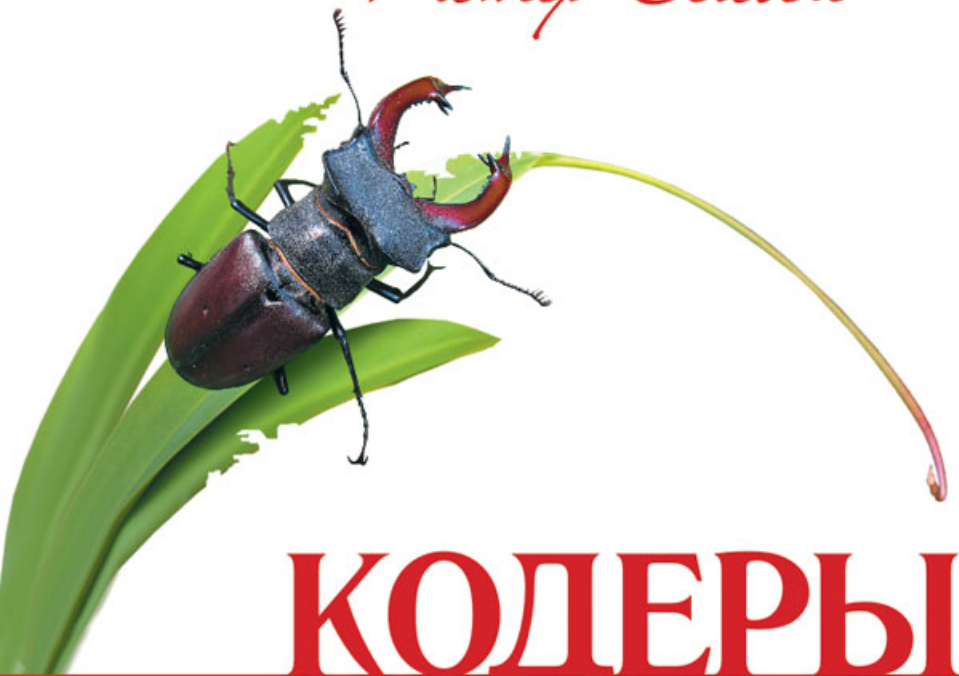
ПРОФЕССИОНАЛЬНО

Питер Сейбел



КОДЕРЫ ЗА РАБОТОЙ

РАЗМЫШЛЕНИЯ О РЕМЕСЛЕ ПРОГРАММИСТА

СИМВОЛ®

Coders at Work

Reflections on the Craft of Programming

Peter Seibel

Apress®

ПРОФЕССИОНАЛЬНО

Кодеры за работой

Размышления о ремесле программиста

Питер Сейбел

СИМВОЛ®

Санкт-Петербург — Москва
2011

Серия «Профессионально»

Питер Сейбел

Кодеры за работой
Размышления о ремесле программиста

Перевод А. Коробейникова и В. Петрова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Научные редакторы	*И. Сагалаев, С. Тепляков*
Редактор	*Т. Темкина*
Корректор	*О. Макарова*
Верстка	*К. Чубаров*

Сейбел П.

Кодеры за работой. Размышления о ремесле программиста. – Пер. с англ. – СПб: Символ-Плюс, 2011. – 544 с., ил.

ISBN 978-5-93286-188-2

Программисты – люди не очень публичные, многие работают поодиночке или в небольших группах. Причем самая важная и интересная часть их работы никому не видна, потому что происходит у них в голове. Питер Сейбел, писатель-программист, снимает покров таинственности с этой профессии. Он взял интервью у 15 величайших профессионалов: Кена Томпсона, создателя UNIX, Берни Козелла, участника первой реализации сети ARPANET, Дональда Кнута, Гая Стила, Саймона Пейтон-Джонса, Питера Норвига, Джошуа Блоха, Брэда Фицпатрика, создателя Живого Журнала, и других. Все они «подсели» на программирование еще в школе. Тогда, на заре зарождения отрасли, лишь в немногих учебных заведениях читались курсы по компьютерным наукам. Поэтому будущим гуру приходилось покорять профессиональные вершины самостоятельно, но всех их отличает творческое горение и полная самоотдача любимому делу.

Вы узнаете, что они думают о будущем программирования и как сами научились программировать, как, по их мнению, нужно проектировать ПО, как выбор языка программирования влияет на продуктивность и можно ли облегчить выявление труднонаходимых ошибок.

ISBN 978-5-93286-188-2
ISBN 978-1-4302-1948-4 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 15.02.2011. Формат 70×90 $^{1}/_{16}$. Печать офсетная.
Объем 34 печ. л. Тираж 1500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Амелии

Оглавление

Об авторе

Питер Сейбел – писатель-программист или программист-писатель. Получив высшее филологическое образование и какое-то время проработав журналистом, пленился Сетью. В начале 1990-х программировал на Perl для журнала «Mother Jones» и портала Organic Online. Участвовал в революции Java как сотрудник WebLogic, позже преподавал программирование на Java на заочных курсах при Калифорнийском университете Беркли. В 2003 году оставил работу архитектора транзакционной системы сообщений, основанной на Java, планируя за год освоить язык Лисп. Вместо этого два года писал книгу «Practical Common Lisp» (Common Lisp на практике), получившую премию Jolt Productivity Award. С тех пор работает «главной обезьяной» Gigamonkeys Consulting: учится учить, заниматься тай-цзи и быть отцом. Проживает в Беркли (Калифорния) с женой Лили, дочерью Амелией и собакой Мелани.

Благодарности

Прежде всего я хотел бы поблагодарить всех своих собеседников, которые любезно согласились потратить свое время и без которых эта книга была бы просто брошюркой с вопросами без ответов. Кроме того, я благодарен Джо Армстронгу и Берни Козеллу и их семьям за то, что предложили мне остановиться у них в Стокгольме и Виргинии. Еще одна особая благодарность – Питеру Норвигу и Джейми Завински, которые не только наговорили собственные интервью на мой диктофон, но и помогли мне связаться с другими будущими героями этих интервью.

Пока я путешествовал по всему свету, проводя интервью, меня приютили еще несколько семей: благодарю за гостеприимство Дэна Уэйнреба и Черил Моро (Бостон), Гарета и Эмму Маккоуэнов (Кембридж, Англия), а также моих родителей, которые предоставили мне отличный плацдарм для действий в Нью-Йорке. Кристофер Родс помог мне заполнить паузу между интервью экскурсией по Кембриджскому университету; благодаря ему и Дэйву Фоксу тот вечер закончился обедом и походом по кембриджским пабам.

Дэн Уэйнреб не только приютил меня в Бостоне, но и был самым усердным рецензентом на протяжении всей работы над этой книгой с той самой поры, когда я только выбирал потенциальных собеседников. Зак Бин, Люк Горри, Дэйв Уолден и моя мама тоже читали рукопись и своевременно меня подбадривали. Зак к тому же – что для моих книг уже традиция – является автором некоторых слов на обложке, на этот раз подзаголовка книги. Алан Кэй внес замечательное предложение включить Дэна Ингаллса и Питера Дойча. Скотт Фальман рассказал мне много полезного о начале карьеры Джейми Завински, а Дэйв Уолден прислал материалы по истории корпорации BBN Technologies, чтобы я смог подготовиться к интервью с Берни Козеллом. Если же я кого-то забыл, примите мою благодарность и вместе с нею извинения.

Спасибо издательству Apress, особенно Гари Корнеллу, который и предложил написать эту книгу, Джону Вакка и Майклу Бэнксу за их предложения, а также моему редактору Кэндейс Инглиш, которая исправила бесчисленные ошибки.

Наконец, самые искренние слова благодарности моей семье – и большой, и малой. Обе мои мамы – родная и теща – приходили присмотреть за ребенком, чтобы я мог еще немного поработать; мои родители на неделю приютили моих жену и малышку, позволив мне сделать последний мощный рывок. И главная благодарность – жене и дочери. Лили и Амелия, хотя иногда мне требуется некоторое время на работу, без вас, девочки, она не имела бы смысла. Я люблю вас.

Введение

Если не считать работу, проделанную Адой Лавлейс – аристократкой XIX века, которая придумала программы для так и не законченной «Аналитической машины» Чарльза Бэббиджа, – компьютерное программирование как область человеческой деятельности появилось совсем недавно: Конрад Цузе представил свой электромеханический компьютер Z3, первую работающую вычислительную машину общего назначения, в 1941 году, всего 68 лет назад. И всего 64 года прошло с тех пор, как шесть женщин – Кей Антонелли, Джин Бартик, Бетти Холбертон, Мартин Мельцер, Фрэнсис Спенс и Рут Тейтельбаум, – служивших в американских «вычислительных войсках» и составлявших вручную баллистические таблицы, были привлечены к созданию программ для ЭНИАК – первого электронного цифрового компьютера общего назначения. Среди ныне живущих многие – старшие представители поколения «бэби-бума»[1] и все родители «бэби-бумеров» – увидели свет, когда в мире не было ни одного программиста.

Теперь, разумеется, все иначе. Программисты заполонили все вокруг. Согласно данным американского Бюро трудовой занятости, в 2008 году в США примерно один из каждых 106 работников – всего более 1,25 млн человек – был разработчиком программного обеспечения или инженером-программистом. Это не считая профессиональных программистов за пределами США, студентов и программистов-любителей, а также тех, кто официально занимается чем-то другим, но тратит сколько-то времени – порой даже много времени – на то, чтобы подчинить компьютер своей воле.

И хотя написанием программ занимались и занимаются миллионы человек, хотя миллиарды, даже триллионы строк кода уже написаны, кажется, будто само понятие «программист» непрерывно уточняется с течением времени. Все еще идут споры о том, к какой области относится программирование – к математике или к инженерной деятельности. Ремесло, искусство или наука? Конечно же, идут споры, зачастую оже-

---

[1] Поколение «бэби-бума» (Baby Boom Generation) – жители Соединенных Штатов, рожденные в 1945–1964 гг. Окончание Второй мировой войны ознаменовалось ростом рождаемости, в результате чего на свет появилось почти 80 млн человек. – *Прим. науч. ред.*

сточенные, по поводу лучших способов программирования: Интернет полон сообщений в блогах и форумах, посвященных тому или иному способу написания кода. Книжные магазины набиты книгами о новых языках программирования, новых методах, новых попытках осмыслить задачи программирования.

Эта книга предлагает читателю нестандартный подход к понятию программирования. Она следует традиции, заложенной журналом «Paris Review», когда-то пославшим двух преподавателей проинтервьюировать романиста Э. М. Форстера; впоследствии все подобные интервью были объединены в сборник «Writers at Work» (Писатели за работой).

Я беседовал с пятнадцатью в полной мере состоявшимися программистами, имеющими большой опыт работы: с такими гениями кода, как Кен Томпсон, создатель операционной системы UNIX, и Берни Козелл, участник первой реализации сети ARPANET; с программистами, сочетающими выдающиеся академические заслуги с невероятными практическими навыками, – Дональдом Кнутом, Гаем Стилом и Саймоном Пейтон-Джонсом; промышленными исследователями – Фрэн Аллен из IBM, Джо Армстронгом из Ericsson, Питером Норвигом из Google; с выпускниками научно-исследовательского центра Xerox PARC Дэном Ингаллсом и Л. Питером Дойчем; с разработчиками ранних версий Netscape Джейми Завински и Бренданом Айком; с участниками проектирования и реализации языков программирования, применяемых сегодня для построения веб-приложений, – тем же Айком, Дугласом Крокфордом и Джошуа Блохом; и наконец с Брэдом Фицпатриком, создателем Живого Журнала (Live Journal) и одним из способнейших программистов эпохи Интернета.

Я расспрашивал каждого из них о программировании: как они научились этому, что открыли, создавая программы, и что думают о будущем программирования. В частности, я старался направить разговор на темы, вечно актуальные для программистов: как нужно проектировать программное обеспечение? как выбор языка программирования влияет на продуктивность и позволяет избегать ошибок? можно ли облегчить выявление труднонаходимых ошибок?

Поскольку однозначные решения всех этих проблем пока не найдены, неудивительно, что ответы оказались довольно разными. Джейми Завински и Дэн Ингаллс подчеркивали важность запуска кода сразу после его написания, а Джошуа Блох говорил о проектировании API и тестов, помогающих кодировать до этапа реализации. Что касается Дональда Кнута, то он поведал, что сделал эскиз полной версии системы компьютерной верстки TeX еще до того, как написал первую строку кода. А вот разные мнения относительно языка Си. Фрэн Аллен считает, что язык Си – причина снижения интереса к компьютерным наукам в последние

два десятилетия, а Берни Козелл называет Си «самой большой угрозой для безопасности современных компьютеров». В то же время Кен Томпсон утверждает, что угрозы для безопасности создают сами программисты, а не языки, а Дональд Кнут считает указатели в Си «одним из самых потрясающих усовершенствований в нотации», виденных им. Некоторые из моих собеседников высмеивали утверждение, будто формальные доказательства корректности помогают улучшить качество программ, но Гай Стил прекрасно проиллюстрировал их силу и в то же время ограниченность возможностей.

Кое с чем, однако, согласны все. Так, почти каждый настаивает на важности написания хорошо читаемого кода; большинство считают самыми трудноуловимыми ошибки в коде с параллельными вычислениями; никто не думает, что все проблемы программирования решены, – многие все еще ищут новые пути к разработке программного обеспечения с помощью автоматического анализа кода, улучшения организации совместной работы программистов или путем использования (или создания) более эффективных языков программирования. Однако практически все согласны, что вездесущие многоядерные процессоры серьезно повлияют на процесс разработки программ.

Эти разговоры отражают состояние данной области на определенном этапе, и, конечно, отдельные вопросы, затронутые в книге, с течением времени станут представлять лишь исторический интерес. Но даже в такой молодой области, как программирование, история сможет многому нас научить. Кроме того, думаю, все поднятые в книге темы раскрывают глубинный смысл того, что же такое разработка программного обеспечения, как мы можем лучше создавать его и что будет полезно программисту сегодня и через несколько поколений.

В заключение о названии книги: мы назвали ее «Кодеры за работой» по аналогии с уже упомянутой серией «Писатели за работой» журнала «Paris Review» и книгой «Founders at Work» (Учредители за работой) издательства Apress, которая применяет к основанию технологической компании примерно тот же подход, какой эта книга пытается применить к компьютерному программированию.

Я пониманию, что «кодирование» – лишь часть более широкого понятия «программирование». И всегда верил, что нельзя быть хорошим кодером, не являясь при этом хорошим программистом, или быть хорошим программистом, не являясь хорошим проектировщиком, общительным и думающим человеком. Разумеется, на страницах книги поднимаются все эти, а также многие другие вопросы. Не сомневаюсь, что обсуждения, которые вы собираетесь прочитать, прекрасно это отражают. Приятного чтения!

1

Джейми Завински

Лисп-хакер[1], один из первых разработчиков Netscape и владелец ночного клуба, Джейми Завински (Jamie Zawinski, JWZ) принадлежит к той избранной группе хакеров, которых узнают как по настоящим именам, так и по трехбуквенным инициалам.

Завински начал программировать еще подростком – поступил в лабораторию искусственного интеллекта при университете Карнеги–Меллона в качестве программиста на языке Лисп. Пробыв в колледже ровно столько, чтобы его возненавидеть, он около десяти лет проработал в мире Лиспа и искусственного интеллекта, удивительным образом погрузившись в умирающую хакерскую субкультуру, пока другие программисты, его сверстники, вырастали на микрокомпьютерах.

Он работал в Калифорнийском университете в Беркли с Питером Норвигом, который описал его как «одного из лучших программистов, которых он когда-либо нанимал», а затем в Лисп-компании Lucid, где участвовал в разработке редактора Lucid Emacs, позже переименованного в XEmacs, которая в итоге пришла к большому расколу проекта Emacs – одному из самых известных в истории свободного ПО.

---

[1] Хакер – ИТ-специалист высшей квалификации, в совершенстве изучивший работу компьютерных систем и ПО. Хакерами часто называют взломщиков программ, что в общем случае неверно. – *Прим. науч. ред.*

В 1994 году он наконец покинул компанию Lucid и мир Лиспа и перешел в Netscape, где стал одним из первых разработчиков UNIX-версии броузера Netscape Navigator, а позже – почтового клиента Netscape.

В 1998 году Завински, как и Брендан Айк, стал одним из основателей mozilla.org – организации, которая занималась переводом броузера Netscape в проект с открытым исходным кодом. Однако год спустя, разочарованный в отсутствии прогресса на пути к выпуску продукта, он покинул проект и приобрел ночной клуб DNA Lounge в Сан-Франциско, которым владеет до сих пор. В настоящее время Завински отдает все свои силы борьбе с Калифорнийским управлением по контролю за потреблением алкоголя, рассчитывая превратить клуб в место встречи любителей живой музыки всех возрастов.

В этом интервью мы среди прочего говорим о том, почему C++ – отвратительный язык программирования, о радости от того, что миллионы людей используют созданные тобой программы, а также о том, как важно «пинать» подающих надежды программистов.

Сейбел: Как вы научились программировать?

Завински: О, это было так давно, что я едва помню. Впервые я использовал компьютер для программирования, наверное, в восьмом классе. У нас было несколько компьютеров TRS-80, и мы дурачились с Бейсиком. По-моему, это вообще были внеклассные занятия. Помнится, невозможно было даже сохранять программы, так что мы просто вводили те, что печатались в журналах и так далее. Затем я прочел стопку книг. Помню, я читал книги по языкам, которые не мог использовать на наших компьютерах, и писал на бумаге программы на языках, о которых только прочитал.

Сейбел: Что это были за языки?

Завински: Один из них – APL. Я прочел статью про него и подумал, что это классный язык.

Сейбел: Ну да, не нужно иметь навороченную клавиатуру.[1] А в старших классах были компьютерные уроки?

---

[1] Язык программирования APL отличается очень короткой нотацией (большинство операций обозначаются одним-двумя символами), что делает программы крайне непонятными для чтения, но достаточно простыми для записи. – *Прим. науч. ред.*

Завински: В старших классах мы изучали Фортран. И все.

Сейбел: Как же вы пришли к Лиспу?

Завински: Я читал много фантастики. Мне казалось, что искусственный интеллект – это действительно классно, что компьютеры скоро покорят мир. И я решил почитать что-нибудь еще на эту тему. У меня в старших классах был приятель, Дэн Зигмонд, мы обменивались книгами и поэтому оба выучили Лисп. Однажды он пошел на встречу группы пользователей компьютеров Apple в Университете Карнеги–Меллон – там можно было просто обменяться программами, и Дэн надеялся добыть кое-что бесплатно. Он разговорился с каким-то студентом колледжа, и тот сказал: «Ого, пятнадцатилетний парень, который знает Лисп, – это фантастика. Спроси Скотта Фальмана насчет работы». Дэн так и поступил. Фальман взял его на работу. Тогда Дэн сказал: «Возьмите и моего друга», – то есть меня. Так что в итоге мы оба оказались у Фальмана. Думаю, он размышлял примерно так: «Раз уж эти школьники интересуются такими штуками, не будет большого вреда, если они поболтаются в лаборатории». Нам поручили самую простую грязную работу – надо было что-то перекомпилировать, когда выходила новая версия компилятора, и попробуй пойми, как это делается. Просто жуть. Представьте: двое ребят-школьников, а вокруг аспиранты занимаются языками программирования и искусственным интеллектом.

Сейбел: Значит, запустить программу на Лиспе вам впервые удалось в Карнеги–Меллоне?

Завински: Пожалуй, да. Какое-то время мы дурачились с языком XLISP, который использовался на Маках. Хотя, думаю, это было позже. Я там по-настоящему научился программировать (на рабочих станциях PERQ, установленных для проекта Spice) на языке Spice Lisp, который затем стал языком CMU Common Lisp. Странное это было место. Мы узнавали, что такое разработка программного обеспечения, на ежедневных собраниях – просто слушали и все. Но встречались и действительно забавные персонажи, например парень, который был кем-то вроде нашего менеджера – присматривал за нами. Звали его Скеф Хоули – этакий белобрысый детина довольно дикого вида. Выглядел просто устрашающе. Говорил он мало. Сколько раз, бывало, сижу я в своем отсеке – работаю, что-то делаю, пишу программу на Лиспе. И тут, шаркая, подходит босой Скеф с кружкой пива и становится позади меня. Я ему: «Привет», – а он в ответ буркнет что-то или вообще промолчит. Просто стоит и смотрит, как я жму на клавиши. Я что-то делаю, и тут он вдруг: «Пффф, ошибка», – и уходит. А я остаюсь в полном недоумении. Такой дзэнский подход: учитель стукнул меня палкой – значит, надо помедитировать.

Сейбел: Я связался по электронной почте с Фальманом, и он написал, что вы были талантливы и быстро всему учились. Но он также припомнил вашу недисциплинированность. «Мы потихоньку пытались научить его работать в команде, научить писать такой код, чтобы вы или кто угодно еще мог его понять и через месяц». Вы помните эти уроки?

Завински: Не помню, чтобы я учился чему-то такому. Конечно, очень важно писать код, к которому потом сможешь вернуться. Но мне сейчас 39, а тогда было 15, и кое-что уже забылось.

Сейбел: В каком году это началось?

Завински: В 1984 или 1985. Думаю, летом, когда я из 10 класса перешел в 11. Занятия в школе заканчивались часа в четыре или около того, я отправлялся туда и оставался там до восьми-девяти вечера, хотя и не каждый день. Так или иначе, я проводил там немало времени.

Сейбел: И после школы вы ненадолго пришли в Университет Карнеги–Меллона?

Завински: Да. Дело в том, что я ненавидел учиться в старших классах. Это было самое жуткое время в моей жизни. И перед выпуском я спросил Фальмана, не возьмет ли он меня на полный день. Он ответил: «Нет, но у меня есть друзья, которые начинают новое дело. Поговори с ними». Это была фирма Expert Technologies (ETI). Кажется, Фальман был в деле. Они разрабатывали систему автоматической пагинации «желтых страниц» на Лисп, и я уже знал там кое-кого, кто работал с Фальманом. Они взяли меня, и все было хорошо, но через год я запаниковал: «Господи, я же устроился на эти две работы совершенно случайно, такое больше не повторится. Что будет, когда я уйду отсюда?» Без диплома колледжа мне светил разве что Макдональдс. Значит, надо было добывать диплом.

План был такой: я работаю на полставки в ETI, а остальное время учусь, тоже в половинном режиме. На деле же вышло, что работать и учиться пришлось по полной. Так продолжалось недель шесть, может, даже девять. Знаю только, что успел пропустить выбор курсов на семестр, так что деньги было уже не вернуть. Но не успел получить какие-нибудь оценки. В общем, учусь я или нет, непонятно.

Это было ужасно. В школе тебе говорят: мол, у нас тут один отстой и тесты, но в колледже все будет лучше. Поступаешь в колледж, и первый год там все то же самое. И тебе говорят: в магистратуре будет лучше. Все тот же отстой, только в другом месте – не для меня. Вставать в восемь, зубрить. Мне не разрешили пропустить курс «Введение в вычислительную технику», где объясняли, как пользоваться мышью. «Я полтора года работаю в этом университете, – говорил я, – и знаю, как пользоваться мышью». «Нет, мы не можем позволить вам, – отвечали мне. –

У нас такой порядок». И все в таком же духе. Я не выдержал и бросил колледж. И рад, что сделал это.

Потом я работал в ETI года четыре или около того, пока компания не стала разваливаться. Мы работали на Лисп-машинах[1] серии TI Explorer, и я – кроме того, что работал над экспертными системами, – тратил массу времени на возню с пользовательским интерфейсом и на понимание того, как они вообще работают, сверху донизу. Я любил их, любил копаться в операционной системе и понемногу осознавать, как это все устроено.

Я написал кучу кода и разместил в нескольких группах новостей объявление о том, что ищу работу, предлагая при этом взглянуть на фрагмент моего кода. Питер Норвиг увидел его и назначил мне собеседование. Моя тогдашняя подружка переехала сюда, в Калифорнию, чтобы учиться в Университете Беркли, и я переехал вместе с ней.

Сейбел: Норвиг тогда был в Беркли?

Завински: Да. Это была очень странная работа. Там был целый выводок практикантов, которые исследовали понимание людьми естественных языков: это были лингвисты по образованию, которые слегка занимались программированием. И нужен был тот, кто собрал бы написанные ими куски и обрывки кода и сляпал бы из них что-то работающее.

Это было невероятно трудно, потому что у меня не хватало подготовки, чтобы понять, что, черт возьми, они там делают. То и дело получалось так: я остолбенело смотрю на что-то и не знаю, что это значит и в каком направлении двигаться, что читать, чтобы понять это. И я спросил Питера. Он внимательно выслушал меня и сказал: «В общем ясно, что пока это для тебя непонятно. Во вторник сядем, и я тебе все объясню». Значит, делать мне было пока нечего. Я углубился в работу с оконными системами, скринсейверами и другими штуками, связанными с пользовательским интерфейсом, которыми раньше занимался для забавы.

После шести-семи месяцев я почувствовал, что трачу свое время впустую. Я не делал для них ничего серьезного, это было похоже на каникулы. Позже не раз, действительно много работая, я оглядывался и спрашивал себя: «Зачем ты оставил эту работу, похожую на каникулы? Что тебе не нравилось? Тебе платили за разработку скринсейверов!»

---

[1] Речь о компьютерах, аппаратно оптимизированных для выполнения приложений на Лиспе, в отличие от обычных компьютеров, оптимизированных для ассемблерного кода. Такие компьютеры широко применялись для исследования задачи искусственного интеллекта, поскольку компьютеры общего назначения с ними просто не справлялись. См. *http://en.wikipedia.org/wiki/Lisp_machine*. – *Прим. науч. ред.*

В конце концов я устроился в компанию Lucid – одну из двух оставшихся Лисп-компаний. По-настоящему меня заставило уволиться чувство, что в Беркли я ничего не достигну. Вокруг меня были сплошь лингвисты, кое с кем я до сих пор дружу, они хорошие ребята – но не программисты. Абстрактные понятия им намного интереснее решения реальных задач. Я хотел делать что-то такое, чтобы можно было ткнуть пальцем и сказать: «Смотри, какую классную штуку я сделал».

Сейбел: Именно работая в Lucid, вы начали заниматься графическим редактором XEmacs. А когда вы туда пришли, вы что-нибудь писали на Лиспе?

Завински: Да, в одном из первых проектов, над которым я работал. Я, правда, не помню, что это был за компьютер, но это точно был 16-процессорный компьютер с поддержкой параллельных вычислений, на котором мы использовали собственную реализацию языка Common Lisp[1] с управляющими структурами, позволяющими распараллеливать задачи на разные процессоры.

Я немного поработал над задачей уменьшения накладных расходов при создании потоков, чтобы, например, выгоды от применения параллельного вычисления чисел Фибоначчи не сводились на нет накладными расходами создания стека для каждого потока. Мне это действительно нравилось. Я впервые имел дело с таким замысловатым компьютером.

А до этого я поднимал Лисп на новых типах машин. Обычно это означало, что кто-то уже написал компилятор под новую архитектуру железа и скомпилировал загрузчик Лиспа. Затем я брал бинарный, вроде бы работающий код и расшифровывал формат загрузчика новой машины, чтобы затем написать небольшую программу на Си, которая бы загрузила бинарные файлы на страницу памяти, сделала ее исполняемой и передала ей управление. После чего, вполне возможно, вы получали командную строку Лиспа и могли вручную загружать другие программы.

Из-за отсутствия нормальной документации этот процесс для каждой новой архитектуры был непростым делом. Приходилось компилировать код на Си, а затем просматривать и редактировать его в Emacs байт за байтом. Давайте-ка посмотрим, что же произойдет, если вот этот бит установить в ноль... Рухнет или нет?

---

[1] Речь идет о коммерческой реализации языка Common Lisp компании Lucid, получившей название Lucid Common Lisp. Позднее права перепродавались от одной компании к другой, пока не перешли к компании LispWorks, которая и продает эту реализацию под маркой Lucid Common Lisp. – *Прим. науч. ред.*

Сейбел: Когда вы говорите, что не было нормальной документации, это значит, что документация была неточной или что ее не было вовсе?

Завински: Нет, документация была, но зачастую она не отвечала действительности. Возможно, ошибка вкралась несколькими версиями раньше – кто знает? Но в определенный момент ты изменяешь этот бит, и машина уже не воспринимает твою программу как исполняемый модуль, и тебе приходится выяснять, что же произошло.

Сейбел: Ну, такое случается сплошь и рядом, начиная от низкоуровневого системного программирования и заканчивая высокоуровневым API, когда всё начинает работать совсем не так, как ты ожидаешь, или не так, как написано в документации. Как вы справлялись с этим?

Завински: Да просто начинаешь ожидать этого. Чем раньше поймешь, что сбился с пути, тем раньше сможешь выяснить, где именно. Лично я пытался создать исполняемый файл. Я знал, что компилятор Си может создавать исполняемые файлы. Поэтому алгоритм работы был такой: берешь хороший исполняемый файл и начинаешь его ковырять, пока он не превратится в плохой. Это основной механизм обратной разработки (reverse engineering).

Думаю, именно в компании Lucid я исправил самый сложный компьютерный баг. Я дошел до момента выполнения исполняемого файла, когда тот пытался загрузить интерпретатор Лиспа, но после выполнения 500 инструкций процесс загрузки падал. Тогда я начал выполнять процесс загрузки пошагово, чтобы выяснить, где же он падает. Хотя это было бессмысленно, создавалось впечатление, что процесс падал каждый раз в другом месте. Я стал исследовать ассемблерный код компьютерной архитектуры, о которой имел лишь смутное представление. Наконец до меня дошло. «Господи, при пошаговом выполнении он делает что-то не то. Возможно проблема связана с временными задержками». В итоге я понял, что происходило: дело в том, что это была одна из первых машин с упреждающим исполнением команд[1]. В этом случае вы-

[1] Упреждающее исполнение команд (Speculative Execution), или исполнение команд по предположению, – это совокупность методов, позволяющая ЦП с конвейерной архитектурой обрабатывать команды без уверенности в том, что они реально будут исполняться в программе (например, в случае условного перехода). Если предположение оказывается верным, то исполнение команд продолжается и выигрывается время, а если нет (misspeculation), результаты упреждающего исполнения аннулируются. – *Прим. науч. ред.*

полнялись обе ветви кода[1]. Но GDB[2] при пошаговой отладке выполнял только одну из ветвей. Так что баг был в GDB.

Сейбел: Здорово.

Завински: Точно. Но это меня подкосило. «Господи! Мне придется отлаживать GDB, который я первый раз вижу». Чтобы обойти ошибку отладчика, нужно остановить выполнение процесса перед инструкцией ветвления, задать точки останова в обеих ветвях и продолжить выполнение. Именно таким способом мне удалось воспроизвести ситуацию и понять, что же происходит на самом деле. Затем я потратил около недели на исправление GDB, но так и не смог понять, в чем же дело. Я предполагал, что из-за проблем с одним из регистров отладчик считал, что всегда выполняется одна из ветвей условия или что-то в этом роде. Поэтому я изменил команду пошагового выполнения инструкций, чтобы определить, когда оно дойдет до инструкции ветвления, и там сказать: «Стоп, это не делай». Теперь я мог просто пошагово выполнять программу. Выполнение в конце концов останавливалось, я вручную задавал точку останова и продолжал выполнение. Когда что-то отлаживаешь, понимая, что не только путь выбран неверный, так еще и инструмент никуда не годится, – что может быть хуже.

Разработка систем на Лиспе была особенно запутанной, поскольку GDB был совершенно неприменим к Лисп-коду, который не содержал никакой отладочной информации. Причина была в том, что интерпретатор Лиспа был разработан с помощью компилятора, о котором GDB не имел ни малейшего понятия. Думаю, для некоторых платформ создавались стековые фреймы, которых отладчик GDB просто не понимал. На этом этапе GDB был способен лишь на пошаговый прогон ассемблерного кода. Поэтому мы хотели избавиться от него как можно скорее.

Сейбел: А потом у вас появился отладчик Лиспа, и вы оказались во всеоружии.

Завински: Ага.

Сейбел: И примерно в то же время компания Lucid сменила курс, решив создавать интегрированную среду разработки для C++.

---

[1] Теоретически, при наличии условия, должна выполняться только одна ветвь программы, но благодаря упреждающему исполнению команд выполнялись обе ветви, хотя результаты одной из них затем отбрасывались. – *Прим. науч. ред.*

[2] GDB (GNU Project Debugger) – переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет выполнять отладку многих языков программирования, включая Си, C++ и Фортран. – *Прим. науч. ред.*

Завински: Это началось еще до меня: когда я пришел туда, интегрированная среда разработки уже создавалась. Люди начали переходить с Лиспа на Energize – так называлась эта среда разработки. Это был отличный продукт, но он появился на два-три года раньше, чем нужно. Никто – по крайней мере, никто среди пользователей UNIX – не думал, что им вообще это нужно. Сейчас все используют такие возможности, но тогда мы тратили кучу времени, объясняя, почему эта штука лучше, чем vi[1] и GCC. Еще я делал кое-что для Emacs. Кажется, тогда я уже переписал компилятор байт-кода[2]. Зачем мне это было нужно? Правильно, потому что я делал что-то вроде адресной книги.

Сейбел: Базу данных для Большого Брата?

Завински: Ага. Но работала она ужасно медленно. Я стал выяснять, почему, и понял, что компилятор ни к черту не годится. Я переписал компилятор, что и привело к моей первой ссоре с непримиримым Стеллменом. Тогда я много чего узнал о Emacs.

Сейбел: А изменения в компиляторе коснулись формата байт-кода или только процесса компиляции?

Завински: Я сделал несколько изменений: внес исправления в интерпретатор байт-кода, написанного на Си, а также добавил несколько новых инструкций для повышения производительности. Но компилятор мог быть сконфигурирован, чтобы генерировать байт-код в старом формате или использовать преимущества нового.

Так вот, я написал новый компилятор, а Стеллмен заявил: «Не вижу необходимости в этих изменениях». А я ему в ответ: «Да что вы? Теперь генерируется более быстрый код». А он: «О'кей, тогда пришли мне все изменения исходников и объясни каждую измененную строку». Тогда я ответил: «Нет, я не буду этого делать. Я полностью переписал старый компилятор, потому что он был дерьмовым». Это ему не понравилось. Мой компилятор был добавлен только потому, что я выпустил его, тысячи людей начали им пользоваться, компилятор им понравился, и они надоедали Стеллмену два года. Вот он и добавил мой компилятор, чтобы его больше не доставали.

---

[1] vi (сокр. от visual) – серия текстовых редакторов операционных систем семейства UNIX, которые применялись совместно с компиляторами GCC для разработки ПО. – *Прим. науч. ред.*

[2] В зависимости от реализации некоторые компиляторы могут преобразовывать исходный текст на языке ЛИСП сразу же в машинный код (native code), а некоторые вначале компилируют исходный код в промежуточный (байт-код), который уже затем интерпретируется в машинный код конкретной машины. Для Emacs были реализованы компилятор и интерпретатор. – *Прим. науч. ред.*

Сейбел: Вы подписали передачу авторских прав на этот компилятор компании Free Software Foundation?

Завински: Да, я сделал это сразу же. По-моему, это было первое, о чем говорилось в том электронном письме. Там было что-то вроде: «Пришли мне изменения и подпиши это». Я подписал и сказал: «Остальное я выполнить не могу. Я не могу прислать вам изменения. Это просто смешно. Тут все описано, взгляните сами». Вряд ли он смотрел на это хоть раз.

Толкуют, будто были какие-то судебные разборки между Lucid и FSF. Это полная ерунда. Мы подписывали бумаги о передаче авторских прав на все, что делали для FSF. Им было зачем-то нужно утверждать, будто мы делали это не всегда. Бывало, мы подписывали одни и те же бумаги несколько раз, потому что они говорили, будто потеряли их. Кажется, подобная шумиха была с подписыванием бумаг по XEmacs, но это было позже, когда я уже ушел.

Сейбел: Итак, вы начали с Лиспа, но явно не зациклились на нем на всю жизнь. Что было потом?

Завински: Следующим языком, на котором я создавал что-то серьезное, был Си. Казалось, будто вернулись времена программирования на ассемблере под Apple II: это был ассемблер PDP-11, который считал себя настоящим языком. Программирование на Си, как вы наверное знаете, малоприятное занятие, поэтому я старался как можно дольше держаться подальше от всего этого. А C++ – просто гадость. С ним все не так. Так что я старался как можно дольше держаться от него подальше и в Netscape писал на Си. Это было просто – ведь мы были нацелены на небольшие компьютеры, на которых нельзя было нормально использовать программы на C++, потому что код распухал до безумия, как только были задействованы библиотеки. И потом компиляторы для C++ постоянно менялись, что приводило к массе проблем с несовместимостью. Поэтому мы с самого начала выбрали для себя ANSI C, и он хорошо служил нам. После всего этого Java отчасти воспринимался как возврат к Лиспу – в том смысле, что этот язык не лезет из кожи вон, желая отпугнуть вас. Им удобно пользоваться.

Сейбел: В каком смысле?

Завински: Автоматическое управление памятью. Функции выглядят именно как функции, а не как подпрограммы. Очень много сделано для обеспечения модульности. В коде на Си всегда есть искушение применить оператор `goto` только потому, что это просто.

Сейбел: Значит, сейчас вы в основном используете языки Си и Perl?

Завински: Ну, я вообще сейчас не много программирую. В основном пишу глупые маленькие скрипты на Perl для поддержания работы

моих серверов. Я написал кучу дурацких программ поиска картинок для моих MP3-файлов или нечто вроде того. Простенькие одноразовые программки.

Сейбел: Вам нравится Perl или он просто всегда под рукой?

Завински: Терпеть его не могу, ужасный язык. Но он установлен абсолютно везде. Садишься за компьютер – и не нужно никого просить установить Perl, чтобы выполнить свой скрипт: Perl там уже есть. Это единственный аргумент в его пользу.

У него неплохая коллекция библиотек. Часто есть библиотека, позволяющая делать именно то, что тебе нужно. Пусть библиотеки иногда неважно работают, но это уже что-то. Не то, что с Java, когда пишешь что-нибудь на этом языке и пытаешься понять, что вышло. Я сам с трудом установил Java на своем компьютере. Это ужасно. Мне кажется, Perl – противный язык. Если научиться использовать его хоть немного, можно сделать его похожим на Си или, скорее, на JavaScript. Сумасшедший синтаксис, непонятные структуры данных. Немного хорошего можно сказать о Perl.

Сейбел: Но он не так плох, как C++.

Завински: Нет, конечно, нет. Они созданы для разных задач. Есть задачи, которые намного проще реализовать на Perl (или подобном ему языке), чем на Си, только потому, что все так называемые скриптовые языки ориентированы на работу с текстом. Вот чего я действительно не понимаю, так это различия между «программированием» и «написанием скриптов». По-моему, чепуха все это. Но если основная твоя работа заключается в обработке текста или запуске программ (например, запустить wget[1], получить от нее какой-то HTML и сопоставить его с образцом), то гораздо легче это сделать на Perl, чем даже на Emacs Lisp.

Сейбел: Не говоря о том, что Emacs Lisp не слишком удобен для работы с утилитами командной строки.

Завински: Ну да, хотя я все время писал разные мелкие утилиты с помощью Emacs. Было время, на раннем этапе работы в Netscape, когда часть процесса сборки включала запуск скриптов с помощью команды `emacs -batch` для работы с некоторыми файлами. Это никому не нравилось.

Сейбел: Представляю... А как насчет XScreenSaver[2] – все еще работаете над ним?

---

[1] wget – программа для загрузки файлов по сети.

[2] XScreenSaver – коллекция из более чем двухсот различных заставок (screen saver) для UNIX и Mac OS. Создана Джейми Завински в 1992 году и до сих пор им поддерживается. – *Прим. науч. ред.*

Завински: Я до сих пор пишу новые скринсейверы – только ради развлечения и только на Си.

Сейбел: Применяете ли вы какую-нибудь интегрированную среду разработки?

Завински: В основном Emacs. Правда, недавно я портировал XScreenSaver на OS X. Я сделал это так: реализовал заново Xlib на базе Cocoa (графической основе Маков), поэтому мне не пришлось переписывать код всех скринсейверов. Они все еще обращаются к Xlib, но я реализовал все соответствующие методы. Сделано это было на Objective-C, который оказался отличным языком программирования, и работа доставила мне огромное удовольствие. Он определенно напоминает Java в лучших его проявлениях, но также напоминает и Си. То есть в основном это Си, и можно напрямую использовать Си-код, вызывая нужные функции без лишних усилий.

Сейбел: Работая в компании Lucid, что вы узнали по технической части кроме политической составляющей разработки Emacs?

Завински: Работая там, я точно стал более хорошим программистом. Во многом потому, что был окружен действительно очень умными людьми. Все, кто там работал, были великолепны. Просто здорово находиться в коллективе, в котором, если кто-то скажет: «Это чепуха» или «Нужно сделать это вот так», – можно просто верить на слово, не сомневаясь, что он знает, о чем говорит. Это было на самом деле здорово. Не скажу, что раньше я не бывал среди умных людей, но это был именно коллектив высококлассных специалистов в равной степени.

Сейбел: А насколько велика была команда разработчиков?

Завински: В компании было человек 70 – точно не знаю, и около 40 из них разработчики. В команде Energize было 20–25 человек. Все разработчики делились по направлениям. Кто-то работал над компиляторами, кто-то над серверной частью базы данных. Кто-то работал над пользовательским интерфейсом, не связанным с Emacs. Я и еще двоетрое занимались интегрированием Emacs с внешним окружением. Так получилось, что я работал в основном над Emacs, пытаясь сделать так, чтобы нашим редактором Emacs 19 можно было пользоваться, чтобы он не падал то и дело и чтобы под ним запускались все пакеты, которые должны запускаться.

Сейбел: То есть вы хотели, чтобы Emacs, который являлся составной частью вашего продукта, был полнофункциональной версией Emacs.

Завински: Изначально мы не собирались включать Emacs в наш продукт. Идея была такая: на вашей машине уже стоит Emacs, вы берете наш продукт, и они совместно работают. Например, на вашей машине установлен компилятор GCC и наш продукт, и они совместно работают.

Кажется, одним из первых кодовых названий нашего продукта было что-то вроде Hitchhiker (попутчик), так как идея была в том, что он будет брать и интегрировать все имеющиеся у вас инструменты – заставит их «общаться» между собой, предоставив им необходимый уровень коммуникации.

Но это не сработало. Мы стали выпускать собственные версии GCC и GDB, потому что не успевали за изменениями этих систем, по крайней мере, успевали не всегда. То же самое было и с Emacs. Поэтому мы выпустили все целиком. В конце концов мы пошли таким путем: «Так, мы заменили Emacs. Черт. Нам пришлось это сделать, поэтому нужно заставить его работать нормально». Только на режим эмуляции vi я потратил уйму времени.

Сейбел: И это несколько недель вашей жизни, которые вам никогда не хотелось бы пережить снова.

Завински: И не говорите. Это было настоящее испытание. Кажется, в результате все заработало. Настоящая проблема была не в том, что режим эмуляции vi работал плохо, а в том, что пользователи были вынуждены постоянно выходить и перезапускать vi. И что бы я ни делал, эту проблему никак не удавалось решить. Пользователь говорил: «Я думал, она будет запускаться за полсекунды, а она запускается за четырнадцать. Это просто смешно. Я не могу этим пользоваться».

Сейбел: Почему вы ушли из компании Lucid?

Завински: Lucid разваливалась. Людей то и дело отправляли в неоплачиваемый отпуск. Я разослал письма своим знакомым: «Привет, кажется, мне скоро будет нужна новая работа». Одним из них был Марк Андрессен. Он сказал: «Забавно, что ты об этом упомянул, ведь на прошлой неделе мы как раз создали компанию». Так я нашел работу.

Сейбел: Итак, вы ушли в Netscape. Над чем вы работали?

Завински: Я практически сразу начал работать над версией броузера для UNIX. К тому моменту было написано совсем немного кода – результат нескольких дней работы. Немногим больше было сделано для версий под Windows и Мак. Модель системы состояла из крупного куска бизнес-логики, независимой от платформы, и небольшой части кода, отвечающего за представление для каждой платформы.

Сейбел: Это был полностью новый код?

Завински: Полностью новый. Большинство основателей компании Netscape были разработчиками броузера NCSA/Mosaic. Они разработали разные версии, которые по сути представляли собой три разные программы. И все шестеро, делавших это, оказались в Netscape. Они не

использовали повторно старый код, но уже решали подобную задачу ранее.

Сейбел: То есть они просто начали писать код с чистого листа?

Завински: Именно так. Я никогда не видел код броузера Mosaic (кстати, я до сих пор его не видел). Как раз в это время у нас были судебные разбирательства: университет утверждал, что мы использовали их код, но мы, кажется, как-то уладили этот вопрос. Ходили слухи, будто мы действительно использовали их код, но мы этого не делали.

И зачем? Каждый хочет попробовать еще раз заново, верно? Во время разработки программы находишь ответы на многие вопросы, и, получив шанс выкинуть все и начать с нуля, конечно же, воспользуешься этим шансом. Во второй раз все должно получаться гораздо лучше. И действительно получается. Например, общепринятая архитектура не предоставляла возможность параллельной загрузки изображений. А ведь это действительно важная возможность. Поэтому мы разработали лучшую архитектуру.

Сейбел: Но это похоже на классический случай синдрома второй системы.

Завински: Вот именно.

Сейбел: Так как же всем вам удалось его избежать?

Завински: Самым святым для нас были сроки. Или мы выпустим новый продукт за полгода, или умрем, пытаясь это сделать.

Сейбел: Как вам удалось справиться со сроками?

Завински: Мы осмотрелись и поняли, что если не сделаем это за полгода, кто-нибудь опередит нас. Поэтому мы решили, что справимся с этим за шесть месяцев.

Сейбел: Учитывая, что вначале была определена дата, вам нужно было жертвовать либо возможностями системы, либо качеством. Как вы поступили?

Завински: Мы долго обсуждали возможности системы. Ну, на самом деле не так и долго, но нам казалось именно так, потому что тогда каждый день был как неделя. В конце концов мы определили все возможности. У нас была белая доска, на которой мы писали наши идеи, связывая их друг с другом. Всего нас было человек шесть-семь, точно не помню. Группа умных, самовлюбленных людей, которые сидят в одной комнате и орут друг на друга неделю или около того.

Сейбел: Шесть-семь человек – это вся команда разработчиков Netscape или только версии для UNIX?

Завински: Это была вся команда разработки клиентской части. Были еще ребята, отвечавшие за серверную часть, которые в основном занимались разработкой собственной версии Apache. Мы мало общались с ними, так как были заняты. Мы завтракали вместе, но не более того. Мы выяснили, что должно быть в системе, и распределили работу так, чтобы над каждой частью проекта работало не более двух человек. Я занимался UNIX-частью, а Лу Монтулли делал основную работу по сетевой составляющей серверной части. Предварительная версия команды была следующей: Эрик Бина занимался общей компоновкой системы, Джон Миттельхаузер и Крис Хаук занимались пользовательским интерфейсом под Windows, Алекс Тотич и Марк Ланетт – пользовательским интерфейсом под Мак. Потом эти команды слегка увеличились. Мы совещались, потом расходились по своим отсекам и стучали по клавиатуре 16 часов в сутки, пытаясь сделать так, чтобы хоть что-то работало.

Обстановка была отличная, мне правда нравилось. Поскольку каждый считал себя правым, мы постоянно спорили, но быстро понимали друг друга. Кто-нибудь заглядывает в твой отсек и говорит: «Какого хрена ты сохранил этот код? Так нельзя делать, это же дерьмо собачье. Ты идиот». Отвечаешь ему: «Да пошел ты!», смотришь на свой код, исправляешь ошибки и сохраняешь его. Мы были весьма резкими парнями, но зато не тратили лишнее время, потому что никто ни с кем не церемонился, не объяснял подолгу, почему считает что-то неправильным. Можно было просто сказать: «Да это куча дерьма, я не буду это использовать». И вопрос тут же решался. Да, атмосфера была напряженная, но мы сделали все довольно быстро.

Сейбел: То есть для быстрого создания программы требовалась многочасовая интенсивная работа?

Завински: Да уж, не курорт. Я знаю, что мы поступали именно так, и это работало. Ответить на этот вопрос можно так: а есть ли примеры того, как кто-то быстро и качественно создал здоровенную систему, обедая дома и регулярно высыпаясь? Бывало ли такое? Не знаю. Может, и бывало.

Но не всегда нужно делать все как можно быстрее. Нужно не перегореть за два года, а быть в состоянии проработать еще десять. А при 80-часовой рабочей неделе это невозможно.

Сейбел: Чем вы больше всего гордитесь из того, над чем работали?

Завински: На самом деле я горжусь самим фактом того, что мы выпустили это. Всю систему. Я был полностью сконцентрирован на своей части – создании пользовательского интерфейса для UNIX. Но горжусь

я именно тем, что мы вообще выпустили систему и людям она понравилась. Пользователи сразу же стали переходить с NCSA Mosaic, говоря: «Ух ты, это лучшая система, которую мы когда-либо видели». У нас на панели инструментов была кнопка для страницы What's Cool (Что есть классного), чтобы показать миру все эти безумные веб-сайты, созданные к тому времени, – штук двести, наверное! Не сказать, чтобы я сильно гордился кодом, скорее именно тем, что дело сделано. Код во многих отношениях был не очень хорош – слишком я торопился. Но все было закончено. Мы выпустили систему – это было главное.

В ту ночь, выпустив бета-версию .96, мы все сидели по разным углам комнаты и смотрели, как идет закачка с прикрученным звуковым сигналом. Это было потрясающе. А через месяц программой, которую написал я, пользовались два миллиона человек. Это было невероятно. Это определенно стоило потраченных усилий – повлиять на жизнь людей, сделать так, чтобы их времяпрепровождение стало веселее, или приятнее, или удобнее благодаря тому, что мы сделали.

Сейбел: После этой бешеной гонки в какой-то момент надо было вернуться к качеству полученного кода. Как вы, ребята, справилась с этим?

Завински: Честно говоря, не очень. Никогда не было времени начать все сначала и переписать код. Да и вообще не слишком это здорово – начинать все заново и переписывать код.

Сейбел: На каком-то этапе вы также работали и над почтовым клиентом, да?

Завински: Когда мы работали над версией 2.0, Марк зашел в мой отсек и сказал: «Нам нужен почтовый клиент». Я ответил: «Отлично, я уже работал над почтовыми клиентами». Я тогда жил в Беркли и несколько недель почти не появлялся в офисе – сидел часами в кафе и набрасывал что-то, пытаясь выяснить, чего же хочу от почтовой программы. Я делал списки, что-то из них вычеркивал, думая, когда же приду к чему-нибудь. Как должен выглядеть пользовательский интерфейс?

И вот я вернулся на работу и стал писать код. Потом Марк снова зашел и сказал: «Мы тут взяли еще одного парня, он занимался почтовыми клиентами. Будете работать вместе». Это был Терри Вейссман, просто фантастический человек – мы прекрасно сработались. И совсем в другом ключе, по сравнению с работой над броузером на ранней стадии.

Мы совсем не кричали друг на друга. Просто не представляю, как можно было так работать, да и вообще, работал ли так хоть кто-нибудь еще. Мы распределили свою работу следующим образом. Я делал набросок дизайна и начинал понемногу писать код, и раз в несколько дней мы

смотрели на список задач и говорили друг другу: «Я буду работать над этим. – А я над этим». И расходились.

Мы сохраняли код в репозитории и встречались снова. Он говорил: «Ладно, я с этим разобрался, а ты что делаешь?!» – «Работаю вот над этим». – «Хорошо, тогда я займусь вон тем». Так мы в некотором роде делили задачи между собой, и все шло превосходно.

Бывали и разногласия. Я предложил добавить фильтрацию на уровне папок, поскольку у нас не было времени сделать это как следует. Он сказал: «Нет-нет, я правда думаю, что мы должны сделать все как надо». А я: «Да у нас же нет времени!» Но он все сделал той же ночью.

Еще кое-что: мы с Терри виделись редко – он жил в Санта-Крус, я в Беркли. До работы нам было ехать примерно одинаково, но с разных сторон. А поскольку по работе нам нужно было общаться только между собой, то мы договаривались так: «Если ты согласен, чтобы я не приезжал, то и я согласен, чтобы ты не приезжал». – «Давай».

Сейбел: Вы вдвоем много переписывались по электронной почте?

Завински: Да, постоянно. Это было еще до эры мгновенных сообщений (Instant messenger, IM) – сейчас, конечно, мы бы прибегли именно к ним, потому что постоянно слали друг другу электронные письма. И переговаривались по телефону.

Итак, мы выпустили версию 2.0 с почтовым клиентом, хорошо принятую пользователями. Потом мы работали над версией 2.1, и я уже начал думать, что работа подходит к концу, но это оказался один из случаев, когда мы так и не смогли выпустить версию с первого раза. Мы с Терри были уже на полпути, когда зашел Марк и сказал: «Мы покупаем такую-то компанию. И там разрабатывают почтовый клиент, похожий на ваш». Я ответил: «Да, хорошо, но у нас уже есть такая программа». Он пояснил: «Понятно, но мы растем очень быстро, и нам все труднее нанимать на работу хороших людей. Иногда можно их нанять, купив другую, хорошо знакомую компанию». – «Ладно. Чем они будут заниматься?» – «Работать над вашим проектом». – «Черт, тогда я займусь чем-нибудь другим».

И как только Netscape купила эту компанию (которая называлась Collabra), нас с Терри отдали под их руководство. Эта компания выпускала почтовый клиент, во многом очень похожий на наш, за исключением того, что он работал только под Windows и потерпел полный крах на рынке.

А потом им крупно повезло – их купила компания Netscape. После чего Netscape передала бразды правления этой компании. Но вместо того чтобы отвечать только за почтовую программу, они оказались во

главе всего подразделения по разработке клиентской части. Мы с Терри работали над Netscape 2.1, когда это произошло и начался процесс переписывания. Тогда стало понятно, что Netscape 3.0 будет выпущен слишком поздно, и наша версия 2.1 превратилась в версию 3.0, потому что нужно было выпустить на рынок основную версию.

А потом версия 3.0, над которой они начали работу, стала версией 4.0, которая, как вы знаете, стала одной из самых больших неудач в области программных разработок. Главным образом это и погубило компанию. Она умирала долго, но это было неотвратимо. Процесс переписывания был инициирован новоприобретенной компанией, которая ничего особенного не добилась, пренебрегла всей нашей работой и нашими достижениями, а сразу же начала страдать синдромом второй системы и потянула нас ко дну.

Они полагали, что, оказавшись у руля, просто обязаны поступать по-своему. Но когда они делали это по-своему в своей компании, они провалились. И когда люди, добивавшиеся успеха, говорили им: «Послушайте, правда, не используйте C++, не используйте потоки», они отвечали: «О чем вы говорите? Вы ничего не понимаете».

Именно такие решения, как отказ от C++ и использования потоков, позволили нам выпустить продукт вовремя. Другой важной составляющей было то, что мы всегда выпускали версии под все платформы одновременно. Это решение они тоже считали глупым: «У 90% пользователей установлена Windows, так что мы сосредоточим усилия на работе версии под Windows, а позже портируем ее под остальные платформы». Так поступали многие компании, потерпевшие крах. Если вы собираетесь выпускать кроссплатформенный продукт, то история показывает, как именно не следует поступать. Если вы действительно хотите выпускать кроссплатформенное решение, то разрабатывать все нужно одновременно. А портирование приводит к паршивому результату на второй платформе.

Сейбел: Версия 4.0 создавалась с нуля?

Завински: Ну, не то чтобы совсем с нуля, но в итоге они переписали каждую строку кода. И использовали C++ с самого начала. Я боролся против этого изо всех сил и, черт возьми, был прав. Все стало распухать, появились проблемы с совместимостью, потому что когда пишешь на C++, невозможно прийти к согласию, какие именно 10% языка можно безопасно использовать. Кто-нибудь говорит: «Мне нужны шаблоны», а затем выясняется, что нет двух компиляторов, которые реализуют шаблоны одинаково.

А когда весь опыт написания кода говорит о том, что мультиплатформенный код означает работу и под Windows 3.1, и под Windows 95,

вы даже не представляете, насколько это важно. Поэтому разработка версии под UNIX (к счастью, это была уже не моя проблема) стала настоящим кошмаром, как и разработка версии для Мака. Это означало, что стал невозможным выпуск под старые версии Windows, такие как Win16. Пришлось сокращать количество поддерживаемых платформ. Может быть и пришло время так поступить, но причины были ни к черту. В этом не было необходимости.

Может показаться, что это мой горький, сугубо личный взгляд на вещи, будто мы с Терри создали великолепную штуку и теперь наказаны за свой успех, а в качестве наказания нами управляют идиоты. То время в Netscape стало несчастьем для меня. Начался период, когда я оставался в компании, ожидая, когда Netscape перейдет в другие руки.

Сейбел: И вы проработали там пять лет?

Завински: Да. И еще один год после продажи Netscape, потому что незадолго до этого стартовал проект mozilla.org, и снова стало интересно. Так что я остался из-за него.

Сейбел: Вам все-таки пришлось иметь дело с C++?

Завински: Ну, скорее с Java. В какой-то момент мы решили переписать броузер на Java. Мысли у нас были такие: «Ура! Мы выкинем весь код версии 4.0, который грозит потопить нашу компанию, и это точно сработает, ведь мы знаем, что делаем!»

Но не сработало.

Сейбел: Не сработало, потому что язык Java был еще сырым?

Завински: Нет. Нас снова разбили на четко определенные группы. Трое работали над почтовым клиентом. И мы сделали его. Мы сделали действительно отличный почтовый клиент – быстрый, со многими удобными функциями, он лучше сохранял ваши данные и никогда не тормозил при записи больших файлов. Мы воспользовались многими преимуществами механизма многопоточности языка Java, работа с которым оказалась не такой мучительной, как я ожидал. Работать на самом деле было приятно. С помощью разработанного нами API мы видели все направления, в которых он мог бы развиваться.

Кроме одной вещи, с которой он не справлялся, – отображения сообщений. Он генерировал HTML, а для его отображения нужен был слой отображения HTML, который тогда не был готов и не был готов никогда. Работа группы планирования пошла насмарку, и именно они были причиной отмены всего проекта.

Сейбел: Значит, вам приходилось бороться с сырыми на тот момент библиотеками Java для построения пользовательского интерфейса.

Завински: Нет, я бы так не сказал. Все работало. Просто в середине окна был большой белый прямоугольник, где мог отображаться только обычный текст. Они подходили к проекту очень академично, оперировали такими понятиями, как объектная модель документа (Document Object Model, DOM) и описание типа документа (Document Type Definition, DTD). «Нам нужно создать вот тут еще один уровень абстракции и создать здесь делегирование для делегирования вон того делегирования. И может быть, на экране наконец появится буква».

Сейбел: По-моему, вас сильно раздражает перепроектирование.

Завински: Да. К концу дня доделай эту чертову хрень! Хорошо, конечно, переписывать код, чистить его, чтобы с третьего раза он-таки получился красивым. Но суть-то не в этом, ты сидишь здесь не для того, чтобы писать красивый код, а для того, чтобы выпускать продукт.

Сейбел: Любители перепроектирования часто говорят: «Ну, все пойдет как по маслу, после того как мы прикрутим эту библиотеку. На самом деле мы сэкономим кучу времени».

Завински: Это чистая теория.

Сейбел: Иногда она оправдывает себя на практике, если человек мыслит здраво, а библиотека не слишком переусложнена. Тогда это действительно экономит время. Можете ли вы четко обозначить свою позицию?

Завински: Я знаю, что это банально, но всегда оказывается верным принцип «чем хуже – тем лучше». Если тратишь время на создание совершенной библиотеки, которая будет делать то, что тебе хочется, и позволит сопровождать версии от 1.0 до 5.0, все прекрасно. Но знаете что: пока три года создаешь версию 1.0, конкурент создаст аналогичный продукт за полгода – и ты вне игры. Ты никогда не выпустишь версию 1.0, потому что кто-то успел раньше.

В версии твоего конкурента, выпущенной за полгода, код – полное дерьмо, и они собираются переписать его за два следующих года, но, видите ли, они могут себе это позволить, потому что ты уже без работы.

Сейбел: Иногда, когда время поджимает, приходится выкидывать большой кусок кода, потому что кажется, что проще написать его заново.

Завински: Да, порою кое-что приходится списывать в утиль. Мне никогда не нравился такой подход, но если тебе достается чужой код, бывает проще написать все заново, чем использовать старый. Ведь нужно потратить время, чтобы понять тот код, выяснить, как им пользоваться, и понять его настолько, чтобы ты смог его отлаживать. Начать с нуля получится быстрее. Он может выполнять только 80% от того, что тебе нужно, но, может быть, именно эти 80% тебе и нужны.

Сейбел: А разве не та же логика – когда кто-то приходит и говорит: «Я не могу понять эту ерунду, я просто перепишу ее заново» – приводит к бесконечному переписыванию, что вам так не нравится в разработке программ с открытым исходным кодом?

Завински: Да. Но есть и другой аспект, помимо вопроса эффективности: намного интереснее писать свой код, чем пытаться разобраться в чужом. Так что совершенно понятно, почему так бывает. Но вся возня с Linux/GNOME – это постоянное метание между чьим-то хобби и настоящим продуктом. Что это? Исследовательский проект, с помощью которого мы экспериментируем и пытаемся понять, как должна выглядеть графическая среда пользователя? Или конкуренция с компанией Apple? Трудно заниматься и тем, и другим.

Но даже это утверждение, которое предполагает, что есть некто, принимающий решение, совершенно не соответствует истине. Все это – просто стечение обстоятельств. Вот и получается, что все постоянно переписывается, но так и не доводится до конца. Все просто прекрасно, если компьютер для тебя – это всего лишь игрушка, с которой всегда интересно повозиться, а не средство для достижения цели, не инструмент, с помощью которого доводишь интересное для себя дело до конца.

Сейбел: Кстати, насчет «интересно повозиться»: вы все еще получаете удовольствие от программирования?

Завински: Иногда. Сейчас я занимаюсь сисадминской работой, которую терпеть не могу, да и никогда не любил. Мне нравится работать над XScreenSaver, потому что в некотором роде скринсейвер (настоящий скринсейвер, а не библиотека XScreenSaver) – это совершенная программа: все делается с нуля и никаких версий 2.0. И ошибки в такой программе бывают редко. А если она падает (ай-ай-ай, произошло деление на нуль), просто ее исправляешь.

И никто никогда не попросит о новой функции для скринсейвера: «Хочу, чтобы он был более желтым». Ведь скринсейвер такой, какой есть. Вот почему для собственного удовольствия я пишу именно их. Это четкий результат, о котором не надо думать слишком много. Он не преследует тебя.

Сейбел: И вам нравятся головоломки из математики, геометрических построений и графики?

Завински: Да. Если повернуть это маленькое уравнение вот этак, что получится? Как сделать, чтобы эти шарики крутились более естественно и менее механически, чем это свойственно компьютеру? И все такое. Как сделать так, чтобы эти гармонические волны больше походили на чьи-то прыжки?

А затем я стал писать все эти мелкие глупые сценарии командной оболочки для самозащиты. Я знаю, что могу сделать это вручную, щелкнув на одной из 30 000 страниц, но почему бы не написать сценарий и не сэкономить время? Мне ничего не стоит это запрограммировать. А непрограммистам это кажется волшебством.

Мне очень понравилось портировать библиотеку XScreenSaver для Мака. Пришлось написать немало нового кода, обдумать API и всю общую структуру.

Сейбел: Вы проектировали API? Как вы структурировали код?

Завински: Я одновременно рассматривал существующие API и пытался найти лучший способ создать слой между миром X11 и миром Мака. Как мне все это структурировать? Какие API Мака подойдут лучше? Впервые за долгое время я сделал что-то и подумал: «Здорово! Пожалуй, тут я еще кое-что могу».

И это было незабываемое ощущение, потому что индустрия разработки ПО вымотала меня. Я не мог больше выносить всю эту политику как в мире корпораций, так и в среде свободных разработчиков. Я был сыт этим по горло. Я хотел работать над чем-то, где не было бы споров по всяким мелочам, и не хотел видеть, как мой продукт уничтожается бюрократическим решением, перед которым я бессилен.

Сейбел: А не было искушения вернуться и поработать над Mozilla?

Завински: Нет. У меня не было никакого желания снова погружаться в споры и эти долбаные противостояния по Bugzilla. Это совсем не весело. Но это неизбежно при создании больших продуктов. Если для работы над проектом требуется больше одного человека (что естественно для таких проектов, как Mozilla), по-другому не получится. Но я не хочу больше этих баталий, за многие годы это желание полностью выбили из меня. Как программист, я могу пойти работать где-то еще. Но мне это не нужно, да я и не хочу. Как только меня что-то достанет, я сразу ухожу. А если я организую собственную компанию, то не смогу там быть программистом, поскольку придется ею руководить.

Сейбел: Что вам нравится в программировании, кроме того что два миллиона человек пользуются вашим продуктом?

Завински: Трудно сказать. Думаю, поиск решения задачи. Это не совсем то же, что головоломка, да я и небольшой любитель головоломок. Просто пытаешься понять, как попасть из точки А в точку Б, и думаешь, как заставить машину выполнять то, что тебе нужно. Удовольствие от программирования заключается главным образом в этом.

Сейбел: Есть ли для вас понятие красоты кода? Помимо возможности поддержки, существует ли эстетическая составляющая?

Завински: Да, конечно. Когда что-то выражено очень верно – лаконично и по сути, – вроде хорошо сформулированного афоризма или мгновенной карикатуры, выполненной одним росчерком пера и очень похожей на оригинал. Что-то в этом духе.

Сейбел: Как вы думаете, программирование и писательская деятельность – это похожие интеллектуальные занятия?

Завински: В каком-то смысле да. Программирование все же гораздо строже. Но они очень близки во всем, что касается способности выражать собственные мысли. Никакого беспорядка: все, что приходит в голову и требует перевода в слова, выражаешь максимально четко. По-моему, именно этим программирование очень близко к сочинению прозы.

Мне кажется, здесь задействованы одни и те же отделы мозга, но выразить это словами нелегко. Я часто читаю код и чувствую, когда в нем что-то не так. Как и в большинстве договоров. Отсутствие гибкости, множество повторений. Смотрю на него и думаю: почему бы не разбить его на подпрограммы (которые мы называем параграфами). И то, что обычно вначале идут такие и такие определения, которые используются бла-бла-бла...

Сейбел: Поговорим о буднях программирования. Как вы проектируете код? Как структурируете его? Может быть, ваша недавняя работа над портированием XScreenSaver под OS X послужит примером?

Завински: Сначала я для затравки создаю маленькие демонстрационные программы, которые больше никем и никогда не используются. Делается это только для того, чтобы выяснить, как расположить окно на экране, и тому подобное. Поскольку я реализую протокол X11, то прежде всего беру один из скринсейверов и составляю список всех вызовов X11, которые он делает.

Потом я делаю для каждого из них заглушку и начинаю понемногу их заполнять, постепенно выясняя, как буду реализовывать тот или другой вызов.

На другом уровне, со стороны Мака, начинаем все с самого начала. Как расположить окно на экране? Затем в какой-то момент приходится прибегать к Xcode. При этом сложнее всего понять, как поднять и заставить работать систему сборки нормальным образом. Приходится экспериментировать, крутить то так, то сяк. Потом думаешь, может поместить этот кусок кода выше, чтобы он обращался вот к этому куску? А может быть, стоит вывернуть это наизнанку? Приходится перетасовывать немало кода, пока в голове не сформируется разумный поток управления. Потом я чищу код, перемещаю его в более подходящие файлы, так чтобы вот этот кусок кода был вместе с вот этим куском.

Это как рисовать жирные стрелки на схеме. Потом я перехожу к следующему скринсейверу, а он использует другие три функции, которых не было в предыдущем, поэтому мне их тоже нужно реализовать. Каждая из этих задач достаточно проста. Но над некоторыми из них приходится попотеть, потому что в API для X11 есть миллион настроек для отображения текста на экране или для поворота прямоугольника. Постепенно этот кусок кода становится все сложнее. Но большинство задач довольно просты.

Сейбел: Итак, для каждого обращения к X11 вы пишете свою реализацию. А вам никогда не казалось, что у вас набирается куча практически одинакового кода?

Завински: Конечно. После двух-трех раз вырезания и вставки похожего кода думаешь: ага, пора остановиться и поместить этот код в подпрограмму.

Сейбел: Представим, что вы снова работаете над проектом такого же масштаба, что и почтовый клиент. Вы говорили о том, что пишете несколько абзацев текста и список функциональных возможностей. Это практически все, что вы сделаете до начала работы над кодом?

Завински: Да. Может, добавлю небольшое описание различий между библиотекой и клиентской частью. А может, и нет. Если бы я работал один, то я бы с этим не заморачивался, потому что для меня это очевидные вещи. Далее, первое, что я бы сделал, – это решил, сверху или снизу начинать. Можно начать так: расположить на экране окно с несколькими кнопками, а уже потом зарываться в детали и писать логику работы этих кнопок. А можно начать с другого конца – с разбора и сохранения почтовых сообщений. Можно начать с любой стороны или одновременно с двух сторон и затем встретиться посередине.

Я заметил, что получение чего-либо на экране как можно раньше помогает мне лучше сосредоточиться на задаче и понять, что делать дальше. Ведь если смотришь на огромный список задач и не знаешь, за что взяться, то на самом деле не важно, за что возьмешься в первую очередь. Но если есть на что конкретно смотреть, пусть даже это вывод отладочной информации синтаксического анализатора почтовых сообщений, – совсем другое дело! Это уже что-то: куда будем двигаться дальше? Ладно, вместо отображения этой древовидной структуры можно заняться генератором HTML или чем-то в этом духе. Или сделать более детальный разбор заголовков. Просто ищешь очередную задачу, которую нужно решить.

Сейбел: Вы применяете рефакторинг для сохранения целостности внутренней структуры кода? Или с самого начала способны четко представить, как различные части кода будут взаимодействовать между собой?

Завински: Обычно я достаточно хорошо представляю это. Очень мало таких случаев, когда я говорил себе: «Да я же сделал все шиворот-навыворот! Придется все переставить». Но иногда это происходит.

Когда пишу первую версию программы, я стараюсь поместить все в один файл. Потом я начинаю видеть в этом файле структуру. Вот эти блоки очень похожи. Вот в нем уже тысяча строк, так почему бы не переместить что-то в другой файл? Да и API в этом случае получается естественнее. Проектирование – это непрерывный процесс, никогда не знаешь, насколько проект хорош, пока программа не будет готова. Поэтому я предпочитаю как можно скорее снять пробу, получить что-нибудь на экране и посмотреть на это со всех сторон.

И потом, начав писать код, внезапно понимаешь: «Нет, это глупая идея. И почему я решил, что этот модуль сделать легко, когда на самом деле это гораздо сложнее?» Это такие вещи, которые не понять, пока не начнешь писать код и не почувствуешь, как они от тебя ускользают.

Сейбел: Каковы признаки того, что какие-то вещи от тебя ускользают?

Завински: Когда погружаешься во что-нибудь с мыслью «Так, здесь я за полдня напишу кусок кода такой-то длины», а потом приступаешь к работе и постепенно начинаешь чуять недоброе: «Ага, нужен еще кусок, надо бы и за него взяться. Да тут работы выше крыши!»

Сейбел: Я заметил, чем хороший программист отличается от плохого: хороший программист легко переходит от одного уровня абстракции к другому, он способен не смешивать эти уровни при внесении изменений и точно определяет уровень, на котором эти изменения нужно внести.

Завински: В том, где и что размещается, нужно придерживаться определенного стиля, это очень важно во всех отношениях. Решить какую-нибудь проблему на уровне, более близком к пользователю, или внести какое-то, возможно, более крупное изменение, которое отразится на всей системе? Любое из этих решений может быть правильным, и очень сложно понять, какое из них какое. Мне нужно сделать какое-то изменение, но является ли оно единичным частным случаем или таких случаев будет 12?

Думаю, одна из самых важных вещей, по крайней мере для меня, когда создаешь что-то с нуля, заключается как раз в том, чтобы как можно скорее довести программу до состояния, когда ты, программист, сможешь ее использовать. Хотя бы немного. Это подскажет, что делать дальше, – ты буквально почувствуешь, что нужно делать. Когда что-то уже есть на экране и есть кнопка, связанная с некоторой функцией, появляется ощущение, какая кнопка будет следующей. Конечно, это GUI-центричное описание того, что я имею в виду.

Сейбел: Мы немного говорили об ужасных ошибках, с которыми вы сталкивались, например та история с GNB. Но давайте еще немного поговорим о процессе отладки. Для начала, какие инструменты вы предпочитаете? Инструкции печати отладочной информации? Отладчики исходного кода (symbolic debugger)[1]? Формальные доказательства корректности?

Завински: За последние годы многое изменилось. Когда я работал на Лиспе, процесс отладки заключался в запуске программы, ее остановке и последующем изучении данных. Был специальный инструмент, который позволял копаться в памяти, и я изменил его таким образом, что все эти функции стали доступны через цикл Чтение-Вычисление-Печать (то есть через Lisp Listener). Когда этот инструмент выводил содержимое объекта, появлялось контекстное меню, щелкнув на котором, можно было получить значение этого объекта. Все это упрощало следование по цепочкам объектов и всякое такое. Я уже тогда думал о подобных вещах: погрузиться в середину кода, искать, экспериментировать.

Позже, уже работая на Си и используя GNB в Emacs, я попытался сделать то же самое. Именно исходя из этой модели мы создавали Energize. Но мне кажется, что он так никогда нормально и не заработал. Со временем я вообще прекратил пользоваться такими инструментам, а просто вставлял инструкции печати и запускал все снова. И так раз за разом, пока не исправишь ошибку. Интересно, что при переходе на все более и более примитивные среды, такие как JavaScript, Perl, это становится единственным вариантом, поскольку там нет никаких отладчиков.

В те дни люди вообще слабо представляли, что такое отладчик. «Да зачем он тебе нужен? Что он делает – добавляет за тебя инструкции печати? Не понимаю. Что за странные слова ты говоришь?» В те дни отладка в основном заключалась в инструкциях печати.

Сейбел: Имела ли тут значение разница между Лиспом и Си? Помимо инструментов одно из отличий в том, что в Лиспе можно тестировать маленькие кусочки. Можно вызвать небольшую функцию, если в правильности работы есть сомнения, остановить ее на середине и посмотреть, что происходит. А в Си запускаешь программу целиком, во всем ее величии и сложности, и задаешь точку останова в каком-нибудь месте.

---

[1] Сейчас все уже пользуются отладчиками, позволяющими проходить по исходному тексту программ, но так было не всегда. В то время, о котором идет речь, не все среды разработки имели подобные отладчики. – *Прим. науч. ред.*

Завински: Лисп и аналогичные языки позволяют в этом отношении больше, чем Си. Perl, Python и им подобные в этом смысле немного более похожи на Лисп, но все равно мало кто так делает.

Сейбел: Но ведь GNB дает возможность заглянуть внутрь. Чем он вам не подходит?

Завински: Мне он всегда казался неприятным. Отчасти из-за того, что имеет отношение к Си. Я анализирую массив и вдруг вижу кучу чисел, и нужно лезть туда и вернуть все к нормальному виду. Он никогда не работал правильно, как мог бы работать с нормальным языком.

Сейбел: В то время как в Лиспе, если вы смотрите на массив, он выводится как нужно, поскольку отладчик знает, что есть что.

Завински: Вот именно. Мне всегда казалось, что GNB прыгает вниз-вверх, и в стеке все оказывается перепутанным. Идешь вверх по стеку, а нижняя часть стека из-за проблем в GDB изменяется. Или думаешь, что в регистре должно быть одно значение, но поскольку находишься в другом кадре стека, оно оказывается совсем другим.

Такое чувство, что я не могу по-настоящему доверять сведениям отладчика. Он что-то вывел, посмотрим – так, это число. Правильное оно или нет? Неизвестно. Зачастую вообще оказываешься без всякой отладочной информации. Берешь кадр стека, и кажется, будто у этой функции нет аргументов. Минут десять пытаешься вспомнить, через какой регистр передается нулевой аргумент. Потом сдаешься, заново компонуешь программу и добавляешь инструкцию печати.

Похоже, со временем инструменты отладки становятся все хуже и хуже. С другой стороны, люди наконец-то начинают понимать, что распределение памяти вручную – тупиковый путь. Сегодня это уже неактуально, поскольку наиболее сложные ошибки, когда приходится закапываться в структуры данных, происходят редко, ведь в Си они обычно были связаны с проблемами повреждения памяти.

Сейбел: Вы используете операторы утверждений (assertions) или другой более-менее формальный способ документирования или проверки инвариантов?

Завински: Мы тогда ходили вокруг да около, не зная, как приступиться к операторам утверждений в базовом коде Netscape. Очевидно, что добавление операторов утверждений – всегда хорошая идея как для отладки, так и, как вы говорили, для документирования. Этим выражается намерение. Мы добавили множество таких операторов. Но вопрос в том, что произойдет при нарушении утверждения в финальных (не отладочных) версиях? Что тогда? Мы склонились к мысли возвращать нулевое значение в надежде, что программа будет продолжать работать.

Ведь если броузер падает, это действительно плохо, гораздо хуже, чем возврат к циклу ожидания, большие утечки памяти или что-то в этом роде, поскольку все это меньше расстраивает пользователей.

Многие программисты инстинктивно говорят: «Выдавайте сообщение об ошибке!» Нет, не нужно. Никого это не волнует. С такими проблемами гораздо легче справиться в языках, поддерживающих исключения, таких как Java. В таких языках просто перехватываешь все исключения на самом верхнем уровне и готово. И не нужно беспокоить пользователя сообщением о том, что какое-то значение равно нулю.

Сейбел: Вы когда-нибудь просто проходили программу пошагово – для отладки или, как некоторые советуют, для проверки написанного кода?

Завински: Нет, не совсем. Обычно я использую пошаговое выполнение для отладки. Пожалуй, иногда для проверки правильности написанного кода. Но не часто.

Сейбел: Так что же вы делаете при отладке?

Завински: Сначала пробегаю глазами код, читаю его, пока не подумаю: «Так, этого не может быть, все должно работать правильно». Тогда я добавляю некоторый код для проверки и разрешения этого противоречия. Либо если, читая код, я не вижу никаких проблем, то запускаю его на выполнение, останавливаю где-нибудь в середине и смотрю, что происходит. Сложно говорить об этом вообще. Ситуации бывают разные.

Сейбел: Что касается операторов утверждений – насколько формально вы к ним подходите? Кто-то использует стандартное утверждение: «Я считаю, что в этом месте кода некоторое условие должно быть истинным». А кто-то мыслит более формально: у функций есть предусловия, постусловия и есть глобальные инварианты. Какова ваша позиция?

Завински: Я точно не думаю об этом в контексте математического доказательства корректности. Скорее я за стандартные утверждения. Конечно же, всегда полезно, получив входные данные в функции, хотя бы приблизительно понимать, какие у них ограничения. Может ли это быть пустая строка? Что-то в таком духе.

Сейбел: Тестирование – тема, весьма близкая к отладке. В Netscape была специальная группа обеспечения качества или вы все тестировали сами?

Завински: И то и другое. Мы все время запускали свои программы – это лучший способ проверки качества на месте. Но была и группа обеспечения качества, у которой были формальные тесты. И каждый раз перед выпуском новой версии они все проверяли по списку. Перейти на такую-то страничку, щелкнуть там-то. Вы должны увидеть это. Или не должны увидеть.