Пять вопросов о проектировании языков программирования

Автор оригинала: Paul Graham

Руководящая философия

1. Языки программирования для людей

Языки программирования это то, как люди говорят с компьютерами. Компьютер будет рад поговорить на любом языке, который не будет двусмысленным. Причина почему у нас есть высокоуровневые языки — потому что люди не могут справиться с машинным языком. Суть языков программирования в том, чтобы предотвратить наш бедный хрупкий человеческий мозг от перегрузки массой деталей.

Архитекторы знают, что некоторые проблемы проектирования более приземлённые, чем другие. Одни из самых ясных и абстрактных проблем проектирования это проектирование мостов. В этом случае

ваша работа в том, чтобы охватить требуемое расстояние как можно меньшим количеством материала. На другом конце спектра — проектирование стульев. Проектировщики стульев должны тратить своё время на размышления о человеческих задницах.

Разработка софта имеет схожее различие. Проектирование алгоритмов для маршрутизации данных через сеть — хорошая, абстрактная проблема, как проектирование мостов. Тогда как проектирование языков программирования подобно проектированию стульев: нужно справляться с человеческими слабостями.

Большинству из нас тяжело это осознавать. Проектирование элегантных математических систем звучит гораздо более привлекательно для большинства из нас, чем потворство человеческим слабостям. Роль математической элегантности в том, что некоторая степень элегантности делает программы проще для понимания. Но элегантностью всё не ограничивается.

И когда я говорю, что языки должны быть спроектированы, чтобы учитывать человеческие слабости, я не имею ввиду что языки должны быть спроектированы для плохих программистов. На самом деле вы должны проектировать софт для лучших программистов, но даже лучшие программисты имеют свой предел. Я не думаю, что хоть кому-нибудь понравится программировать на языке, где все переменные обозначались бы буквой «x» с целочисленными индексами.

2. Проектируйте для себя и для своих друзей

Если вы посмотрите на историю языков программирования, большинство из самых лучших языков были спроектированы для использования своими же авторами, а большинство из худших — спроектированы для других людей.

Когда языки проектируются для других людей, то это всегда какая-то конкретная группа людей: люди не такие умные, как создатели языка. Так вы получаете язык, который говорит с вами снисходительно. Cobol — самый яркий пример, но большинство языков пронизаны этим духом.

Это не имеет ничего общего с тем, насколько высокоуровневым является язык. C — достаточно низкоуровневый, но он был создан для использования его авторами, вот почему хакеры любят его.

Аргумент в пользу проектирования языков для плохих программистов в том, что плохих программистов больше, чем хороших. Пожалуй это так. Но это небольшое количество хороших программистов пишет непропорционально больше софта.

Меня интересует вопрос, как создать язык, который понравится самым лучшим хакерам? Мне кажется этот вопрос идентичен вопросу, как создать хороший язык программирования?, но даже если это не так, то по крайней мере это интересный вопрос.

3. Дайте программисту столько контроля, сколько возможно

Многие языки (особенно те, которые созданы для других людей) ведут себя как няньки: они стараются предостеречь вас от вещей, которые, по их мнению, будут вам не полезны. Я придерживаюсь противоположного мнения: дайте программисту столько контроля сколько можете.

Когда я впервые изучал Lisp, больше всего мне понравилось то, что мы разговаривали на равных. В других языках, которые я к тому моменту изучил, был язык, а была моя программа на этом языке, и они существовали довольно отдельно. Но в Lisp'е функции и макросы, которые я написал, были такими же на которых был написан сам язык.

Я мог переписать сам язык, если бы захотел. Он имел такую же привлекательность, что и софт с открытым кодом.

4. Краткость — сестра таланта

Краткость недооценена и даже презирается. Но если вы загляните в сердца хакеров, вы увидите, что они очень любят краткость. Сколько раз вы слышали, как хакеры с любовью говорят о том, что, скажем, в APL они могут делать удивительные вещи с помощью всего лишь пары строк кода? Я полагаю, что по-настоящему умные люди на самом деле любят обращать внимание на это.

Я полагаю, что почти всё, что позволяет сделать программы короче — хорошо. Должно быть множество библиотечных функций, всё, что может быть неявным — должно таким быть; синтаксис должен быть в большей степени кратким; даже имена сущностей должны быть короткими.

И не только программы должны быть короткими. Мануалы тоже должны быть короткими. Добрая часть мануалов наполнена разъяснениями, оговорками, предупреждениями и особыми случаями. Если вам потребуется сократить мануал, лучший вариант это исправить язык, который требует так много объяснений.

5. Признайте, что такое хакерство

Многим людям хотелось бы, чтобы хакерство было бы математикой или, хотя бы, чем-то похожим на естественные науки. Я думаю, что хакерство больше похоже на архитектуру. Архитектура связана с физикой, в том плане, что архитектору нужно проектировать здание, которое не упадёт, но настоящая цель архитектора в том, чтобы создать великое здание, а не сделать открытия в области статики.

Что хакеры любят, так это создавать великие программы. И я думаю,

что, по крайней мере, в наших собственных мыслях, мы должны помнить, что писать замечательные программы — это замечательно, даже когда эта работа нелегко переводится в обычную интеллектуальную валюту научных работ. С интеллектуальной точки зрения, не менее важно как разрабатывать язык, который полюбят программисты, так и создавать ужасный, воплощающий идею, о которой вы можете опубликовать статью.

Открытые проблемы

1. Как организовать большие библиотеки?

Библиотеки становятся важной частью языков программирования. Они становятся настолько большими, что это может быть опасно. Если требуется больше времени, чтобы найти функцию в библиотеке, которая делает то, что вам нужно, чем написать эту функцию самому, то весь код не делает ничего, кроме утолщения вашего мануала. (Руководства Symbolics были тому примером.) Так что нам придётся решит проблему с организацией библиотек. В идеале спроектировать их так, чтобы программист мог бы догадаться, какая функция библиотеки подойдёт.

2. Люди действительно напуганы префиксным синтаксисом?

Это открытая проблема в том смысле, что я думал над ней несколько лет и всё ещё не знаю ответ. Префиксный синтаксис кажется мне абсолютно естественным, возможно кроме использования его в математике. Но может так, что большая часть непопулярности Lisp'ов просто из-за незнакомого синтаксиса… Стоит ли что-то с этим делать, если это правда, это другой вопрос.

3. Что вам нужно для серверного софта?

Я думаю, что большинство приложений, которые будут написаны в следующие двадцать лет будут веб-приложениями, в том смысле, что программы будут расположены на сервере и будут общаться с вами через веб-браузер. И чтобы написать такие приложения нам нужны новые штуки.

Одна из таких вещей это поддержка нового способа выпуска серверных приложений. Вместо одного или двух больших релизов в год, как десктопный софт, серверный софт будет выпускаться серией мелких изменений. У вас может быть пять или десять релизов в день. И у всех всегда будет последняя версия.

Вы знаете как проектировать программы, чтобы они были поддерживаемыми? Серверный софт должен быть спроектирован способным к изменениям. У вас должна быть возможность изменять его легко, или хотя бы знать что значит мелкое изменение и что является важным.

Ещё одна вещь, которая может быть полезной в серверном софте, это, внезапно, непрерывность поставки. В веб-приложении вы можете использовать что-то вроде CPS, чтобы получить эффект от подпрограмм в stateless мире веб-сессий. Может иметь непрерывность поставки стоит того, если эта возможность не будет слишком дорогой.

4. Какие новые абстракции осталось открыть?

Я не уверен, насколько разумна такая надежда, но лично мне бы очень хотелось открыть новую абстракцию — нечто такое, что могло бы иметь такое же большое значение, как функции первого класса или рекурсия или хотя бы параметры по умолчанию. Может это невыполнимая мечта.

Такие вещи часто не открывают. Но я не теряю надежду.

Малоизвестные секреты

1. Вы можете использовать любой язык, какой пожелаете

Раньше под созданием приложений подразумевалось создание десктопного софта. А в десктопном софте есть большой уклон в сторону написания приложений на том же языке, что и операционная система. Так десять лет назад, написание софта в целом означало написание софта на C. В конце концов традиция эволюционировала: приложения не должны быть написаны на необычных языках. И эта традиция так долго развивалась, что нетехнические люди, вроде менеджеров и венчурных капиталистов, тоже это выучили.

Серверный софт уничтожает эту модель полностью. С серверным софтом вы можете взять любой язык, какой пожелаете. Почти никто ещё этого не понимает (особенно менеджеры и венчурные капиталисты). Но некоторые хакеры понимают это, вот почему мы слышали про такие indy-языки как Perl и Python. Мы не слышим про Perl и Python, потому что люди используют их для написания приложений для Windows.

Что это означает для нас, людей заинтересованных в проектировании языков программирования, что есть потенциальная аудитория для нашей работы.

2. Скорость приходит от профайлеров

Разработчики языка или, по крайней мере, его реализаторы любят писать компиляторы, которые генерируют быстрый код. Но я думаю, что не это делает языки быстрыми для пользователей. Кнут давно заметил, что скорость зависит всего от нескольких узких мест. И любой, кто пытался ускорить программу знает, что вы не сможете угадать, где узкое

место. Профайлер вот ответ.

Разработчики языка решают не ту проблему. Пользователям не нужно, чтобы бенчмарки работали быстро. Им нужен язык, который может показать какие части их программы должны быть переписаны. В этот момент скорость нужна на практике. Так может будет лучше, если реализаторы языка потратят половину времени, которое они тратят на оптимизацию компилятора, и потратят его на написание хорошего профайлера.

3. Вам нужно приложение, которое заставляет ваш язык развиваться

Может это не истина в последней инстанции, но кажется лучшие языки развивались вместе с приложениями, в которых их использовали. C был написан людьми, которым нужно было системное программирование. Lisp был разработан отчасти для символьного дифференцирования, Маккарти так не терпелось начать, что он начал писать программы дифференцирования даже в первом документе о Lisp'е в 1960 году.

Это особенно хорошо, если ваше приложение решает некоторые новые проблемы. Это подталкивает ваш язык иметь новые возможности, которые нужны программистам. Лично мне интересно писать язык, который будет хорош для серверных приложений.

[Во время дискуссии Гай Стил также высказал эту мысль, добавив, что приложение не должно состоять из написания компилятора для вашего языка, если только ваш язык не предназначен для написания компиляторов.]

4. Язык должен быть подходящим для написания одноразовых программ.

Вы знаете, что значит одноразовая программа: это когда вам нужно быстро решить какую-то ограниченную задачу. Я полагаю, что если вы посмотрите вокруг, то обнаружите множество серьёзных программ, которые начинались как одноразовые. Я не удивлюсь, если большинство программ начинались как одноразовые. Таким образом, если вы хотите создать язык, который будет подходящим для написания софта в целом, то он должен и подходить для написания одноразовых программ, потому что это начальная стадия многих программ.

5. Синтаксис связан с семантикой

Традиционно считается, что синтаксис и семантика — вещи сильно отличающиеся. Может это прозвучит шокирующе, но это не так. Я думаю, что то, что вы хотите получить в вашей программе связано с тем, как вы это выражаете.

Недавно я говорил с Робертом Моррисом, и он заметил, что перегрузка операторов это большой плюс в победу языков с инфиксным синтаксисом. В языках с префиксным синтаксисом любая функция, которую вы определяете фактически является оператором. Если вы хотите сложить новый тип числа, который вы выдумали, вы можете просто определить новую функцию для его добавления. Если вы сделаете так в языке с инфиксным синтаксисом, вы увидите, что между использованием перегруженного оператора и вызовом функции существует большая разница.

Идеи, которые со временем возвращаются

1. Новые языки программирования

Оглядываясь в 1970-е, было модно разрабатывать новые языки программирования. Сейчас это не так. Но я полагаю, что серверный

софт снова вернёт моду на создание новых языков. С серверным софтом вы можете использовать любой язык, какой пожелаете, так если кто-нибудь создаст язык, который будет казаться лучше остальных, то будут люди, которые решатся его использовать.

2. Разделение времени

Ричард Келси подал эту идею, время которой снова пришло, и я полностью её поддерживаю. Моё предположение (и Microsoft тоже) многие вычисления переместятся с десктопа на удалённые сервера. Другими словами, разделение времени вернулось. Я думаю понадобится поддержка этого на уровне языка. Например, Ричард и Джонатан Ривз проделали много работы для внедрения планирования процесса в Scheme 48.

3. Эффективность

Недавно казалось, что компьютеры уже достаточно быстры. Всё больше и больше мы слышим про байт-код, что по крайней мере, для меня означает, что у нас есть мощность в запасе. Но я думаю, что с серверным софтом, у нас её нет. Кто-то должен будет заплатить за серверы, на которых работает софт, и число пользователей, которых сервер может выдержать в расчёте на одну машину, будет делителем их капитальных затрат.

Думаю, что эффективность будет иметь значение, по крайней мере в узких местах вычислений. Особенно важным это будет для операций ввода-вывода, потому что серверные приложения производят множество таких операций.

В конце концов может оказаться, что байт-код это не выход. Sun и Microsoft в данный момент похоже сражаются лицом к лицу на байт-код поле. Но они это делают, потому что байт-код это удобное место для

встраивания себя в процесс, а не потому что байт-код сам по себе хорошая идея. Может оказаться, что вся эта битва пройдёт незамеченной. Было бы забавно.

Западни и ловушки

1. Клиенты

Это только предположение, но оно в том, что выиграют только те приложения, которые будут полностью серверными. Проектируя софт, который работает на предположении, что у каждого будет ваш клиент, похоже на создание общества основанного на предположении, что все будут честными. Это было бы определённо удобно, но вам придётся допустить, что этого никогда не будет.

Я думаю будет быстрое увеличение девайсов с доступом в веб, и можно предположить, что они будут поддерживать базовый html и формы. У вас есть браузер на телефоне? Будет ли телефон в вашем PalmPilot'e? У вашего blackberry будет больший экран? Будет ли у вас возможность выйти в интернет со своего gameboy? С ваших часов? Я не знаю. И мне не придётся это узнать, если я сделаю ставку, что всё будет на сервере. Просто гораздо надёжнее иметь все мозги на сервере. .

2. Объектно-ориентированное программирование

Я понимаю, что это противоречивое утверждение, но я не считаю, что ООП это что-то важное. Я думаю, что это подходящая парадигма для конкретных приложений, которым нужны специфические структуры данных, вроде оконных систем, симуляций, САПР-систем. Но я не понимаю, почему она должна быть подходящей для всех программ.

Я думаю, что люди в больших компаниях любят ООП, отчасти, потому что оно даёт многое из того, что выглядит как работа. То, что,

естественно, может быть представлено как, скажем, список целых чисел, теперь может быть представлено как класс со всеми видами строительных лесов, с шумом и суетой.

Другая привлекательная черта ООП в том, что методы дают вам некий эффект функций первого класса. Но это не новость для программистов на Lisp'ах. Когда у вас есть настоящие функции первого класса, вы можете просто использовать их любым способом, который соответствует поставленной задаче вместо того, чтобы проталкивать всё в шаблон из классов и методов,.

Я думаю, что это означает для языкового дизайна, что вы не должны слишком глубоко встраивать ООП в него. Может ответ в том, чтобы предлагать более общие, основополагающие вещи, и позволить людям проектировать любые объектные системы в виде библиотек.

3. Проектирование комитетом

Если ваш язык проектирует комитет, то вы в западне, и не только по причинам, которые всем известны. Всем известно, что комитеты имеют тенденцию создавать комковатый, непоследовательный дизайн языка. Но я думаю, что большая опасность в том, что они не берут на себя риски. Когда во главе один человек, то он берет риски, которые комитет никогда не согласится взять на себя.

Нужно ли рисковать, чтобы создать хороший язык? Многие люди могут подозревать, что проектирование языка — это то, где вы должны держаться довольно близко к традиционной мудрости. Могу поспорить, что это не так. Во всем остальном, что делают люди, вознаграждение пропорционально риску. Так почему же в проектировании языков должно быть по-другому?

Пол Грэм, эссе, проектирование, разработка