появилось желание попытаться извлечь из старой программы, которая делает все сразу, ее содержательную часть, извлечь сущность из того, что программа делает, – бизнес-логику. Во-первых, это дало толчок новому направлению по извлечению бизнес-логики, а, во-вторых, пришла идея, что

трансформировать нужно не исходную программу целиком, а отдельные бизнес-правила, извлеченные из нее. Затем уже из этих компонент составить новую систему в современной модульной (объектно-ориентированной) архитектуре, где каждая компонента большого программного комплекса решает свою изолированную задачу.

Код, отвечающий за обмен с терминалом, а иногда и с базой данных, обычно сильно специфичен в разных системах, и его проще написать заново под конкретную целевую платформу.

Направление по извлечению бизнес-правил сейчас очень активно развивается. Сформулировано несколько алгоритмов, позволяющих автоматически извлекать некоторые правила (компоненты) из программ.

Один из вариантов – извлечение вычислительной логики. У вас есть программа, которая вычисляет несколько величин, а вас интересует алгоритм вычисления одной из них. Тогда можно извлечь из программы всю логику, относящуюся непосредственно к вычислению этой переменной, и отсечь все остальное.

Если в исходной программе есть какой-то полезный фрагмент, часто повторяющийся в этой и других программах приложения, можно оформить его в отдельную подпрограмму и заменить раскопированный код на ее вызовы. Этот метод называется структурным.

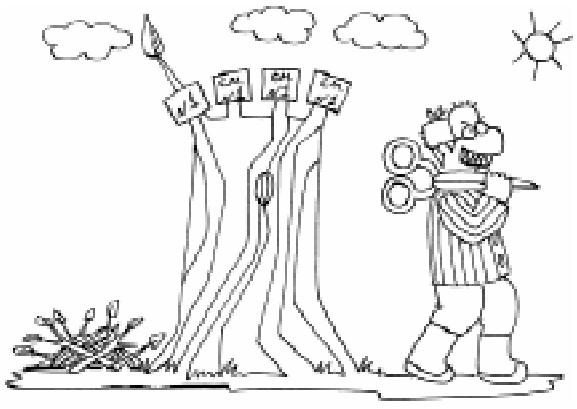
Существует также метод извлечения бизнес-правил, основанный на анализе значений определенных переменных. Предположим, что программа читает данные из входного файла, а читаемые записи бывают пяти типов, каждый из которых обрабатывается по-разному. Можно разделить такую программу на 5 отдельных, каждая из которых обрабатывает записи только одного типа. Этот метод также полезен для удаления неиспользуемого кода, когда известно, что, например, один из этих пяти типов больше уже не встречается на практике. Можно «вычистить» из программы все, относящееся к обработке этого ненужного больше типа, и оставить всю логику, связанную с оставшимися четырьмя.

Достаточно популярной задачей, для решения которой применяется извлечение бизнес-правил, является добавление к уже существующей диалоговой системе, работающей на мейнфрейме, Интернет-доступа. Необходимо выделить из программы, делающей все сразу, бизнес-логику, так чтобы к ней можно было обратиться из нового интерфейса, но и старый остался бы работоспособным. Создание нового интерфейса в большой степени автоматизируется, html-страницы генерируются на основе существующих экранных форм.

Обеспечение доступа с персональных компьютеров к приложению, работающему на мейнфрейме, с использованием современного графического интерфейса, значительно облегчает работу пользователей (обычный для мейнфрейма интерфейс – это черно-зеленый текстовый терминал) и используется как альтернатива полному реинжинирингу. Это позволяет не только сократить объем работы по переводу программ на новые языки, но и исключает необходимость преобразования базы данных, что особенно важно, если накоплен очень большой объем реальных данных, переносить которые сложно.

Извлечение бизнес-правил используется и с целью восстановления документации по приложению.

Однако оказалось, что научиться извлекать корректные бизнес-правила по тому или иному принципу – это только полови-



...полезный фрагмент, часто повторяющийся в ... программах приложения, можно оформить его в отдельную подпрограмму и заменить раскопированный код на ее вызовы...

на задачи. Нужно еще уметь найти ту самую переменную или фрагмент кода, которые могут быть интересны для компонентизации. Поэтому стало активно развиваться направление, которое на первый взгляд может показаться и не относящимся к реинжинирингу – создание средств анализа и визуализации устаревших приложений. Эти же средства анализа полезны и в повседневной деятельности программистов, сопровождающих старые приложения. Самое распространенное средство, которым пользуются программисты, – текстовый редактор, в котором есть поиск подстроки. Но программа состоит из многих файлов, а сопровождаемое приложение может содержать тысячи программ, и очень сложно что-нибудь изменять, пользуясь только текстовым редактором, без опасности испортить что-нибудь в другой части системы. Поэтому существует потребность в анализирующих средствах, повышающих эффективность и надежность работы.

Анализ приложений бывает разноуровневый. Начальная стадия – это инвентаризация, определение набора файлов, из которых состоит приложение. Файлы часто разбросаны по нескольким библиотекам, поэтому не все сразу удается их собрать, и в то же время в библиотеках могут скопиться уже не используемые файлы.

Следующий уровень – выявление взаимосвязей в приложении, опять же на уровне крупных объектов: например, какая программа работает с какой экранной формой, какие программы к каким таблицам в базе данных обращаются, какие программы вызываются из других.

...выявление взаимосвязей в приложении...

Еще один уровень – исследование конкретной программы. Имея информацию верхнего уровня и зная задачу (например, нужно добавить новое поле на экранную форму), мы можем изолировать ту часть приложения, которая нам интересна, и далее уже детально изучать «внутренности» этого подмножества. Наиболее часто интересующие программиста или аналитика вопросы: где эта переменная меняет свое значение? Как управление попадает в этот фрагмент программы? Где находятся операторы работы с внешними данными – таблицами, экранными формами и т. п.? Есть ли в программе неиспользуемые переменные или операторы, которые никогда не выполняются?

Наибольший интерес представляют задачи, связанные со взаимным влиянием переменных. Допустим, вас интересует некоторая переменная, которая используется в программе, и вы хотите узнать, что влияет на вычисление ее значения. Задача, сходная с извлечением вычислительных бизнес-правил, однако результат представлен по-другому. Вместо извлечения бизнес-правила, которое само является программой, пусть и меньшего размера, используются средства визуализации исходной программы. Выбрав начальную точку, можно по шагам исследовать взаимосвязи между переменными с учетом порядка выполнения операторов. Исследовать можно оба направления: либо «предков», либо «потомков» нашей переменной.

Для облегчения сопровождения приложений развивается и направление, находящееся на стыке анализа и компонентизации, – «улучшение» программ. Часто можно преобразовать программу так, чтобы результат ее работы не изменился, но исходный код стал более понятным. Сюда относится удаление мертвого кода и неиспользуемых переменных, уменьшение количества goto, форматирование исходного кода (приведение к стандартному стилю) и т. п. Подобные улучшения очень полезно проводить и перед «настоящим» реинжинирингом – переводом на другую платформу, ведь чем

лучше написана программа, чем меньше в ней «плохих» конструкций, тем лучше получится автоматически генерируемый код на новом языке.

Одним из практических применений реинжиниринга является переход с так называемых горящих платформ. В какой-то момент владельцы прикладной системы решают, что они не хотят больше работать на старой платформе – например, она может больше не сопровождаться производителем. В таких случаях необходим полный комплекс реинжиниринга – и бизнес-логика, и экранные формы, и базы данных. К горящим платформам относятся не только «экзотические» вычислительные машины, но и некоторые CASE-средства.

Перенос баз данных может быть и самостоятельной содержательной задачей. Часто осуществляется перевод иерархических (например, IMS) и сетевых (например, IDMS) баз данных в реляционные. При этом нужно не только разработать эквивалентную структуру новой базы данных, но и переработать операторы работы с базой данных, поскольку идеология реляционной базы отличается от идеологии исходной. К смене базы данных можно отнести и преобразование файлов с индексно-последовательным доступом в реляционные базы данных.

Одним из самых известных применений реинжиниринга было решение проблемы 2000 года. Наибольшую трудность представлял поиск – идентифицировать все переменные, так или иначе связанные с датами.

Авторы системы поиска знали, что в любой компании существуют стандарты по написанию программ, в том числе и на имена переменных. Поэтому поиск обычно начинался с перебора известных стандартных образцов. Скажем, все переменные, содержащие в имени слово «year» или буквы «yr» считаются подозрительными на дату. Таким образом формировалось начальное множество «подозрительных» переменных. После этого вступала в силу вторая половина анализа, связанная с потоками данных. Отслеживались все связи переменных, «подозрительными» становились все переменные, которые зависели от начального множества, дальше процесс многократно повторялся. Необходимо было отслеживать и все внешние связи программ – вызовы подпрограмм, работу с базами данных. Одна программа могла записать в колонку таблицы дату, а другая программа могла прочитать ее, уже не называя переменную датой.

Обнаруженные даты изменяли двумя способами. Либо добавляли еще две цифры во все обнаруженные «плохие» переменные, либо использовали определенное граничное число и добавляли в программу сравнения с этим числом для определения того или иного столетия по двум цифрам года. И то, и другое можно было достаточно успешно делать автоматически.

Похожими являлись и задачи, связанные с введением единой европейской валюты и с переходом на девятитизначные почтовые индексы взамен пятизначных в США.

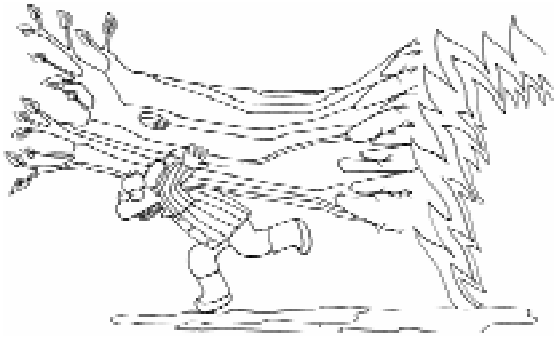
Направления и приложения реинжиниринга:

– полный перевод приложения на новую платформу (вычислительную машину, язык программирования и т. п.);

– смена базы данных или какой-либо другой компоненты приложения;

– перевод приложения с одной разновидности языка программирования на другую (обычно при смене вычислительной машины);

– перевод из CASE-среды в традиционный код программы (обычно связано с



...является переход с так называемых горящих платформ...

прекращением поддержки CASE-средства его производителями);

– использование анализирующих средств просто для облегчения сопровождения существующего приложения – изучения программ с целью исправления ошибок/внесения изменений, оценки трудоемкости изменений на основе списка «задетых» компонент и т. п.;

– clean & shine: удаление из существующей системы неиспользуемых более компонент, типов обрабатываемых данных и т.п. (связано с большим временем жизни системы и изменениями бизнес-процессов), здесь в основном используется «простое» удаление мертвого кода и неиспользуемых данных и domain-based компонентизация (удаление неиспользуемого кода с учетом значений переменных); garbage in – garbage out;

– извлечение бизнес-правил с целью восстановления актуальной документации по системе (которая обычно утеряна, устарела и т. п.);

– проектирование и создание новой системы с той же функциональностью, но в существенно другой программной архитектуре на основе извлеченных бизнес-правил.

Практически все направления могут быть в той или иной степени автоматизированы, но полностью избавиться от ручного труда нельзя, автоматизированные средства лишь уменьшают процент требуемого ручного труда.

Тиунова Анна Евгеньевна,
ЗАО «Ланит-Терком», департамент реинжиниринга, начальник отдела разработки средств реинжиниринга.