*Все течет, все меняется у нас на глазах, все обретает новый облик, и, однако, перемен ни в чем нет.*

*© Жан Бодрийар*

В своем выступлении "Stop Writing Dead Programs" Джек Рашер, играя роль своеобразного компьютерного палеоантрополога, вместе со зрителем путешествует от сотворения компьютеров до нашего времени как «Доктор Кто», исследуя эволюционное и революционное развитие программирования. Автор затрагивает темы сложности, интерактивности, визуализации, сохранения обратной совместимости, а главное изменения подходов к созданию программ. Свой рассказ о программах живых и мертвых Джек начинает с воспоминаний о молодости: «В начале была стопка проткнутых перфокарт, и было это сложно, и сложность была in a dumb way!». Это была пакетная обработка с действительно долгим циклом обратной связи – проверить результаты работы программы можно было только через 45 минут. Кроме того, программы всегда начинались с пустого листа и были таким грязно-белым ящиком для входящих и выходящих данных. Время шло, а программисты так и не могли менять программы во время их выполнения, ошибки обрушивали программы, а компиляция и запуск были мало между с собой связаны. Но погодите! Оказывается, большинство программистов остаются заложниками «дедовских» методов программирования и пишут свой код точно так же и по сей день; а ведь сколько килограммов терабайт с тех времен утекло... После нескольких минут восхваления Clojure и Erlang/OTP (впрочем, я согласен, что хвалить есть за что) автор переходит к проблеме репрезентации программ. Он обнаруживает, как многие вещи, что изначально были удобны, стали рудиментами бородатых годов: бессмертный 80-столбцовый терминал прошел весь путь от перфокарт до эмуляторов, написанных на Electron, а раскладка vi, оказалась такой, какой оказалась, лишь потому, что клавиатура у Билла Джоя была такой, а не другой, а текстовое представление все еще остается только лишь текстовым. Что ж, если бы Джек знал анекдот про кончики у сосисок и сковородку (<https://www.anekdot.ru/id/1062438/>), он бы его непременно вспомнил! Но такие, казалось бы, случайные и не связанные с настоящим решения все еще накладывают свой отпечаток на современный стиль программирования. Несуразное прошлое крепко держит нас за горло, но Джек Рашер готов вытащить нас из плена и немедленно находит решение. А решение состоит в интерактивном программировании (как Лисп из 60-х) и визуализации кода и вывода программ (как Oracle Designer из 80-х). Но обо всём по порядку.

С моей точки зрения, идеи, которые несет Джек Рашер, несколько противоречивы. Хотя начало, посвященное тому, что мы делаем много глупостей по историческим причинам (и не только в программировании, вспомним хотя бы, как ширина боковых двигателей на космических шаттлах восходит к ширине крупа двух древнеримским лошадей!) действительно сильное и комичное, вторая половина, похоже, полностью игнорирует производительность и, честно говоря, все, что находится за пределами веб-царства Джека Рашера. Причина, по которой программы запускаются и выполняются до завершения, заключается в том, что именно это делают процессоры. Вы можете абстрагировать это, но теперь вы просто используете код пользовательского уровня и делаете его неотлаживаемой и неизменяемой функцией языка. Конечно, функциональные языки выглядят аккуратно, но где ваши переменные? Как они размещаются в памяти? Собираетесь ли вы получать все из кэша или холодной оперативной памяти? Легко разрушить «дедовский» способ программирования, если вы не соблюдаете аппаратные ограничения. Думает ли автор, что люди использовали перфокарты потому, что им этого хотелось? Это была просто реальность того, что в то время оборудование было недостаточно производительным. Все языки, которые он упоминает, по-прежнему основаны на старом низкоуровневом коде, который на самом глубоком уровне процессора все еще является машинным кодом…

> ВЫ ПРОЧИТАЛИ ПРОБНУЮ ВЕРСИЮ ЭССЕ, ПРОДОЛЖЕНИЕ ПО ССЫЛКЕ: <https://pastebin.com/c31CEvw8>

Простите за простыню текста, Дорогие Читатели. Если вы не хотите читать полностью, ознакомьтесь с первыми двумя и последним абзацем.

Разработчики когда-то усложняли себе жизнь? Или всё-таки они жили в условиях крайних технологических ограничений? Конечно, мне бы хотелось, чтобы язык всегда нативно поддерживал отладку, и я считаю, что замечательно иметь возможность видеть работу кода без необходимости каждый раз его перекомпилировать. Тем не менее, код все равно должен работать на оборудовании – отсюда и ограничения. Компьютеры представляют собой пакетно обрабатывающие процессоры. Каждую программе хотя бы время от времени придется запускать «с чистого листа». И именно по этой причине мы до сих пор пишем программы. Именно поэтому их называют «программами». Если все, что вас волнует, — это результат для каких-то конкретных данных, тогда хоть делайте все вручную, хоть используйте калькулятор, электронную таблицу или блокнот. «Живые» языки программирования привлекательны, но не всегда практичны. Эти типы языков требуют больших затрат и не подходят для некоторых приложений. Лучшим примером этого являются операционные системы. Некоторые языки просто лучше справляются с некоторыми задачами, чем другие.

На этой ноте и перейдем к визуальному программированию. Кажется, что самая большая проблема со всеми визуальными программами заключается в том, что они отлично подходят для анализа данных или теоретических алгоритмов, но гораздо меньше для вашего заурядного «корпоративного программирования», такого как (веб)сервисы. При создании сервисов почти все программирование связано с созданием модели реального мира, а не столько с визуализацией и преобразованием данных. Все эти примеры графиков, таблиц, потоков и т.д. очень хорошо работают для Data Science (поэтому там так популярны такие вещи, как Jupyter), но они не очень хорошо обобщаются для моделирования предметной области. Вам нужен правильный инструмент для работы, но специализированный графический инструмент действительно хорош только для решения проблем, которые можно смоделировать графически. Но печальная реальность такова, что сложные проблемы сложны на любом языке. Как сказал Ричард О'Киф, «невозможно прийти к простой системе, добавляя простоту к сложной системе». И попытки писать что-либо на визуальном языке обычно проваливаются, когда вы приближаетесь к более высоким уровням сложности. Существует множество языков графического программирования, можно даже назвать Photoshop языком программирования. Проблема в том, что существует множество экспериментальных визуальных языков, но ни один из них на самом деле не создает ничего «нового». Они тратят свое время, пытаясь скопировать функциональность C. Прекратите копировать C в своем визуальном языке! Екклесиаст, глава 1:

Что было, то и будет; и что делалось, то и будет делаться, и нет ничего нового под солнцем.

Бывает нечто, о чем говорят: «смотри, вот это новое»; но это было уже в веках, бывших прежде нас.

Нет памяти о прежнем; да и о том, что будет, не останется памяти у тех, которые будут после.

Идея визуального программирования не была проигнорирована, ее пытались использовать снова и снова, но во многих отношениях она терпела неудачу. Клавиатура остается лучшим доступным устройством ввода, и языки программирования построены вокруг этого: программного ввода и выражения алгоритмов. Зрительная кора головного мозга может обрабатывать гораздо больше, но человеческий разум не может выражать идеи быстрее, чем он может их произнести или напечатать. Нам нужны не нетекстовые языки, нам нужны более совершенные инструменты визуализации кода, которые берут существующий текстовый код и аннотируют его в легком для понимания визуальном формате. Кроме того, я хочу, чтобы мое программное обеспечение было легко контролировать версиями. Графики хороши, когда они созданы на основе одной и той же программы и одних и тех же данных. Графики ужасны, когда дело доходит до сравнения двух графиков друг с другом. Все, что не является текстом, плохо различается. Впрочем, существует множество способов создания текстовых форматов, которые различать ещё ужаснее.

Подводя итог, хотелось бы сказать, что это распространенный метод, когда пытаются заставить кого-то заменить свой устаревший старый надежный инструмент на другой, сногсшибательный, с целью заявить, что новый сделает все, и заставить старый инструмент выглядеть так, как будто он вышел из эпохи пещерного человека... Однако люди опробовали все идеи, представленные в выступлении Джека на разных языках. И если вы не понимаете причин, лежащих в основе текущих стандартов, вы упускаете значительную часть общей картины мира разработки ПО. Противоположностью обратной совместимости является полный хаос, и сохранение обратной совместимости в течение длительного времени, кажется, лучше хаоса. Иногда произвольные вещи, которые исторически обусловлены, не являются плохими. Для меня хорошим примером является 80 столбцов. Конечно, сейчас мы можем иметь дело с гораздо более длинными строками, и это дает очевидные преимущества, но постоянство соблюдения фиксированной, довольно узкой ширины столбца означает, что высокозрелые наборы инструментов хорошо работают с ними, будь то такие вещи, как размеры шрифта и разрешения монитора, методы отступов, или даже просто человеческий фактор привыкания к этому (который сохраняется, поскольку каждый новый программист по умолчанию привыкает к тому, к чему привыкли его предшественники), что делает его если не более удобным, то, по крайней мере, более простым в отношении осознания и привыкания. «Я видел дальше других лишь потому, что стоял на плечах гигантов», – писал Исаак Ньютон Роберту Гуку в 1676 году. Возможно, существует какая-то идеальная ширина строки, которую кто-то мог бы придумать. И, возможно, если это произойдет, будет некоторая польза от изменения традиционных 80 столбцов на новинку. Но до тех пор имеет смысл придерживаться этой конвенции именно потому, что она условна. Не чините то, что не сломано. И позиционируя какое-либо решение как обычную историческую ошибку программирования, автор упускает из виду некоторые весьма важные моменты, которые занимают центральное место в разговоре, который якобы касается человеческого фактора. Тем не менее, я все еще ценю общую концепцию, переданную Джеком Рашером, заключающуюся в том, что мы можем найти лучшие способы программирования. «Лучший стимул для бега вперед — иметь позади что-то такое, от чего приходится убегать»!