Размышления о программировании

От Аристотеля к Витгенштейну

Thinking About Programming

From Aristotle to Wittgenstein

Abstract. *The report presents abstracts on general issues of software engineering. Software development considered as a new kind of human activity, which is mistakenly attributed to engineering. Engineering based on applicable laws of mathematics, physics and chemistry to design new products. In software development has not yet opened its Newton's laws, Lagrange's equations, or at least strength of materials that helped to design and prove the correctness of the new architecture of a nontrivial software system. Programming is humanities discipline and the serious moves in its theoretical basis can only be achieved by using the achievements of the humanities: philosophy, psychology, linguistics, semiotics, etc.*

The report examines constraints of the object paradigm, widely used in the construction of software systems. The constraints add excess complexity to the software architecture. The report discusses a problem of creating a universal syntax of domain-specific language (DSL). The author introduces a hypothesis that attempts to solve this problem that have been made in the last decade, are not widely used because these approaches have tried to simulate the static world of Aristotle's objects, it is possible that the basis for a universal syntax of DSL can be a dynamic world of Wittgenstein's interactions and the categorical approach, reflecting the fundamental features of human thinking.

Key words: *software engineering; object paradigm; domain-specific language; ontology, Aristotle; Wittgenstein; category theory; philosophy; psychology; linguistics; semiotics.*

Аннотация. *В докладе представлены тезисы по общим вопросам технологии программирования. Программирование рассматривается как новый вид человеческой деятельности, которая по ошибке отнесена к инженерии. Инженерия – это там, где применяются законы естественных наук: математики, физики, химии при конструировании новых продуктов. В разработке ПО еще не открыты свои законы Ньютона, уравнения Лагранжа или хотя бы сопромат, которые помогли бы спроектировать и доказать правильность архитектуры новой нетривиальной программной системы. Программирование скорее гуманитарная дисциплина и серьезных продвижений в ее теоретическом основании можно добиться лишь, используя достижения гуманитарных наук: философии, психологии, лингвистики, семиотики и др.*

В докладе анализируются ограничения объектно-ориентированной парадигмы (ООП), широко применяемой при построении программных систем, которая привносит в их архитектуру избыточную сложность. Рассматривается задача о создании универсального синтаксиса предметно-ориентированного языка (DSL). Выдвигается гипотеза о том, что попытки решения этой задачи, которые предпринимались в последнее десятилетие, не получили широкого распространения потому, что в этих подходах пытались моделировать статический мир объектов Аристотеля и вполне возможно, что основой для универсального синтаксиса DSL может стать динамический мир взаимодействий Витгенштейна и категорный подход, отражающие фундаментальные особенности человеческого мышления.

Ключевые слова*: разработка програмного обеспечения; объектная парадигма; предметно-ориентированный язык; онтология; Аристотель; Витгенштейн; теория категорий; философия; психология; лингвистика; семиотика.*

I. Предисловие

Мне не нужен язык, который позволяет создавать хорошие программы. Я ищу язык, на котором нельзя будет написать плохую программу.

Развитие информатики представляется рекой, которая рождается далеко в прошлом (Евклид, III век до н.э.; Вавилон, XIX век до н.э.; раньше?) из едва заметных ручейков первых алгоритмических вычислений. Неспешно двигаясь по истории, ручейки объединяются в реку, которая, неся свои воды через века, вбирает в себя притоки из смежных дисциплин, накапливает величественность и мощь и, наконец, срывается ниагарским водопадом из второго в третье тысячелетие, превращаясь в стремительный бурлящий поток, который захватывает и несет с собой из прошлого в будущее миллионы людей.

Броуновской частице, которую то бросает на стремнину с турбулентным течением, то в застоявшееся болото, то на мелководье, то в омут, мир информационных технологий видится загадочным, изменчивым и непредсказуемым. Однако, радость постоянного движения, героического преодоления трудностей, бешеного вращения калейдоскопа новых впечатлений, со временем сменяется тоской, томлением духа и непреодолимой потребностью на мгновение приподняться над суетой, взглянуть со стороны на это бешено бурлящий поток, и попытаться разглядеть, если не общее направление бурной реки, то хотя бы, ближайший поворот той протоки, в которой барахтаешься.

II. Введение

«Если ученый не умеет популярно объяснить восьмилетнему ребенку, чем он занимается, значит, он шарлатан»

Курт Воннегут, "Колыбель для кошки", 1963

I. «Серебряной пули нет», сказал Ф.Брукс еще в прошлом веке и надолго остудил пыл благородных рыцарей от программирования в борьбе с Драконом сложности. С тех пор его Величество Дракон служит оправданием тому, что пользователям вместо одних плохо работающих программ, навязываются все новые и новые, которые не намного лучше, но пожирают все больше ресурсов процессоров и памяти компьютеров («А что? Пипл хавает!» (с) Б.Титамир). В угоду Дракону, выращена целая армия программистов - «code & fix» («делай и переделывай»). «Не умением, а числом!», «Рефакторинг – наше все!» [1], «Программировать могут даже члены республиканской партии!» [2] - написано на их знаменах. Чтобы прокормить Дракона, каждый день они создают гигабайты исходного кода, значительная часть которого никогда не будет востребована пользователями. За прошедшие годы этот отвратительный Дракон разросся и стал «Идеей. Исторической необходимостью. Нашим государственным интересом. Могущественным фактором, оправданием наших объединенных усилий» [3].

II. Я не собираюсь искать очередную «серебряную пулю», но не потому, что ее нет, а потому, что не очень верю в драконов. Драконы, как правило, рождаются в сознании людей, когда они сталкиваются с явлениями, которые не находят объяснения в их предыдущем опыте. Для информатики наличие белых пятен не удивительно потому, что это очень молодая отрасль человеческого знания.

III. Я не очень верю в драконов, зато верю в то, что сложность это понятие субъективное. Одно из определений слова «сложный» - трудный, запутанный. Например, сложная задача, сложная (а может быть точнее запутанная?) программа. То, что для одного человека является трудным и запутанным, для другого может быть легким и ясным. Движение планет в геоцентрической модели Птолемея по циклам и эпициклам с поправками к ним выглядит сложным. Движение в гелиоцентрической модели по эллипсам Кеплера – простым. Если редактировать изображения в битовом формате, получится сложно, а если в графическом редакторе, то просто. Изображение множества Мандельброта может загипнотизировать своей сложностью непосвященного наблюдателя, но на адекватном языке множество Мандельброта описывается крайне просто: «множество точек c на комплексной плоскости, для которых итеративная последовательность $z_0 = 0; z_{n+1} = z_n^2 + c$ не уходит в бесконечность».

IV. В природе нет ничего трудного и запутанного. Нечто может только казаться трудным и запутанным человеку, который это нечто пытается понять. Может быть, программные системы, которые мы разрабатываем, мы сами делаем сложными и запутанными? Почему в ERP системе SAP/R3 более 50 тысяч таблиц, а в самом большом словаре русского языка всего примерно 54 тысячи существительных? Я порой с сожалением чувствую себя шарлатаном, когда пытаюсь объяснить заказчику, почему так сложно добавить к программе простую функциональность, о которой он просит.

V. В этой работе, я не ставлю перед собой каких-либо задач, кроме одной – записать свои не очень связанные, а местами и незавершенные мысли о самых общих вопросах программирования, над которыми я размышляю последние двадцать лет. Время неумолимо и мне его может просто не хватить для того, чтобы собрать все нужные «камни» и сложить из них достаточно прочный фундамент новой парадигмы программирования. Буду удовлетворен, если мои мысли окажутся кому-то полезными, а возможно, и подтолкнут кого-то из коллег отправиться на поиск недостающих «камней».

III. Тезисы

«Всякая задача становится тривиальной, если ее сформулировать в адекватном языке»
И.М.Гельфанд

1. О программах
   - 1.1. *Программа = задача + модель + алгоритм + структура данных*
   - 1.2. Программа создается для того, чтобы решить определенную задачу: обеспечить управление КА при перелете к Фобосу, распознать преступника по фотографии в потоке людей, принять SMS-сообщение с одного мобильного телефона и передать его на другой.
   - 1.3. Модель описывает то, что должна сделать программа для решения поставленной задачи, но не как она это должна сделать. Созданные в мышление человека (ментальные) модели предметной области могут относиться как к явлениям реального мира, например, движение космического объекта в гравитационном поле и атмосфере Земли, так и к идеальным понятиям, таким как интернет-магазин. Модель – это ментальный образ будущего инструмента, который позволит нам решить поставленную задачу.
   - 1.4. Ключевым понятием в определении программы является задача. Как неразумно обсуждать техническую систему в отрыве от задачи, которую она решает, так бессмысленно рассматривать программу вне целей, для которых она создается. Для решения разных задач даже в одной предметной области будут созданы разные модели и разработаны разные программы. Для расчета траекторий межпланетных перелетов, достаточно моделировать гравитационное поле как суперпозицию точечных масс Солнца и планет. Для практически приемлемой точности прогнозирования движения спутников на околоземных орбитах в модели гравитации необходимо учитывать неоднородность поля

Земли в виде разложения по гармоническим функциям или специально подобранной системе точечных масс. А решение отдельных задач глобальной спутниковой навигационной системы (ГЛОНАСС/GPS) невозможно без привлечения моделей времени и пространства из Общей теории относительности.

1.5. Итак, программа – это записанное на понятном некоторому вычислителю языке решение стоящей перед нами задачи.

2. Об объектно-ориентированном подходе

2.1. Сегодня для записи программ создано около 2500 языков программирования. История их развития в чем-то схожа с историей развития человеческой речи. Существует мнение, что содержание первых высказываний человека состояли исключительно из требований помощи, которую бы мог оказать ему другой индивид. «Если бы первые высказывания становящегося человека выразить с помощью нашего, развитого языка, они обязательно содержали бы глаголы в повелительном наклонении (“дай!”, “неси!”, “ломай!”, “режь!”, “бей!”, “поднимай!”, “тяни!” и т. п.)». Причем эти команды сопровождались жестами, которые точно указывали к чему конкретно это действие должно быть применено. Ну, очень похоже на команды императивных языков программирования! Например, команды ассемблеров с точным указанием адресов памяти или регистров.

2.2. Человек вначале научается отличать одну практическую ситуацию, взятую в целом, от другой ситуации. Выделение отдельных элементов этих ситуаций (предметов, над которыми совершаются действия, действий, которые совершаются над предметами) осуществляется позже – по мере того, как в практической деятельности человек все больше знакомился с окружающими его вещами, познавал их свойства и их отношения друг к другу и к самому человеку. Постепенно человек начинает выделять из конкретной ситуации объект действия (в программировании – данные) и само действие (в программировании - функции). Овладение способностью выделять объекты действия было настоящей революцией в умственном развитии первобытного чело века. А это уже очень похоже на объектно-ориентированный подход, применяемый в программировании. По крайней мере, в его современной реализации в языках C++, Java, C#.

2.3. При помощи естественного языка человек материализует свои ментальные модели мира, чтобы передать их другому человеку. При помощи ООП программист материализует свои ментальные модели программного продукта, чтобы передать их на исполнение вычислителю. Но действительно ли ООП это главное русло нашей реки? Или это лишь тихая заводь перед крутым поворотом, которая отличается медленным и порой обратным течением воды и которая уже начала заболачиваться?

2.4. Применение ООП к построению ментальных моделей физического мира имеет более двух тысяч лет истории успешного использования, которое берет начало в трудах Аристотеля. В мире Аристотеля существуют только единичные и конкретно определенные вещи с заданным набором свойств и отнесенные к одной и только одной категории.

2.5. Да, теперь мы уже научились говорить не просто «неси», а «неси дрова» или «неси камень». В своем развитии языки программирования остановились на том, что научились различать объекты (дрова и камень), но не научились выделять действия над ними. И с точки зрения языка программирования *men.carry (firewood)* и *men.carry (stone)* будут разными языковыми единицами, если только объекты *firewood* и *stone* не имеют общего предка! Просто реализация этими объектами интерфейса «то, что может носить человек» нас не выручит, поскольку это будет две реализации, а, следовательно, и единиц исходного кода тоже будет две.

2.6. Здесь скрыто одно из основных ограничений ООП, которое делает наши программы сложнее, чем они могли бы быть, заставляет нас использовать костыли в виде паттернов проектирования, чтобы компенсировать врожденную хромоту ООП. Когда мы пытаемся определить, подходит ли нам конкретный дизайн программной системы или нет, мы не можем рассматривать данное решение изолированно. Мы должны рассматривать его с точки зрения разумных предположений о том, как будет использоваться данный дизайн в последствие. Если перевести этот тезис на язык «дров» и «камней», то это будет звучать, примерно, так. Проектируя программные объекты «дрова» и «камень», мы должны быть в курсе планов Господа, по совершенствованию программной системы. А именно, мы должны предполагать, что Он не ограничится созданием дров и камней, а на шестой день сотворения мира создаст человека, который их будет перетаскивать.

2.7. О наследовании. Другое ограничение ООП заключается в том, что каждый объект принадлежит одной и только одной иерархии ("is a") классов, пусть даже с возможностью множественного наследования и имеет раз и навсегда заданный набор свойств. Например, красная роза - это цветок, а цветок - это растение. Это противоречит реальности, в которой объекты могут эволюционировать, приобретать новые свойства, и утрачивать ранее существовавшие. Например, роза может стать товаром, а потом

подарком. Наследником какого класса должна быть роза-подарок? Роза-товар или роза-цветок? Другой пример. Человек рождается с очень ограниченным набором свойств: иметь возраст, вес, рост, пищать, питаться и портить памперсы. Время идет, и он приобретает новые наборы свойств: ученик школы, покупатель, пассажир, солдат, студент, наемный работник, предприниматель, супруг, родитель и т.д. А возможно и не приобретает. Например, не каждый человек служит в армии, учится в вузе, женится и становится отцом или предпринимателем. Или утрачивает. Например, закончил учиться, отслужил в армии или развелся. Следовательно, один и тот же объект должен иметь возможность принадлежать разным классам и этот набор классов должен быть динамическим, т.е. изменяться в ходе эволюции объекта и самой программной системы. На набор классов, к которым относится объект, как правило, накладываются ограничения. Например. Чтобы стать солдатом, человек должен достичь 18 лет. А если человек студент то для того, чтобы стать мужем, необходимо сдать сопромат.

2.8. О сообщениях. Еще одна странность ООП. «Поведение - это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений» (здесь и далее цитаты по [5]). Но, постойте, если я моделирую столкновение двух автомобилей то, кто из них и какие получает и передает сообщения?

2.9. О свойствах. «Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств». Но принадлежит ли свойство объекта самому объекту? Буду утверждать, что нет. Например, мы говорим данное яблоко – зеленое. Но что это означает не самом деле? Это значит, что если мы направим источник света, близкого по спектру к солнечному свету, то данное яблоко поглотит все длины волн, кроме тех, которые соответствуют диапазону зеленого цвета, и наблюдатель, который способен воспринимать весь спектр солнечного света увидит только отраженный зеленый свет. Если источник или наблюдатель имеют другой диапазон, например, инфракрасный, то цвет яблока будет черным. Таким образом, свойство не есть неотъемлемая характеристика объекта, а является возможным проявлением объекта при его взаимодействии с другими объектами. Например, свойство предмета «плавать по поверхности» может проявляться во взаимодействии с водой и не проявляться во взаимодействии со спиртом.

2.10. Об агрегации. Мы говорим: швабра «состоит из» (агрегирует) щетки и палки. Но что это означает? Если мы рассматриваем швабру, как предмет способный перемещаться в пространстве, то в этом случае агрегация никак не проявляется, мы рассматриваем швабру как атомарный объект и нам интересны лишь ее общая масса и размеры. Но если мы станем нагружать швабру и испытывать ее на прочность, то агрегация проявится, как взаимодействие ее структурных составляющих щетки и палки и результат испытания будет зависеть от того, происходит ли это взаимодействие посредством вбитых гвоздей или посредством пазов, шипов и клея. Поэтому, агрегация есть так же взаимодействие объектов.

2.11. Об ассоциации. Если мы говорим, что объект *a* является мужем объекта *b*, то мы декларируем что объекты *a* и *b* связаны ассоциацией. Как же может проявляться эта связь? Во-первых, сама эта связь есть результат взаимодействия трех объектов *a, b* и объекта и *c* (ЗАГС, некий регистратор, где эта связь рождается, и там же она, кстати, может и исчезнуть). Во-вторых, она может проявляться во взаимодействий объектов, например, *a* и *b*: совместное расходование семейного бюджета или рождение и воспитание общего ребенка. Нет смысла спрашивать *a.isMarrid (b)*, он может и соврать. Но вполне осмысленна функция *isMarrid (a, b, c).* Следовательно, связи, как и свойства, есть возможные взаимодействия между объектами.

2.12. Об идентичности. «Идентичность - это такое свойство объекта, которое отличает его от всех других объектов». Что должно соответствовать в действительности этому утверждению? Допустим, я перекрасил свой автомобиль. Для меня он, безусловно, остался тем же самым. Но для ГИБДД это будет совсем другой объект, который получит новый номер государственной регистрации. Другой пример. С автомобиля сняли колеса. Автомобиль остался тем же самым? А если еще и двигатель? А когда автомобиль перестанет быть тем же самым? Идентичность возникает лишь при связывании конкретных объектов. Каждая швабра будет состоять из вполне конкретной щетки и палки. У каждого мужа будет вполне конкретная жена, а у каждого клиента – свой экземпляр счета в банке.

2.13. Итак, в нашем ментальном мире нет объектов и их свойств. А есть структуры и их взаимодействия. Если мы моделируем дорожное движение, то автомобили следует рассматривать как элементарные узлы структуры, которые вступая во взаимодействие (обгоняя, подрезая, тормозя), консистентно изменяют свои координаты в фазовом пространстве (скорость, положение). С другой стороны, автомобили перестают быть элементарными узлам структуры, как только мы начинаем рассматривать их взаимодействие при столкновении. Мы должны перейти на более низкий структурный уровень.

На этом уровне элементарными узлами должны быть: бампер, кузов, рама, двигатель и проч. составные части, а также способы их соединения (болты, сварка и т.д.). Сам факт выделения узла структуры есть акт познавательной деятельности (гештальттеория, [6]), который зависит от решаемой задачи.

3. Об онтологии Людвига Витгенштейна

   3.1. Мир Витгенштейна [7] не является миром вещей, как у Аристотеля. Если для Аристотеля сущность языка - существительное, то по Витгенштейну - сущность языка предложение. Язык, согласно Витгенштейну, представляет собой не набор имен или свойств, а состоит из предложений, в которых выражаются взаимодействия предметов. В мире Витгенштейна первичны факты - взаимодействия между предметами, а вещи определяются совокупностью их возможных взаимодействий. Такой взгляд на мир очень хорошо соответствует современной теории категорий, в которой морфизмы – задают взаимодействия объектов, а функторы – определяют эволюцию модели во времени.

   3.2. В отличие от статичного мира Аристотеля мир Витгенштейна динамичен. В ходе эволюции мира факты могут добавляться и представление о вещах будет меняться. Благодаря такой трактовке мира вещь выступает не как нечто данное, застывшее, вполне определенное, а как некоторая сущность с нечеткими, изменчивыми границами. Программные системы динамически эволюционируют подобно миру Витгенштейна. В ходе развития программной системы разработчики добавляют объектам новые возможные взаимодействия, которые расширяют наборы их свойств и связей.

   3.3. Витгенштейн ввел понятие «языковая игра» - единое целое: язык и действия, с которыми он переплетен. «Сколько же существует типов предложения? Скажем, утверждение, вопрос, повеление? Имеется бесчисленное множество таких типов и бесконечно разнообразны виды употребления всего того, что мы называем "знаками", "словами", "предложениями". И эта множественность не представляет собой чего-то устойчивого, раз и навсегда данного…». Разве мы не играем в подобную языковую игру, когда описываем модель предметной области при помощи пользовательских историй?

4. О предметно-ориентированном языке (Domain Specific Language, DSL)

   4.1. Мне представляется, что в будущем среды программирования будут в основном похожи на CAD системы с встроенным DSL. Программные системы должны создавать специалисты прикладной области, например, инженер будет собирать программную модель для исследования летных качеств нового самолета из готовой элементной базы: фюзеляжа, крыльев, двигателей, соединительных креплений и моделей их взаимодействия с набегающим потоком. Аналогично, будут собираться из готовых компонентов и бизнес-приложения: бухгалтерия, логистика, склад, производство и др.

   4.2. Означает ли это, что для каждой предметной области мы должны создавать свой уникальный DSL? И да, и нет. Да, в том смысле, что практически во всех устоявшихся областях профессиональной деятельности людей, например, математика, медицина, биология, существует свой специфический язык, поэтому в каждой предметной области нам придется создавать свой DSL для описания конкретных моделей решения специфических задач. Нет, потому что профессиональные языки строятся на основе единого синтаксиса естественного языка и должен существовать универсальный синтаксис для описания ментальных моделей, основанный на общих законах человеческого мышления.

   4.3. Попытки создания универсального синтаксиса DSL предпринимались не раз. Перечислю лишь наиболее заметные из них: [8], [9] и [10]. Мне неизвестны примеры, серьезного практического применения предложенных подходов. Может быть, это происходит потому, что в них пытаются моделировать статический мир объектов Аристотеля? Вполне возможно, что основой для универсального синтаксиса DSL может стать динамический мир взаимодействий Витгенштейна и категорный подход, отражающие фундаментальные особенности нашего мышления.

   4.4. DSL подобно ДНК должен описывать:

      1) сценарии сборки модели из готовых компонентов;

      2) инициализацию начального состояния;

      3) законы эволюции модели во времени.

   Главным строительным блоком языка должно стать взаимодействие, в котором проявляются свойства объектов, изменяются их состояния и рождаются новые экземпляры. Каждый объект в ходе эволюции может быть дополнен новым набором потенциальных взаимодействий и это дополнение не должно затрагивать ранее созданный код. Появление «человека» в мире «камней» и «дров» при таком подходе приведет к созданию нового взаимодействия *carry (carrier, thing, space, gravitation)*. Элемент carrier должен реализовывать взаимодействия: *take (carrier, thing)* и *move (carrier, thing, space)*. А элемент *thing* должен реализовывать взаимодействие: *weight (thing, gravitation)* и *shape (thing, space)*. В этом случае наша модель действия нести получится

достаточно универсальной. Метод не будет меняться от того, что человек реализует взаимодействие take руками, животное - зубами, а подъемный кран - специальным захватом. Поэтому мы сможем легко пополнять нашу программную систему любыми элементами, которые реализуют необходимые взаимодействия для carrier, thing и нам не понадобится менять нашу универсальную функцию. На что это может быть похоже? Возможно, будущий DSL будет похож на Mixin-технологию или Anemic Domain Model (кстати, эту модель Мартин Фаулер назвал антипаттерном, [11]), или на «словесное программирование» Д. Кнута [12], где повествовательное описание взаимодействий будут интерпретироваться в виде алгоритмов и структур данных.

[1] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, “Refactoring: Improving the Design of Existing Code”, 1999, Addison Wessley, Inc.

[2] Edsger W. Dijkstra, “American programming's plight”, ACM SIGSOFT Software Engineering Notes Homepage archive,Volume 6 Issue 1, January 1981

[3] С. Лем, “О выгодности дракона”, рассказ из цикла “Звёздные дневники Ийона Тихого” 1991

[4] Ветров А.А., "Семиотика и ее основные проблемы", Издательство политической литературы, Москва . 1968

[5] Гради Буч, Объектно - ориентированный анализ и проектирование с примерами приложений на C++, Бином, 1998

[6] Вертгеймер М. Продуктивное мышление. М.: Прогресс, 1987.

[7] Л.Витгенштейн, "Философские исследования", Кембридж, 1945

[8] Charles Simonyi, “Intentional Programming”, From Wikipedia, the free encyclopedia, (http://en.wikipedia.org/wiki/Intentional_programming)

[9] Krzysztof Czarnecki, Ulrich Eisenecker , “Generativ Programming: Methods, Tools, and Applications”, Addison Wessley, Inc., 2000,

[10] Sergey Dmitriev, “Language Oriented Programming: The Next Programming Paradigm”, 2004, (http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf)

[11] Martin Fowler AnemicDomainModel, 2003, (http://martinfowler.com/bliki/AnemicDomainModel.html)

[12] Donald E. Knuth, Literate Programming, 1984 (http://www.literateprogramming.com/knuthweb.pdf)