Lisp: побеждая посредственность

Летом 1995 года мой друг Роберт Моррис и я основали компанию под названием Viaweb. Наш план заключался в разработке программного обеспечения, которое позволило бы пользователю создавать свой онлайновый магазин. Новизна этого софта на тот момент заключалась в том, что он работал на сервере, используя web-страницы как интерфейс. У многих в то время была та же идея, но, насколько я знаю, Viaweb была первым web-приложением. Идея показалась нам настолько новой, что название для компании мы придумали так, чтобы оно отражало ее суть — Viaweb ("Через web"), подчеркивало, что программа работает в Сети, а не на локальном компьютере.

Другая особенность нашего софта заключалась в том, что он был написан на языке программирования Lisp. Viaweb вначале состоял из двух частей: редактора, написанного на Lisp'e, который использовался для построения сайтов, и системы обработки заказов, написанной на C. Большую часть первой версии составлял Lisp, так как система обработки заказов была маленькая. Позже мы добавили еще два модуля, генератор изображений на C, и программу для администрирования, написанную большей частью на Perl'e.

Это была одна из первых больших пользовательских программ на Lisp, который прежде использовался в основном в университетах и исследовательских лабораториях. Lisp дал нам большое преимущество перед нашими конкурентами, которые применяли менее мощные языки.

секретное оружие

Эрик Реймонд (Eric Raymond) написал статью "Как стать хакером", и в ней, помимо других вещей, он советует будущим хакерам, какие языки они должны выучить. Для начала Реймонд предлагает начать с Python и Java, так как они просты для изучения. Серьезный хакер также выучит C для того, чтобы покопаться в глубинах Unix, и Perl для системного администрирования системы и создания cgi-скриптов. В конце концов, истинный хакер должен подумать об изучении Lisp'a: Lisp стоит выучить ради того глубокого прозрения, которое вы приобретете в результате; этот опыт сделает вас лучше как программиста до конца ваших дней, даже если сам Lisp вы практически не будете использовать.

Это тот же аргумент, который мы постоянно слышим в пользу изучения латыни. Вас не возьмут на работу благодаря лишь ее знанию, разве что преподавателем классической литературы, но это натренирует ваш ум, и вы станете лучше писать на тех языках, которые вы хотите использовать, например, на английском.

Однако, секундочку. А ведь эта метафора не распространяется так далеко. Причина, по которой вы не получите работу с латынью, заключается в том, что на ней никто не говорит. Если вы будете писать

на латыни, вас никто не поймет. Но Lisp — это компьютерный язык, а компьютер говорит на том языке, на котором вы, программист, прикажете ему говорить.

Значит, если Lisp, по словам Реймонда, сделает вас лучше как программиста, почему бы его не использовать? Мне кажется, что если бы художнику предложили кисть, которая позволила бы ему стать еще более хорошим художником, разве не захотел бы он пользоваться ею при написании всех картин, не правда ли?

Я не пытаюсь высмеять Эрика Реймонда. В целом, его совет хорош. То, что он говорит о Lisp'е, — общепринятое мнение: Lisp усовершенствует ваши профессиональные качества, но тем не менее вы не станете его использовать.
Но почему нет? Языки программирования — это, в конце концов, просто инструменты. И если Lisp способствует написанию лучших программ, то следует его использовать. Ну а если нет, то кому он тогда нужен?

Это не только теоретический вопрос. Программное обеспечение — бизнес с жесткой конкуренцией, бизнес, склонный к естественным монополиям. Компания, в которой пишут софт быстрее и лучше, при прочих равных условиях вытеснит конкурентов.
Когда вы основываете компанию, то особенно остро ощущаете это. Либо вы разбогатеете, либо не получите ничего. Если в вашем начинании вы поставите не на ту технологию, ваши конкуренты не оставят от вас и мокрого места.

Мы с Робертом оба хорошо знали Lisp, и у нас не было причин не доверять своим инстинктам и не воспользоваться им. Мы знали, что все остальные пишут на C++ или Perl. Но мы также знали, что это ничего не значит. Если выбирать технологию по тому же принципу, то операционной системой была бы Windows. При выборе технологии нужно обращать внимание не на то, что делают остальные, нужно выбирать то, что будет лучше работать. Это особенно важно для компании-новичка. В большой компании можно делать то же, что делают другие большие компании. Но начинающая компания не может себе позволить повторять действия других таких же новичков. Я думаю, что немногие понимают это, даже среди тех, кто открывают новый бизнес.

Средняя компания большого уровня растет примерно на 10 процентов в год, так что если вы управляете такой компанией и поступаете так, как поступает средний руководитель этого сектора, то можете ожидать такого же прироста — примерно 10 процентов в год.

Конечно, то же самое справедливо и для новых компаний — если вы поступаете так, как поступает средняя компания, следует ожидать среднего результата. Проблема в том, что со средним уровнем отдачи на начальном этапе вы вылетите из бизнеса. Уровень выживаемости среди новых компаний меньше 50 процентов. Так что если вы основали новое

дело, лучше делать что-то необычное. Иначе вы в беде.
В 1995 году мы знали то, что, как мне кажется, наши конкуренты не понимали, а многие не понимают даже сейчас: если вы создаете программное обеспечение, которое будет работать на сервере, вы вольны использовать любой язык, который хотите. В деле создания программ для десктопа присутствует тенденция разрабатывать программы на языке операционной системы. Десять лет тому назад "писать программы" значило "писать на C". Но для веб-приложений вы вольны выбирать тот язык, который хотите, особенно если у вас есть исходный код языка и операционной системы.

Однако эта новая свобода — палка о двух концах. Теперь, когда вы можете использовать любой язык, нужно думать о том, какой же язык выбрать. Компании, которые делают вид, что ничего не изменилось, рискуют обнаружить, что их соперники так не считают.
Если можно использовать любой язык, какой выбрать? Мы выбрали Lisp. Было ясно, что быстрая разработка важна на этом рынке. Все мы начинали с нуля, поэтому компания, которая добавляет новую функциональность быстрее других, будет иметь серьезное преимущество.

Мы знали, что Lisp — действительно очень хороший язык для быстрого создания программ, а серверные приложения еще сильнее увеличивают эффект быстрой разработки, потому что можно использовать программу через минуту после того, как она написана. Если другие компании не хотели использовать Lisp, это было лишь преимуществом для нас. Lisp мог дать нам технологическое превосходство — а нам было важно любое доступное подспорье. Когда мы основали Viaweb, у нас не было опыта в бизнесе. Мы ничего не знали о маркетинге, о том, как нанимать людей, как зарабатывать деньги или привлекать клиентов. Ни у кого из нас не было прежде того, что можно назвать настоящей работой. Единственное, что мы умели — это писать программы. Мы надеялись, что это нас выручит. Мы были готовы воспользоваться любой возможностью получить преимущество в софтверном бизнесе.

Использование Lisp'а можно было назвать экспериментом. Гипотеза заключалась в том, что если мы будем писать на Lisp'е, мы сможем расширять функциональность быстрее конкурентов, и наш продукт будет способен на трюки, недоступные им. А так как Lisp — язык очень высокого уровня, то нам не будет нужна большая команда разработчиков — соответственно и затраты будут ниже.

В этом случае мы сможем предложить лучший продукт за меньшие деньги, и при этом получить прибыль. При оптимальном развитии событий мы должны были привлечь всех клиентов, не оставив никого конкурентам, которым пришлось бы в конце концов уйти из бизнеса. По крайней мере, мы надеялись, что так произойдет.
Каковы же были результаты эксперимента? К удивлению, наша гипотеза себя оправдала. Со временем у нас было много конкурентов, порядка

двадцати — тридцати, но ни у одного из них не было продукта, способного соперничать с нашим. У нас была WYSIWYG-программа для создания онлайновых магазинов, которая работала на сервере, и тем не менее, выглядела как обычное приложение. У наших соперников были CGI скрипты.

Мы всегда были впереди конкурентов по функциональности. Иногда, в порыве отчаяния, они пытались добавить новую функцию, которой не было у нас. Однако с Lisp'ом наш цикл разработки был настолько коротким, что мы могли воспроизвести аналогичную возможность в течение дня или двух после того, как конкурент объявлял о ней в пресс-релизе. К тому моменту, когда журналисты, занимающиеся пресс-релизом, звонили нам, у нас было все то же самое. Должно быть, нашим конкурентам казалось, что мы владели неким секретным оружием, например перехватывали их закрытую переписку или что-то в этом роде. На самом деле у нас было секретное оружие, но гораздо более простое, чем могло показаться. Не было никакой утечки информации. Просто мы могли разрабатывать наш продукт быстрее, чем это казалось возможным. Когда мне было девять, мне в руки попала книга Фредерика Форсайта, "День Шакала". Главный герой книги — киллер, которого наняли для того, чтобы убить президента Франции. Убийце нужно было пробраться мимо полиции, чтобы попасть в квартиру, откуда открывается вид на дорогу, по которой проходит маршрут президента. Киллер прошел мимо полиции на костылях, одетый стариком, и его никто не заподозрил.

Это напоминает наше "секретное оружие". Мы писали наш продукт на причудливом языке со странным синтаксисом, полным скобок, на языке, популярном в сфере Искусственного Интеллекта. Годами такое определение Lisp'а вызывало у меня раздражение. Но теперь это работало в нашу пользу.

В бизнесе нет ничего более ценного, чем техническое преимущество, которого не понимают ваши конкуренты. В бизнесе, как и на войне, сюрприз стоит столько же, сколько сила.

Я слегка смущен, но признаюсь, что никогда открыто не говорил о Lisp'е в то время, когда мы работали над Viaweb. Мы никогда не упоминали его в прессе, и если бы вы искали следы Lisp'а на нашем сайте, все, что вы бы нашли, это названия двух моих книг и мою биографию. И это не случайность. Начинающая компания должна давать своим конкурентам настолько мало информации, насколько это возможно. И если они не знали, на каком языке был написан наш софт, или им было все равно, я предпочел оставить все как есть.

Роберт Моррис говорит, что мне не нужно было скрытничать, потому что даже если бы наши конкуренты узнали, что мы используем Lisp, они бы не поняли, почему: "Если бы они были настолько умными, чтобы понять это, они бы уже писали на Lisp'е." Лучше всего понимали нашу

технологию наши клиенты. Им было все равно, на каком языке был написана наша программа, но они отметили, что работает он на самом деле хорошо. Программа Viaweb позволяла им создавать замечательно выглядящие интернет-магазины буквально за минуты. Таким образом, у нас появлялось все больше пользователей, в основном за счет распространения личного мнения наших клиентов. К концу 1996 у нас было примерно 70 онлайновых магазинов. К концу 1996 их было 500. Шесть месяцев спустя, когда нас приобрел Yahoo, у нас было 1070 пользователей. Сегодня, под именем Yahoo Store, этот продукт продолжает господствовать на своем рынке. Это одна из наиболее прибыльных частей Yahoo, и магазины, построенные с его помощью, — это основа Yahoo Shopping. Я покинул Yahoo в 1999, и поэтому не знаю точно, сколько у них пользователей сейчас, но последние цифры, которые я слышал, — это около 14 тысяч пользователей. Люди спрашивают меня, используется ли Lisp сейчас в Yahoo Store. Да, весь Lisp код все еще там. В Yahoo на серверах работает софт, написанный на всех пяти языках, которые советует хакерам Эрик Реймонд.

Парадокс Блаба

Что же в Lisp'е такого прекрасного? Если он такой замечательный, почему его не используют все? Казалось бы, риторические вопросы, но на самом деле на них есть прямые ответы. Lisp настолько хорош не тем, что в нем есть некое волшебное качество, видимое только его приверженцам, а тем, что он — самый мощный язык программирования из существующих. И причина того, что все вокруг пишут не на Lisp'е, заключается в том, что выбор языка программирования — вопрос не только технологии, но также и привычки, а ничто не меняется так медленно, как привычки. Конечно, оба эти тезиса требуют разъяснений.

Я начну с шокирующего утверждения: языки программирования отличаются друг от друга своей мощностью. По крайней мере мало кто будет спорить, что высокоуровневые языки более мощные, чем машинный язык. Большинство программистов согласятся, что, как правило, программировать стоит не на машинном языке, а на каком-нибудь языке высокого уровня, переводя программу в машинный код с помощью компилятора. Сейчас эта идея получила даже аппаратное воплощение — с восьмидесятых годов команды процессоров разрабатываются скорее для компиляторов, чем для программистов. Каждый знает, что писать всю программу вручную на машинном языке — ошибочно. Но гораздо реже понимают то, что существует и более общий принцип: при наличии выбора из нескольких языков ошибочно программировать на чем-то, кроме самого мощного, если на выбор не влияют другие причины.

Все языки одинаково мощные, если рассматривать их с точки зрения эквивалентности машине Тьюринга, но это не та мощь, которая важна программисту. (Никто ведь не хотел бы программировать машину

Тьюринга). Мощь языка, в которой заинтересован программист, возможно, трудно определить формальными методами, однако одно из объяснений этого понятия заключается в свойствах, которые в менее мощном языке можно получить, только написав на нем интерпретатор для более мощного языка. Если в языке A есть оператор для удаления пробелов из строк, а в языке B его нет, это не делает A более мощным, чем B, так как в B можно написать процедуру, которая делала бы это.

Но, скажем, если язык A поддерживает рекурсию, а B — нет, это нечто, что нельзя исправить написанием библиотечных функций. Есть много исключений из этого правила. Если вы пишете программу, которая должна тесно взаимодействовать с программой, написанной на определенном языке, возможно, окажется разумным писать новую программу на том же языке.

Если вы пишете программу, которая должна делать что-то очень простое, вроде численной обработки больших массивов данных или манипуляций с битами, можно использовать язык не самого высокого уровня абстракции, тем более что программа будет слегка быстрее.

Если вы пишете короткую программу, которую используете один раз и выбросите прочь, возможно, следует использовать тот язык, который имеет лучшие библиотечные функции для данной задачи. Но в целом для программного обеспечения нужно использовать самый мощный (и приемлемо эффективный) язык из всех доступных. Отличный от этого выбор — это ошибка такого же рода, как упор на программирование в машинных кодах, хотя и с меньшими негативными последствиями.

Понятно, что уровень машинного языка очень низок. А высокоуровневые языки часто рассматриваются как одинаковые, по-крайней мере, так принято считать. Но это не так. Технический термин "язык программирования высокого уровня" не обозначает ничего определенного. Не существует четкой границы между множеством "машинных" языков с одной стороны, и множеством "высокоуровневых" с другой. Языки распределены в континууме (возможно, не просто континуум, а некая структура, уменьшающяяся кверху; важна здесь не форма, а сама идея о том, что существует по крайней мере частичный порядок) абстрактности, начиная от самых мощных "языков высокого уровня" вниз к "машинным языкам", которые, в свою очередь, тоже отличаются друг от друга по мощности.

Возьмем Cobol. Cobol — язык высокого уровня, так как компилируется в машинный язык. Но станет ли кто-нибудь утверждать, что по мощности Cobol эквивалентен, скажем, Python'у? Возможно, он ближе к машинному языку, чем Python.

А как насчет Perl четвертой версии? В Perl 5 в язык были добавлены лексические замыкания (lexical closures). Большинство Perl хакеров

согласятся, что Perl 5 мощнее, чем Perl 4. Но раз вы это признали, вы признали, что один высокоуровневый язык может быть мощнее другого. Из этого неизбежно следует, что использовать нужно самый мощный язык.

Впрочем, из этого утверждения редко делается вывод. Программисты старше определенного возраста редко меняют язык по своей воле. Они будут считать достаточно хорошим тот язык, к которому привыкли.

Программисты очень привязываются к своим любимым языкам, а я не хочу оскорбить ничьи чувства, поэтому я объясню свою позицию, используя гипотетический язык с названием Блаб.

Блаб попадает в середину континуума абстрактности. Это не самый мощный язык, но он мощнее, чем Cobol или машинный язык. И на самом деле, наш гипотетический программист на Блабе не будет использовать ни Cobol, ни машинный код. Для машинных кодов есть компиляторы. Что же касается Cobol'а, наш программист не знает, как на этом языке вообще что-то можно сделать. В Cobol'е даже нет некой возможности X, присутствующей в Блабе.

Когда наш гипотетический Блаб-программист смотрит вниз на континуум мощности языков, он знает, что смотрит вниз. Менее мощные, чем Блаб, языки явно менее мощны, так как в них нет некой особенности, к которой привык программист. Но когда он смотрит в другом направлении, вверх, он не осознает, что смотрит вверх. То, что он видит, — это просто "странные" языки. Возможно, он считает их одинаковыми с Блабом по мощности, но со всяческими сложными штучками. Блаба для нашего программиста вполне достаточно, так как он думает на Блабе.

Когда мы поменяем точку обзора программиста, используя любой язык программирования выше по континууму мощности, мы обнаружим, что теперь программист смотрит на Блаб сверху вниз. "Как же можно что-то сделать, используя Блаб? В нем отсутствует даже конструкция Y!"

Используя метод индукции, приходишь к выводу, что только те программисты, которые понимают самый мощный язык, в состоянии осознать полную картину разницы в мощности между различными языками (видимо, именно это имел ввиду Эрик Реймонд, когда говорил о том, что Lisp сделает вас лучше как программиста). Следуя парадоксу Блаба, нельзя доверять мнению других: другие программисты довольны тем языком, который используют, потому что этот язык определяет способ их программистского мышления.

Я знаю это из своего опыта, когда учился в старших классах школы и писал программы на Бейсике. Этот язык не поддерживал даже рекурсию. Трудно представить написание программ без рекурсии, но в то время мне это не нужно было. Я думал на Бейсике. Я был спец. Мастер всего, что изучил.

Пять языков, которые советует хакерам Эрик Реймонд, находятся в разных точках континуума мощности, и то, где они находятся относительно друг друга, — тонкий вопрос. Я скажу, что Lisp находится на вершине континуума. И чтобы поддержать это утверждение, я скажу о том, чего мне не хватает, когда я смотрю на остальные пять языков. Как же можно что-то сделать с ними, думаю я, без свойства Z? И самое большое Z — это макросы. (Рассматривать макросы как отдельное свойство — это немного неправильно. На практике их польза увеличивается такими свойствами Lisp'а, как лексические замыкания и частичная параметризация (rest parameters).

Во многих языках есть что-то, называющееся макросом. Но макросы в Lisp'е уникальны. То, что делают макросы имеет отношение, верите вы или нет, к скобкам. Создатели Lisp'а добавили все эти скобки в язык не для того, чтобы отличаться от других. Скобки в Lisp'е имеют особый смысл, они — внешнее свидетельство фундаментальной разницы между Lisp'ом и другими языками. Программа на Lisp'е состоит из данных. И не в том тривиальном значении, что исходные файлы содержат символы, а строки — один из типов данных, поддерживаемых языком. После прочтения программы парсером Lisp код состоит из готового к использованию дерева структур данных.

Дело не в том, что в Lisp'е странный синтаксис, скорее, его нет вообще. Программы пишутся в готовых синтаксических деревьях, которые в других языках генерируются парсером во время разбора исходного текста. Эти синтаксические деревья в Lisp'е полностью доступны вашим программам, и вы можете писать программы, которые изменяют эти деревья. В Lisp'е подобные программы называются макросы. Это программы, которые пишут программы.

Программы, которые пишут программы? И когда же такое может понадобиться?

Не очень часто, если вы думаете на Cobol'е. И постоянно, если вы думаете на Lisp'е. Было бы удобно, если бы я дал пример мощного макроса и сказал бы: "Вот! Смотрите!". Но если бы я и привел пример, для того, кто не знает Lisp, он выглядел бы не более чем белиберда. Рамки данной статьи не позволяют изложить все необходимое для понимания подобного примера. В книге Ansi Common Lisp я старался излагать материал как можно быстрее, но даже так я не добрался до макросов раньше страницы 160. Однако мне кажется, что я могу дать убедительный аргумент. Исходный текст редактора Viaweb на 20-25 процентов состоял из макросов. Макросы сложнее писать, чем обычные функции Lisp'а, и считается дурным тоном использовать их там, где можно без них обойтись. Поэтому каждый макрос в той программе был необходим. Это значит, что примерно 20-25 процентов кода в программе делают то, что нельзя просто сделать на других языках.

Как бы скептически ни относился Блаб-программист к моим заявлениям

о таинственной мощи Lisp'а, это должно его заинтересовать. Мы не писали этот код для своего собственного развлечения. Мы были маленькой компанией, и программировали так, как только могли, чтобы возвести технологический барьер между нами и нашими конкурентами.

Пытливый читатель может задаться вопросом, а нет ли здесь взаимосвязи? Некоторая большая часть кода делала нечто, что очень сложно сделать на других языках. Получившееся в результате программное обеспечение делало то, что программное обеспечение наших соперников делать не могло. Возможно, между этими фактами есть связь. Я советую вам подумать в этом направлении. Возможно, это все не просто старческие бредни.

Айкидо для начинающих компаний

Однако я не думаю, что смогу убедить кого-нибудь (старше 25) выучить Lisp.
Цель этой статьи — не изменить чье-то решение, а вдохновить тех, кто уже заинтересован в использовании Lisp'а — тех, кто знает, что Lisp — это мощный язык, но беспокоятся из-за того, что Lisp мало используется. В случае конкуренции это преимущество. Мощь Lisp'а умножается тем фактом, что ваши конкуренты этого не понимают.

Если вы раздумываете над использованием Lisp'а в новом бизнесе, не стоит беспокоиться о том, что большинство не понимает этот язык. Вам нужно надеяться на то, что так все и останется. А так оно скорее всего и будет. Большинство довольны тем, что они используют — это природа языков программирования. Компьютерное железо изменяется настолько быстрее личных привычек, что практика программирования обычно отстает от процессора на десять — двадцать лет. В таких местах как MIT писали на высокоуровневых языках уже в начале 60-х, но многие компании продолжали писать на машинных кодах вплоть до 80-х. Бьюсь об заклад, что многие продолжали писать на машинных кодах до тех пор, пока процессор, словно бармен, собирающийся закрыть бар и пойти домой, не выдворил их прочь, переключившись на набор команд RISC.

Обычно технология меняется быстро. Однако с языками программирования все по-другому — они не просто технология, они воплощают собой способ мышления программистов. Это наполовину технология, наполовину религия. В результате сравнения языков программирования принимают форму либо религиозных войн, либо университетских учебников, настолько нейтральных, что они похожи на труды по антропологии. Те, кто желают спокойствия, избегают столь острой темы. Однако вопрос религиозен лишь наполовину, этот вопрос изучить не помешает, особенно если вы собираетесь создавать новый язык программирования. Средний язык — это язык, которым пользуется средний программист. Движется он медленно, как айсберг. "Сборка мусора" (Garbage Collection), впервые введенная в употребление в Lisp'е примерно в 1960 году, сейчас повсеместно признается хорошей штукой.

Динамическая типизация также становится все более популярной. Лексические замыкания, введенные в употребление Lisp'ом в начале семидесятых, сейчас едва видны на экране радара. Макросы, появившиеся в Lisp'е в середине шестидесятых, до сих пор — терра инкогнита.

Очевидно, что средний язык обладает мощной инерцией. Я не предлагаю вам сопротивляться этой силе. Я предлагаю совершенно обратное — подобно практикующему айкидо, воспользоваться этой силой против своих противников.

Если вы работаете в большой компании, это может быть не так просто. Вам будет трудно убедить своего косного босса позволить вам программировать на Lisp'е, особенно если тот только что прочитал в газете о каком-то языке, который, как и Ада двадцать лет тому назад, готов завоевать мир. Но если вы работаете в начинающей компании, в которой еще нет такого босса, вы можете, как это сделали мы, извлечь выгоду из парадокса Блаба: использовать технологию, которой ваши конкуренты, неотрывно привязанные к своему среднему языку, не смогут ничего противопоставить.

Если уж довелось работать в начинающей компании, есть неплохая подсказка, как оценивать конкурентов. Прочитайте список их вакансий. Все другое на их сайте может быть сплошным сочинительством, и лишь описания требуемых специалистов должны быть точными, иначе они наймут не того кандидата, что им нужен.

За те годы, что я проработал в Viaweb, я прочитал множество объявлений о найме на работу. Примерно каждый месяц появлялся новый конкурент. Первое, что я делал после того, как проверял, доступна ли онлайновая демонстрация работы их программы, — смотрел список их вакансий. Через пару лет я научился отличать опасных конкурентов от неопасных. Чем больше отдавало IT-мэйнстримом от описания требуемых кандидатур, тем менее опасна была компания. Самыми безопасными были те, кому требовались специалисты по Oracle. О таких не стоило беспокоиться. Также мы были спокойны, если требовались разработчики на C++ или Java.

Если требовались программисты на Perl или Python, это уже было слегка пугающе — это значило, что компанией или, по крайней мере, ее технической частью заправляли настоящие хакеры. Если бы я когда-нибудь увидел объявление о найме на работу Lisp-хакеров, я бы обеспокоился не на шутку.

Пол Грейхем (Paul Graham), перевод Юрия Лейкинда.
В основе этой статьи — лекция, прочитанная в Кембридже на симпозиуме разработчиков, проведенном Franz, Inc. 25-го марта 2001-го года.

Оригинал статьи можно найти по адресу http://www.paulgraham.com/avg.html.