Пол Грэм

Языки программирования через сто лет

Компьютера. Опубликовано 03 августа 2004 года

Об авторе. Пол Грэм - известный специалист по языку Lisp, автор классических учебников On Lisp и ANSI Common Lisp. В 1995 году он стал одним из создателей Viaweb - первого в истории веб-приложения. В 2002 году Грэм предложил статистический метод фильтрации спама, сейчас используемый в большинстве противоспамовых систем.
Статья публикуется в сокращении.

Языки программирования, подобно формам жизни, образуют эволюционное древо. На этом древе есть и тупиковые ветви, и некоторые из них уже известны. Кобол, несмотря на всю свою популярность в былые годы, похоже, не оставил интеллектуальных потомков.

Я считаю, что похожая судьба ждёт и Джаву. Люди спрашивают меня: "Как можно говорить, что Джаве не быть? Она уже стала успешным языком". И я не могу не согласиться с ними. Джава - успешный язык, если считать мерилом успеха площадь полок с учебниками Джавы в книжных магазинах или количество студентов, убеждённых, что знание Джавы поможет им найти работу. Я имел в виду другое. Мне кажется, Джава окажется таким же эволюционным тупиком, как Кобол.

Это лишь гипотеза. Я могу ошибаться. Но речь не о Джаве, сейчас я хотел бы поговорить об эволюции языков и побудить всех задуматься о том, на какой ветви эволюционного древа помещается язык X. Смысл постановки такого вопроса заключается отнюдь не в том, чтобы через столетие наши бестелесные духи могли заявить, дескать, мы же предупреждали. Придерживаться основных ветвей эволюции может оказаться неплохой тактикой при выборе языка и сегодня.

Во что превратятся языки программирования через сто лет, мне интересно, потому что хотелось бы знать, на какую ветвь древа стоит делать ставки сейчас.

***

Любой язык программирования можно поделить на две части: некий набор фундаментальных операторов, которые играют роль аксиом, и остаток языка, который, в принципе, может быть описан в терминах этих фундаментальных операторов.

Я считаю, что фундаментальные операторы - это самый важный фактор, влияющий на выживание языка в долгосрочной перспективе. Всё остальное может меняться.

Важен не только хороший подбор аксиом, но и их немногочисленность. Математики всегда понимали, что аксиом должно быть как можно меньше, а они-то кое-что в этом смыслят.

Есть подозрение, что на основных ветвях эволюционного древа языков программирования располагаются языки с самыми небольшими и прозрачными ядрами. Чем крупнее доля языка, которая может быть написана на нём самом, тем лучше.

Конечно, уже в самой постановке вопроса о том, какими станут языки программирования через сто лет, есть немалое допущение. А будут ли тогда вообще писать программы? Может быть, мы просто будем говорить компьютерам прямым текстом, чего от них хотим?

Однако до сих пор особого прогресса по этой части не наблюдалось. Подозреваю, что и через сто лет людям придётся объяснять компьютерам свои желания посредством программ.

Может вызывать сомнения сама возможность прогнозирования того, какой станет техника через сто лет. Но не стоит забывать, что у нас за плечами почти пятьдесят лет истории программирования. Если

обратить внимание на то, как медленно языки менялись до сих пор, попытки что-то предсказать уже не покажутся такими бесперспективными.

Скорость развития языков программирования так мала из-за того, что на самом деле языки не являются технологиями. Языки программирования - это форма записи. Программа представляет собой формальное описание проблемы, которую нужно решить с помощью компьютера. Так что по темпам развития языки программирования ближе к математической нотации, чем, скажем, к средствам траспорта или связи. Математическая нотация тоже эволюционирует, но не такими гигантскими скачками, как техника.

***

Из чего бы ни делались компьютеры через сто лет, можно не сомневаться, что они будут гораздо быстрее, чем сейчас. Если действие закона Мура сохранится, компьютеры станут в 74 квинтиллиона раз быстрее. Это сложновато себе представить. Впрочем, по всей вероятности, закон Мура окажется несостоятельным. Всё, что должно удваиваться каждые восемнадцать месяцев, рано или поздно наталкивается на какой-нибудь фундаментальный предел.

Но даже если компьютеры станут быстрее лишь в жалкий миллион раз, это приведёт к не менее радикальным подвижкам самых основ, на которых строятся языки программирования. Помимо всего прочего, появится больше применений для языков, которые сейчас считаются "медленными", то есть тех языков, которые не транслируются в очень эффективный код.

Несмотря на это, приложения, требующие высокой производительности, будут существовать всегда. Некоторые задачи, которые решаются с помощью компьютеров, порождаются самими компьютерами. Например, скорость, с которой необходимо обрабатывать видео, напрямую зависит от скорости, с которой машина способна его генерировать. Кроме того, существует класс задач, которые по определению обладают неограниченной способностью поглощать все доступные ресурсы: визуализация, криптография, моделирование.

Если некоторые приложения могут становиться всё менее эффективными, а другие по-прежнему будут пытаться "выжать из железа последнее", языкам предстоит отвечать за неуклонно расширяющийся спектр задач. И это уже начинает происходить. Существующие реализации некоторых популярных новых языков ошеломительно расточительны по меркам прошлых десятилетий.

Такое происходит не только с языками программирования. Это всеобщая историческая тенденция. Развитие техники даёт новым поколениям возможность делать вещи, которые раньше считались бы излишеством. Лет тридцать назад люди были бы потрясены, узнав, насколько обыденными станут в наше время междугородные телефонные звонки. А лет сто назад известие о том, что сейчас за один день посылка может преодолеть путь от Бостона до Нью-Йорка через Мемфис, поразило бы всех ещё сильнее.

***

Я уже знаю, что случится со всеми дополнительными ресурсами, которые предоставит сверхбыстрое аппаратное обеспечение через сто лет - их почти целиком будут тратить впустую.

Когда я учился программировать, компьютеры располагали скудными возможностями. Я помню, как приходилось вычищать пробелы из программ на Бейсике, чтобы они помещались в четыре килобайта памяти моего TRS-80. Мысль о том, что все эти изумительно неэффективные программы сожрут ресурсы, делая одно и то же снова и снова, кажется мне кощунством. Однако, похоже, здесь интуиция мне изменяет. Я напоминаю человека, выросшего в бедности и продолжающего экономить даже на самом необходимом, например, на лекарствах.

Но не всякое расточительство вредно. При современной инфраструктуре телефонной связи поминутная оплата междугородных звонков начинает казаться крохоборством. Если есть такая возможность, элегантнее считать все звонки однотипными и не учитывать расстояние, которое разделяет абонентов.

Бывают примеры правильного расточительства, и бывает и неверная расточительность. Меня интересует полезное. Например, расточительство, которое вынудит тратить больше, но взамен позволит упростить устройство. Какую выгоду можно извлечь из огромных ресурсов нового быстрого аппаратного обеспечения?

Жажда скорости так глубоко въелась в нас с нашими жалкими компьютерами, что без сознательных усилий с ней не совладать. В устройстве языков программирования следует сознательно отыскивать ситуации, в которых можно извлечь из возросшей эффективности хотя бы некоторый излишек удобства.

***

Причина существования большинства типов данных - это производительность. Например, во многих современных языках есть и строки, и списки. Семантически строки - это, в той или иной степени, частный случай списков, элементы которых - символы. Так нужен ли тогда отдельный тип данных? На самом деле, не нужен. Строки существуют только для повышения эффективности. Ну не глупо ли забивать семантику языка хитростями, которые позволяют программам работать быстрее? Строки в языках - это очередной пример преждевременной оптимизации.

Если считать основой языка набор аксиом, то дурно в погоне за эффективностью копить лишние аксиомы, которые не дают языку дополнительной выразительной силы. Эффективность важна, но мне кажется, что добиваться её следует иначе.

Думаю, верным решением этой проблемы было бы разделение смысла программы и тонкостей реализации. Вместо того чтобы иметь и строки, и списки, лучше обойтись только списками, но иметь возможность дать компилятору совет по оптимизации, который, при необходимости, позволит ему представить строки в виде последовательности байтов.

Поскольку скорость не важна в большинстве программ, как правило, можно не заботиться о подобных мелочах. Чем быстрее будут становиться компьютеры, тем вернее будет это утверждение.

***

Сокращение количества информации о конкретной реализации сделает программы гибче. Спецификации меняются, пока программа пишется, и это не только неизбежно, но и желательно.

Слово "эссе" происходит от французского глагола "essayer", который в переводе означает "пытаться". Первоначально так назывались тексты, которые писали, для того чтобы в чём-либо разобраться. С программным обеспечением то же самое. Думаю, некоторые из лучших программ представляли собой "эссе", в том смысле, что их авторы начинали работу, не представляя себе в точности, что они пытаются написать.

Программисты на языке Лисп всегда осознавали значение гибкого подхода к типам данных. В самой первой версии программы они стремились использовать списки для всего. Эта версия может быть так потрясающе неэффективна, что приходится нарочно стараться не думать о том, как она работает, как, поедая бифштекс, лучше не задумываться о том, из чего он сделан (во всяком случае, со мной дело обстоит именно так).

Через сто лет программисты захотят такой язык, на котором можно оперативно и с минимальными усилиями набросать первую, невероятно неэффективно работающую версию программы. По крайней мере, так это можно описать в современных терминах. Они скажут, что им нужен язык, на котором легко программировать.

Неэффективные программы - отнюдь не кощунство. Кощунство - это языки, которые заставляют программистов выполнять ненужную работу. Не расход машинного времени, а пустая трата времени программиста - вот истинная неэффективность. И это будет становиться всё очевиднее по мере повышения скорости работы компьютеров.

***

Избавиться от строк можно позволить себе уже сегодня. Но как далеко можно зайти с этим упрощением типов данных? Есть варианты, которые шокируют даже меня, несмотря на то что я специально работал над расширением собственных взглядов. К примеру, можно ли избавиться от массивов? В конце концов, массивы - это лишь частный случай хэш-таблиц, в которых в качестве ключей используют множество целых чисел. А станем ли мы заменять сами хэш-таблицы списками?

Есть ещё более ошеломительные перспективы. Например, в языке Лисп, который в 1960 году изобрел профессор Маккарти, отсутствовали числа. Если рассуждать логически, то отдельная запись для чисел не нужна, ведь их можно представить в виде списков: число n может быть представлено как список из n элементов. Вычисления можно выполнять таким способом. Просто это невыносимо неэффективно.

В действительности, никто не предлагал реализовывать числа в виде списков. Фактически, реализация научной работы, которую Маккарти написал в 1960 году, вообще не планировалась. Это была чисто теоретическая попытка создания более элегантной альтернативы машине Тьюринга. Когда кто-то неожиданно взял работу Маккарти и превратил её в работающий интерпретатор Лиспа, числа, разумеется, не были представлены списками, они были представлены двоичными значениями, как и в любом другом языке.

Может ли развитие языков программирования зайти настолько далеко, что в них не будет чисел как фундаментального типа данных? (Я задаю себе этот вопрос не всерьез, а только ради того, чтобы подразнить будущее. Это похоже на гипотетическое столкновение неминуемого с недвижимым, только в нашем случае против невообразимо неэффективной реализации ставятся невообразимо огромные ресурсы). Думаю, может. Почему бы и нет? Будущее - штука несиюминутная. Если что-то позволяет уменьшить число аксиом в основе языка, это и есть та сторона, на которую следует делать ставки, когда t стремится к бесконечности. Если через сто лет идея окажется невыполнимой, то, может быть, через тысячу лет уже всё изменится.

(Расставлю точки над i: я не предлагаю вести все числовые вычисления действительно при помощи списков. Я предлагаю, чтобы прежде любых дополнительных упоминаний о реализации именно так определялась основа языка. На практике программы, ведущие вычисления, вероятно, будут представлять числа в виде двоичных значений, но это оптимизация, а не часть семантики основы языка.)

***

Многочисленные программные прослойки между приложением и аппаратным обеспечением - другой хороший способ израсходовать лишние такты. Эту тенденцию тоже можно наблюдать уже сегодня: многие новые языки программирования компилируются в байт-код. Эмпирически можно считать, что каждый уровень интерпретации понижает скорость вдесятеро. Такую цену приходится платить за гибкость.

Писать программы как набор уровней - это мощный метод, даже если он используется в приложениях. При программировании "снизу вверх" программа пишется в виде серии уровней, каждый из которых служит языком для вышестоящего уровня. Чем большую часть своего приложения вы сможете превратить в язык для написания приложений подобного рода, тем удобнее будет повторно использовать ваш код.

В восьмидесятых годах идею повторного использования каким-то образом связали с объектно-ориентированным программированием, и сколь угодно многочисленные доказательства обратного, по-видимому, уже не избавят от этого клейма. Хотя иногда объектно-ориентированный код годится для повторного использования, таким его делает не объектно-ориентированность, а программирование "снизу вверх". Возьмём, например, библиотеки: их можно повторно использовать, потому что, по сути, они представляют собой язык. И неважно, написаны ли они в объектно-ориентированном стиле или нет.

Кстати, я вовсе не предрекаю гибель объектно-ориентированного подхода. Хотя, по моему мнению, за исключением некоторых специализированных областей применения, объектно-ориентированность ничего не даёт хорошим программистам, она очень привлекательна для больших организаций. ООП - это приличный способ написания путаного лапшеобразного кода, позволяющий строить программы в виде серии патчей. Большие организации всегда были склонны разрабатывать программное обеспечение таким образом, и думаю, этому и через сто лет не измениться.

***

Поскольку речь идет о будущем, стоит затронуть тему параллельных вычислений, поскольку, как кажется, именно в них собака и зарыта. То есть, вне зависимости от момента нашего разговора, параллельные вычисления, по-видимому, буду оставаться чем-то таким, что произойдет в будущем.

Нагонит ли их будущее когда-нибудь? О параллельных вычислениях говорят, как о чем-то неизбежном, уже лет двадцать, и до сих пор это не особенно повлияло на методики программирования. Или повлияло? Разработчики микросхем уже должны думать о нём, как и программисты, пишущие системное программное обеспечение для многопроцессорных компьютеров.

Но в действительности вопрос в том, насколько высоко по лестнице абстракций сумеет взобраться параллелизм? Придётся ли прикладным программистам учитывать его существование через сто лет? Или он так и останется заботой создателей компиляторов, но практически не видим в исходном коде приложений?

Вероятнее всего, возможности параллелизма, по большей части, растратят впустую. Это частный случай моего более общего прогноза, согласно которому дополнительные вычислительные ресурсы, которые мы получим, окажутся, по большей части, будут израсходованы зря.

Я ожидаю, что вместе с бешеной скоростью применяющегося "железа" параллелизм будет доступен, если отчетливо востребовать его, но использоваться он будет нечасто. Это подразумевает, что параллелизм, который станет распространён через сто лет, не будет цельным. Скорее всего, для обычного программиста это будет выглядеть так: процессы можно будет разветвлять, и все процессы будут исполняться параллельно.

И как в случае со специфическими реализациями структур данных это будет осуществляться на довольно поздних стадиях разработки - при оптимизации. Первая версия, как правило, будет игнорировать все преимущества, которые могут дать параллельные вычисления, в точности как игнорируются преимущества, полученные благодаря специфическому представлению данных.

За исключением некоторых особых видов приложений, параллелизм не будет распространен в программах через сто лет. Обратное положение дел стало бы поспешной оптимизацией.

Кто изобретет язык программирования будущего

Одна из поразительных тенденций последнего десятилетия - это появление множества языков с отрытыми исходниками, таких как Perl, Python и Ruby. Дизайн языков захвачен хакерами [примечание: здесь и далее слово "хакер" употребляется в его исходном положительном значении, а не как синоним "компьютерного преступника"]. Результаты пока неоднозначны, но уже способны воодушевить. В языке Perl, например, попадаются сногсшибательные идеи. Правда, попадаются и ужасные, но так всегда и бывает с амбициозными проектами. Бог знает, что может вырасти из Perl за сто лет, если он продолжит мутировать такими темпами.

Говорят, что те, кто не может - учит. Это не так (некоторые из лучших хакеров, с которыми я знаком, являются профессорами), но действительно есть множество вещей, которыми невозможно заниматься, когда преподаёшь. Занятия наукой накладывают кастовые ограничения. В любой научной области есть темы, над которыми можно работать, и темы, которых лучше избегать. К несчастью, разница между ними обычно заключается в том, насколько интеллектуально звучит описание исследования в научных статьях, а не в том, насколько оно важно для получения хороших результатов.

Увы, между дозволенными исследованиями и исследованиями, которые приводят к появлению хороших языков, нет почти ничего общего. Возьмём, к примеру, типы данных, которые, кажется, превратились в неисчерпаемый источник тем для научных статей. А ведь статический контроль типов исключает существование истинных макросов, без которых, по моему мнению, ни одним языком не стоит пользоваться.

Языки всё реже разрабатываются в "исследовательских" целях, и всё чаще как проекты с открытым исходным кодом. Дело, впрочем, не в этом. Тенденция, скорее, состоит в том, что языки теперь придумывают не авторы компиляторов, а прикладные программисты, которым и придётся с ними работать. Мне кажется, это хорошая тенденция, и я рассчитываю, что она сохранится.

Язык будущего сегодня

Развитие физики в течение следующих ста лет предсказать невозможно. Программирование - совсем другое дело. Думаю, принципе, сейчас вполне возможно изобрести язык, который будет привлекателен для пользователей через сто лет.

Вот один из способов: просто возьмём и попробуем написать такую программу, которую хотелось бы иметь возможность написать. Не нужно заботиться о том, существует ли компилятор, чтобы перевести её в машинный код, или "железо", на котором она заработает. Когда пишете, рассчитывайте на неограниченные ресурсы. Чтобы вообразить неограниченные ресурсы, не нужно ждать сто лет. Это можно сделать прямо сейчас.

Какую программу хотелось бы писать? Неважно, лишь бы программирование требовало меньше труда. Есть одно "но": если бы ваше представление о программировании не было искажено языками, к которым вы привыкли сейчас, что угодно требовало бы меньше труда. Привычки могут быть настолько навязчивыми, что пересилить их можно с великим трудом. Может показаться, что таким ленивым существам как мы должно быть очевидно, как выражать свои мысли в программах с наименьшими усилиями. На самом же деле, язык, на котором мы думаем, так ограничивает наши представления о возможном, что более простые способы формулировки программ удивят нас.

Приблизительно оценить сложность написания программы можно по её длине. Разумеется, измерять длину программы надо не в символах, а в отдельных синтаксических элементах - фактически, нас интересует размер дерева грамматического разбора. Написать короткую программу не всегда проще, но лучше уж целиться в ясную мишень краткости, чем в расплывчатую мишень меньшего труда. Это значит, что при разработке языка надо действовать так: посмотрим на программу и спросим себя, существует ли способ записать её ещё более кратко.

Кому нужен язык будущего

На поверку оказывается, что написать программу на воображаемом языке будущего удаётся, только когда используются лишь базовые возможности языка. Написать процедуру сортировки можно и сейчас. Но нельзя предвидеть, какие библиотеки понадобятся через сто лет. Скорее всего, многие из них будут служить для решения проблем, которых сейчас попросту нет. Если проект SETI@home завершится успешно, нам потребуются библиотеки для связи с инопланетянами. Впрочем, возможно, инопланетяне уже достаточно развиты, чтобы поддерживать связь в формате XML.

С другой стороны, я думаю, основу языка будущего можно было бы придумать и сегодня. В сущности, кое-кто может сказать, что он, по большей части, уже придуман в 1958 году.

Если бы мы получили язык программирования будущего, стали бы мы использовать его? Чтобы попытаться ответить на этот вопрос, оглянемся назад. Захотел бы кто-нибудь программировать на современных языках сорок лет назад?

С одной стороны, ответ отрицательный. Сегодняшние языки подразумевают инфраструктуру, которой не было в 1960 году. Например, языки, в которых отступы в тексте имеют значение (к их числу относится Python), сложно использовать с терминалами на базе печатающих устройств. Однако если не

обращать внимания на проблемы такого рода (предположим, что программы просто пишутся на бумаге), вопрос остаётся: захотели бы программисты шестидесятых использовать языки, которые мы используем сейчас?

Думаю, да. Самым ограниченным из них, наверное, мешали бы въевшиеся в мозг особенности ранних языков (разве можно манипулировать данными без операций с указателями? как организовать ветвление, когда нет goto?) Однако у самых умных программистов тех времён не возникло бы проблем с большинством современных языков, доведись им столкнуться с ними.

Сейчас из языка будущего, по крайней мере, вышел бы отличный псевдокод. Но подошёл бы он для написания настоящего программного обеспечения? Поскольку язык будущего должен транслироваться в быстрый код для некоторых видов приложений, он, вероятно, сможет работать достаточно эффективно на современном "железе". Нам придётся больше заботиться об оптимизации, чем программистам через сто лет, но мы всё равно можем остаться в выигрыше.

Таким образом, у нас есть две идеи, и если сложить их, обнаружатся интересные возможности. Во-первых, язык программирования будущего, в принципе, может быть изобретён сегодня. Во-вторых, такой язык может годиться для программирования и в наше время. Сложно удержаться и не подумать: так почему бы не попробовать написать язык будущего прямо сейчас?

При разработке языков программирования стоит помнить о будущем. Когда учат водить автомобиль, один из принципов такой: смотреть надо вперёд, а не на дорогу под колёсами. Даже если всё, что вас волнует, расположено не далее трёх метров. С языками программирования следовало бы придерживаться того же правила.