ПРОГРАММИРОВАНИЕ*, JAVASCRIPT*, JAVA*, C++*, БЛОГ КОМПАНИИ WRIKE

Когда появится следующий большой язык программирования с точки зрения Дарвина

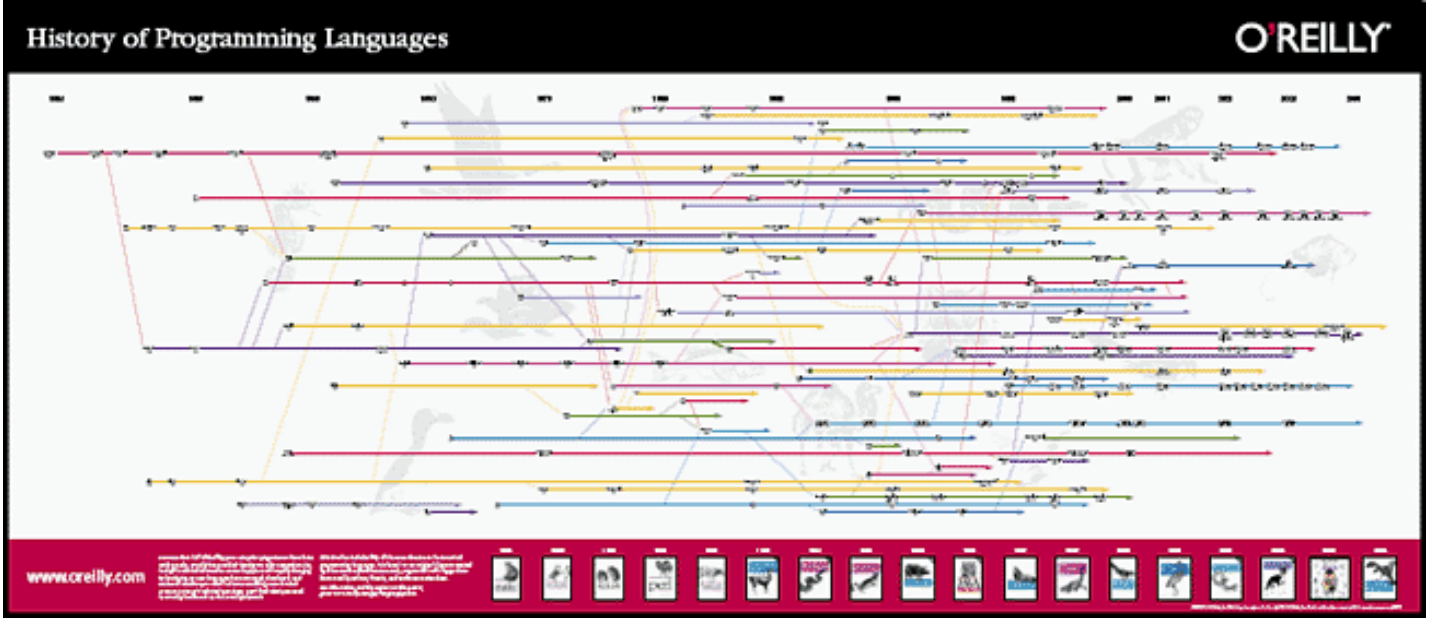
dm_wrike 13 марта 2017 в 13:46

Good news everyone!
Futurama

Следующего большого языка программирования не предвидится. По крайней мере, на то нет причин с точки зрения теории эволюции.

Эволюция работает не только в животном мире, но и в любой подходящей среде. Впервые эта идея получила широкое распространение с выходом книги Ричарда Докинза «Эгоистичный ген» в 1976 году. В ней был введен знакомый каждому термин «мем», как пример эволюции в социальной и культурной среде. Языки программирования тоже эволюционируют. А значит их развитие подчиняется принципам эволюции, на основании которых можно сделать предположение о будущем их развитии.

Для работы эволюционных принципов необходимы следующие механизмы: изменчивость, наследование признаков и естественный отбор. Кажется, что выполнение этих требований для языков программирования достаточно очевидно. Простой пример — диаграмма развития языков программирования, которая, может быть, даже красуется на постере у вас за спиной.



Можно было бы возразить, что создание нового языка программирования — это исключительно заслуга творца. Джеймс Гослинг создал Java, Гвидо Ван Россум — Python, а Керниган и Ричи — C. Однако, на практике они лишь реализуют механизм изменчивости собирая из “генов” языков программирования что-то новое. Никто не в состоянии сам заставить весь мир разрабатывать, например, на ABAP, что, впрочем, не умаляет заслуг авторов.

Для понимания того, когда появится следующий большой язык программирования, стоит ознакомиться с принципом острова. Но прежде давайте рассмотрим применение других принципов эволюции к языкам программирования, хотя бы ради любопытства.

Вымирание и гены

Используя аналогию с устной речью, вымершими будем считать те языки, на которых никто не разрабатывает в настоящий момент.

Даже если компиляторы этих языков существуют, программы на них написанные можно запустить, а некоторые “историки программирования” все еще могут написать на них новый код.

Все знают, что предками языков C (1972) и Pascal (1970) были B (1969) и Algol (1958-68), которые смело можно считать вымершими. Утверждать что язык B забыли, потому что он “старый” — неправильно. Lisp (1958) старше, а его диалекты в ходу до сих пор, как и само название языка. Да и C, несмотря на то что уже не молод, будет с упорством отстаиваться как лучший язык программирования миллионами разработчиков по всему миру.

Посмотрим на пример кода на B из оригинального руководства по языку. Согласитесь, он не так уж и далек от современного C:

```
main() {
   extrn putchar, n, v;
   auto i, c, col, a;

   i = col = 0;
   while(i<n)
      v[i++] = 1;
   while(col<2*n) {
      a = n+1 ;
      c = i = 0;
      while (i<n) {
         c =+ v[i] *10;
         v[i++]  = c%a;
         c =/ a--;
      }

      putchar(c+'0');
      if(!(++col%5))
         putchar(col%50?' ': '*n');
   }
   putchar('*n*n');
}
```

```
v[2000];
n 2000;
```

Есть причина, по которой B и подобные ему языки забыты, — это “ген” структур данных. Единственное что отличает C и Pascal от своих предшественников — наличие конструкций “struct” и “record”. Почему это, казалось бы, небольшое изменение обеспечило живучесть C и Pascal на десятилетия, а их предшественники лишь вскользь упоминаются в университетских курсах? Просто изменилась среда.

Проект «Манхэттен» показал, что возможностей электромеханических табуляторов недостаточно для проведения физических расчетов: необходимых для разработки атомного оружия. О том, как проводились такие вычисления рассказывается в одной из глав книги «Вы, конечно, шутите, мистер Фейнман!». Конечно, именно Пентагон стал заказчиком нового инструмента для проведения вычислений — компьютера, первый из которых увидел свет в 1946 году (ENIAC).

Что нужно для проведения физических расчетов — скаляры, векторы, матрицы и нехитрая программа. Долгое время такого инструментария было достаточно для решения всех насущных задач. Именно это и было реализовано в языке Fortran (1957), и даже в редакции 1977 структуры данных в нем не появились. По тем же принципам строились и другие языки того времени.

Появлению C мы обязаны возникновением нового класса задач — системного программирования. Необходимость создавать операционные системы, в разработке которых требуются “не

математические” структуры данных, привела к тому, что язык закрепился и здравствует. Оказалось, что иметь возможность оперировать произвольными структурами — это очень удобно и позволяет решать любые классы задач, в то время, как языки B или Algol показали свою ограниченность и были забыты. Пожалуй только Fortran сохранил свою актуальность по сей день, и то в 1990 году структуры данных в него все таки добавили.

Конвергентная эволюция

Живые существа в сходных условиях вырабатывают аналогичные свойства адаптации к ним независимо друг от друга. Например, и птицы и летучие мыши имеют крылья. При этом птицы это птицы, а летучие мыши — млекопитающие. Инструмент для полета был реализован ими независимо, но одинаково, поскольку решал одну и ту же задачу.

Европейская и американская школы программирования не склонны заимствовать наработки друг друга, и предпочитают изобретать свои решения самостоятельно. Именно поэтому C и Pascal возникли практически в одно время и реализовали одну и ту же особенность — концепцию структур данных, пусть и немного по-разному.

Аналогичную ситуацию можно проследить между Windows 1.0 (1985) и node.js (2009). В случае Windows 1.0 решалась задача обеспечения одновременной асинхронной работы одного пользователя со многими приложениями, в node.js — многих пользователей с одним приложением использующим асинхронные

операции ввода-вывода. Поскольку обе системы основываются на однопоточной модели исполнения, в них были придуманы одинаковые решения — цикл обработки очереди сообщений. Не удивительно что и та, и другая реализации склонны “зависать” и требовать “ребута”. В node.js концепция watchdog вообще считается нормой и самой собой разумеющейся.

При этом решение той же проблемы на основе вытесняющей многозадачности, лишенное детских проблем “зависаний” всей системы, было реализовано как в операционных системах UNIX (1969) так и в программировании — Erlang (1986). Причины, по которым авторы node.js решили изобрести свой велосипед, возможно были теми же, по которым C появился после Pascal — несовместимость американской и европейской школ программирования.

К слову, в том же 2009 году появился язык программирования Go, которые решает проблему массовой многозадачности гораздо более надежным способом — ”go routines”. Фактически, в нем реализован тот же принцип легковесной многозадачности, что и в Erlang.

Слепая эволюция

Эволюция слепа, и процесс эволюционного развития не предполагает построения “идеальной архитектуры” или проведения “полного рефакторинга”. Человеческий глаз имеет слепое пятно, из-за чего мозг вынужден достраивать недостающие части картинки, и это не обусловлено каким-то разумными

ограничениями. Все можно было сделать гораздо лучше, если бы эволюция “пересмотрела” архитектуру глаза. Однако, эволюция так не работает — она никогда не перерабатывает имеющиеся решения с нуля, но лишь улучшает то, что есть. Подробнее о процессе эволюции зрения рассказано, например, в книге «Что, как и почему мы видим на самом деле».

Так же и язык C. С момента своего появления он изменяется и расширяется, но никогда не перерабатывается полностью. Сейчас мы имеем дело с монстром C/C++. Он содержит все врожденные проблемы управления памятью, неприлично обширный синтаксис, огромное количество библиотек и процесс компиляции, требующий отдельной вычислительной фермы. При этом он полон сил, активно развивается и вообще пышет здоровьем.

Другой пример — Java. Начиная с первой версии, в язык были включены примитивы многопоточного программирования. Конструкция “synchronized”, поддержка блокировок на уровне любого объекта и, простите, “Hashtable”. И вот это представили как решение проблем многопоточного программирования, объявили что Java — “concurrent” и выпустили в свет. Сейчас очевидно, что разработку многопоточного кода нужно основывать на принципах “stateless” и “immutability”, что для “concurrency” нужные специализированные структуры данных — “ConcurrentHashMap”, а блокировки стоит использовать только в крайних или специфичных ситуациях. Увы, фарш невозможно провернуть назад. “Hashtable” останется в Java навсегда, а необходимость рассказывать каждому новому разработчику, почему нужно использовать “StringBuilder”, а не “StringBuffer” — это крест, который несет все

Java-сообщество.

В примере с Java важно, что такие “атавизмы” на самом деле никак не влияют на “живучесть” языка программирования. Все перечисленные проблемы были устранены в .Net изначально, и кому какое до этого дело.

Принцип острова

Новые виды наиболее активно появляются в новой изолированной среде. Такой средой может быть как остров в буквальном смысле, — Ява и ее малый оленёк или Мадагаскар с лемурами, так и озеро — например, Байкал и его прозрачная голомянка. Континент Австралия с населяющими его сумчатыми также является островом в эволюционном смысле.

Попав в независимую среду, виды начинают быстро эволюционировать и образовывать новые виды. Именно принцип острова определяет возникновение мейнстримных языков программирования.

Рассмотрим островное происхождение языка Java. То, что Java стал мейнстримным языком программирования, — по большому счету, случайность. Для этого не было никаких разумных предпосылок. Разработка Java началась в 1991 году (тогда язык назывался Oak). По легенде, основной платформой для исполнения Java-приложений должны были стать кофемолки. В целом это неудивительно, поскольку Java разрабатывалась в недрах Sun Microsystems — “железной” компании, так что

разработка языка для встраиваемых систем там была вполне логичной. Но в 1993 году появился первый браузер Mosaic, и это все изменило.

Появилась совершенно новая, никем не занятая и не освоенная среда, полностью свободная от багажа предыдущего опыта и технологий. Никто не знал что и как в ней делать. И вот в 1995 году технологию Java представили миру. Правда, по нынешним стандартам, технология была так себе. Ирония состоит в том, что Java изначально позиционировалась как клиентский язык программирования, отсюда Applet — технология встраивания в браузеры, окончательно уничтоженный только в 2015, и обреченный с рождения браузер HotJava. Java стал одним из основных языков серверного программирования скорее вопреки задумке авторов. Справедливости ради, на то были разумные основания. В 1990-х, выбирая, на чем разрабатывать серверную логику — на C++ или Perl, любая альтернатива этим двум казалась бы лучшим выбором. А как серверная технология Java оказалась настолько хороша, что уже в 2002 году ее скопировал Microsoft.

Тем временем, в том же 1995 году появился JavaScript. Пользуясь полнейшим провалом Java на стороне клиента, он стал доминирующей технологией разработки клиентских приложения для веба.

Справедливости ради следует добавить что тогда же, в 1995 году появился PHP и занял экологическую нишу быстрой разработки. Вообще, эра веба крайне благоприятно сказалась на развитии

языков на букву “P”.

Острова не только для языков программирования

Конечно веб не остался единственным островом. Среда меняется и новые доминирующие виды на них появляются регулярно. Просто это не языки программирования, а что-то другое.

Хорошим шансом для появления нового массового языка программирования стало появление смартфонов. Но не случилось. Apple по инерции использовала в iPhone (2007) Objective-C (1984), а Google — Java в Android (2008). Становление нового рынка было слишком горячим фронтом противостояния, чтобы отвлекаться на эксперименты (см. «Как поссорились Apple и Google»). Да, конечно есть Swift, но о нем вспомним чуть позже.

Другой островок появился где-то в районе 2009 года, когда все начали добавлять в веб-приложения поддержку WebSocket для обновления данных на странице в реальном времени. Так появились Node.js и Go, и вновь показался на радарах Erlang. Но несмотря на технологическое превосходство Go над Node.js и привычную семантику по сравнению с вычурным Erlang, Go не стал лидером, как, впрочем, и никто из его “коллег”. Островок оказался слишком маленьким и на базу для дальнейшей экспансии не потянул. А совсем недавно там добавился еще один обитатель-конкурент — Elixir (2012), адекватный синтаксис и исполнение в среде Erlang.

Остров интернета вещей — компьютер размером с пуговицу

работающий от солнечной энергии на котором можно запустить Doom, это да. Сейчас в тренде смарт-часы, и это дает шанс добиться популярности новым операционным системам, например Tizen.

Остров машинного обучения в основном населяют C++ и Python, немного Java, иногда Lua (1993) и конечно R (1993). В целом машинное обучение — больше про алгоритмы и специфичные способы обработки данных.

Генная инженерия языков программирования

За время своего развития мейнстримные языки программирования накопили багаж негативных аспектов, которые хочется исправить. Периодически происходят попытки искусственно переломить ситуацию и создать “в пробирке” замену языкам “выросшим на воле”. Однако, нарушения законов эволюции пока не наблюдается. Новые языки программирования конечно добиваются определенных успехов, но только в узких областях, и о полной победе речи не идет. Вот только некоторые примеры.

Apple выпустила Swift на замену Objective-C в 2014. Похоже что дела в этом направлении идут неплохо, но нет причин ждать чего то большего.

В JetBrains устроили локальную революцию с Kotlin в 2011, решив переписать на него IDEA а заодно избавить весь мир от NullPointerException. Пока не помогло.

Google громко анонсировал Dart как “убийцу JavaScript” в том же 2011. Криминала, впрочем, не случилось, как и нативной поддержки Dart в браузерах. В целом Dart и TypeScript (2012) неплохо живут и компилируются в JavaScript, но свергнуть короля веба они оказались не в состоянии, хотя и существенно потеснили его в рамках разработки крупных проектов.

Mozilla решила переписать Firefox на Rust (2010). Распространение системы типов на многопоточность — это интересно, да и все остальное в Rust выглядит основательно. Есть одна загвоздка — на Go худо-бедно уже можно найти живые приложения, а вот от Rust выхлопа пока нет.

Мажорное обновление Python 3, ломающее обратную совместимость, сформировалось в 2006 и называется Python 3000. Первый релиз вышел в 2008. Судя по тому что Python 2 бодр, весел, и все еще с нами, изначальные оценки по внедрению третьей версии к 3000 году были недалеки от истины.

Если с Python дела обстоят более-менее серьезно, то инициативу Perl 6 уже можно рассматривать как анекдот. Начав разработки в 2000, с целью заменить Perl 5 (1994), Perl 6, похоже, все-таки зарелизили в 2015.

Следующего большого языка программирования не предвидится

Похоже, что дело обстоит именно так. Новых естественных островов, на которых мог бы закрепиться свежий лидер, похоже не

планируется, а эксперименты по созданию новых языков не могут побороть инертность индустрии разработки. Пожалуй, это не так уж и плохо.

Отсутствие новостей — хорошие новости. Если вы планируете начать изучать JavaScript, Java, C#, Python или PHP, это стоит сделать. Мейнстримные языки с нами всерьез и надолго, и их знание всегда пригодится. При этом, новые языки вроде Rust или Go тоже имеют перспективу, возможно, не как замена всего и вся, но как инструмент для решения интересных задач.

Дмитрий Мамонов

Департамент разработки,
Подразделение мержа в мастер,
Отдел работы с гит,
Оператор баш консоли с “пробегом”

Похожие публикации

Как построить грамотную систему тестирования? Инсайты от QA-экспертов: видео и презентации с митапа в Wrike

0

Wriketeam • 7 июня 2016 в 13:29

Опрос Wrike: Чем недовольны сотрудники? В основном, работой друг с другом

3

Anne_Usova • 17 декабря 2015 в 15:52

7 причин, по которым маркетологи Wrike любят Wrike

4

Anne_Usova • 10 декабря 2015 в 11:59

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

14

alexkuku • вчера в 16:12

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

4

lahmatiy • вчера в 13:05

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе 25

ПЕРЕВОД

Smileek • вчера в 10:32

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации 2

ПЕРЕВОД

ru_vds • вчера в 12:04

Как адаптировать игру на Unity под iPhone X к апрелю 0

P1CACHU • вчера в 16:13

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни 33

HostingManager • вчера в 13:49

Обзор рынка моноколес 2018 70

lozga • вчера в 06:58

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ 40

alizar • вчера в 15:14

Стивен Хокинг и его работа — что дал ученый человечеству?

8

marks • вчера в 14:46

Sunlike — светодиодный свет нового поколения

17

AlexeyNadezhin • вчера в 20:32