ПРОФЕССИОНАЛЬНО

Роберт Гласс

КРЕАТИВНОЕ программирование

2.0

Символ®

software creativity 2.0

Robert L. Glass

d.* BOOKS

ПРОФЕССИОНАЛЬНО

КРЕАТИВНОЕ ПРОГРАММИРОВАНИЕ 2.0

Роберт Гласс

Санкт-Петербург — Москва
2009

Серия «Профессионально»

Роберт Гласс

КРЕАТИВНОЕ ПРОГРАММИРОВАНИЕ 2.0

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*Т. Темкина*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Гласс Р.

Креативное программирование 2.0. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 352 с., ил.

ISBN 978-5-93286-152-3

Роберт Гласс исследует важный, но часто упускаемый из виду вопрос о роли творчества в программном инжиниринге и программировании. Почти полувековой личный опыт как разработчика, преподавателя и исследователя помогают автору охватить множество проблем. Что важнее – процесс или продукт? Каково соотношение между «интеллектуальным» и «канцелярским» трудом, а также между теорией и практикой в программировании и как сделать их взаимодействие более эффективным? В каких ситуациях полезнее подход со строгим контролем, а когда лучше свободное экспериментирование?

Первое издание, вышедшее в 1995 году, стало бестселлером. Второе издание подверглось значительной доработке. Том Демарко в своем предисловии сравнивает его со знаменитым «Мифическим человеко-месяцем» Фредерика Брукса. Первоначальный текст был дополнен результатами более чем десятилетних поисков недостающего звена между творчеством и программированием. Со свойственным автору мягким юмором и эрудированностью он дает читателю советы и рекомендации огромной важности, проливает свет на историю становления индустрии разработки ПО, исподволь подводя к мысли о том, что настоящий программист – это не оператор, не наборщик кода, а настоящий Творец.

ISBN 978-5-93286-152-3
ISBN 978-0-9772133-1-3 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 25.05.2009. Формат 70x90$^{1}/_{16}$. Печать офсетная.
Объем 22 печ. л. Тираж 1000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Всем, кто, занимаясь программированием, старается изменить мир.

«...потому что только тот, кто достаточно безумен,
чтобы надеяться изменить мир, действительно
изменяет его...»

Джек Керуак

Благодарности

Хочу поблагодарить...

...Стива Макконнелла (Steve McConnell, автор «Code Complete»[1] и многих других важных современных книг о разработке программного обеспечения) за активную поддержку и критику...

...Дэна Рида (Dan Read) из developer., который был в высшей степени любезным издателем...*

...Университет Линкопинга в Швеции, где я читал лекции, работая над первоначальным вариантом этой книги, и Университет Гриффита в Австралии, где я читал лекции, работая над ее новой версией.

Роберт Л. Гласс

[1] Стив Макконнелл «Совершенный код. Практическое руководство по разработке программного обеспечения». – Пер. с англ. – СПб.: Питер, 2006.

Другие книги Роберта Гласса

Опубликованы developer.* Books

«Software Conflict 2.0: The Art and Science of Software Engineering», 2006

Опубликованы Computing Trends

«Software 2020», 1998

«Software Folklore», 1991

«Computing Shakeout», 1987

«Computing Catastrophes», 1983

«Software Soliloquies», 1981

«The Second Coming: More Computing Projects Which Failed, with Sue DeNim», 1980

«The Power of Peonage», 1979

«Tales of Computing Folk: Hot Dogs and Mixed Nuts», 1978

«The Universal Elixir, and other Computing Projects Which Failed», 1977

Опубликованы Addison-Wesley/Prentice-Hall/Yourdon Press

«Facts and Fallacies of Software Engineering», 2003[1]

«ComputingFailure.com», 2001

---

[1] Роберт Гласс «Факты и заблуждения профессионального программирования». – Пер. с англ. – СПб.: Символ-Плюс, 2008.

«Computing Calamities», 1999

«Software Runaways», 1998

«An ISO 9000 Approach to Building Quality Software, with Osten Oskarsson», 1996

«Software Creativity», 1995

«Measuring and Motivating Maintenance Programmers, with Jerome B. Landsbaum», 1992

«Building Quality Software», 1992

«Software Conflict», 1991

«Measuring Software Design Quality, with David N. Card», 1990

«Software Communication Skills», 1988

«Real-Time Software», 1984

«Modern Programming Practice–a Report From Industry», 1982

«Software Maintenance Guidebook, with Ronald A. Noiseux», 1981

«Software Reliability Guidebook», 1979

Опубликованы IEEE Computer Society Press

«In the Beginning: Recollections of Software Pioneers», 1998

Оглавление

Предисловие Тома Демарко

В индустрии разработки программного обеспечения давно утвердилось собственное разделение на «красные» и «синие» штаты, то есть на сторонников авторитарных и либеральных методов. Одни уповают на «дисциплину» и «контроль», другие воспевают «скорость». Такое раздвоение сильно осложняет сам процесс создания программ, поскольку часто люди не способны услышать друг друга. Как и в случае красных и синих в политике, самые громкие голоса раздаются на крайних флангах, а центр представлен относительно слабо. При этом настоящая работа делается как раз на стыке между авторитаристами и либералами.

Боб Гласс активно пропагандирует компромисс, за что мы все должны быть ему благодарны. В книге «Креативное программирование 2.0» он выдвигает рациональный подход, сочетающий в себе лучшее из предлагаемого обоими лагерями; в его этике характер разработки определяется особенностями конкретного проекта, а не способностью экстремистов того или иного толка навязать свое мнение.

Однажды за ужином мой друг и коллега Шейла Брэйди (Sheila Brady), которая в 1990-х руководила проектами разработки ОС в Apple, довольно резко отозвалась о препятствиях, которые должна преодолеть женщина, чтобы стать руководителем. «Предопределенная модель управления – это патриархат», – сказала она. По ее словам, впервые в жизни мы сталкиваемся с управлением в семье, где боссом является отец, и уже поэтому представляем себе руководителя в чем-то похожим на отца. Естественно, такой руководитель любит командовать, поскольку чаще всего именно так складываются отношения между родителями и детьми. Данный результат – почти идеальный пример альфа-самца (вожака стаи). Женщины, по

наблюдению Шейлы, могут быть выдающимися руководителями, при этом нисколько не напоминая вожаков.

У многих из нас, сегодняшних руководителей организаций, где разрабатывают программное обеспечение, отцы тоже были руководителями. Поэтому наше первое представление об управлении связано с Отцом, главой семьи, но и в более позднем возрасте оно также в какой-то мере определяется Отцом – уже как руководителем той или иной организации. То, что когда-то рассказывал Отец о своей конторе или фабрике, запоминается на всю жизнь и влияет на нашу работу, даже мало соответствуя нынешним реалиям.

Например, мой отец руководил литейным производством. Его героями были Тэйлор («Научная организация труда») и Гилбрет («Анализ движения»). Когда мы с ним говорим об управлении, даже нашей маленькой семье не обойтись без раскола на красных и синих. Стоит упомянуть незаменимых работников, как он выходит из себя: «Если бы я обнаружил, что у меня есть незаменимый работник, то немедленно уволил бы его!» Он вырос в Век индустриализации, а я – в Век информации. И управление в эти две разные эпохи развивалось по двум совершенно разным направлениям.

Не подумайте только, что управление, свойственное Веку информации, просто заменило собой управление Века индустриализации. Остатки мышления индустриальной эпохи крепко сидят в глубинах современной организации. Это понудило нас искать, как священный грааль, новые методы, пытаться систематизировать саму систематизационную работу, с помощью которой мы преобразовывали свои организации. Обычно это приводило к столкновениям. Разумное обсуждение методов нередко тонуло в перебранке:

Ты – ковбой.

А ты – любитель покомандовать.

Хакер!

Фашист!

Дело усугубилось тем, что научное сообщество тоже раскололось на два лагеря. Эмпирики и специалисты по психологии познания постоянно разъясняли, что талант выше методов: некоторые разработчики настолько умелы в своем деле, что остальным при всем желании за ними не угнаться, и им остается только тиражировать методы своих более талантливых коллег. В то же время формалисты утверждали, что *есть* единственный правильный метод разработки программного обеспечения, и он настолько систематичен, что можно убедительно доказать его правильность.

Фактически он настолько систематичен, что на его основе можно организовывать промышленное производство.

Но есть еще реальный мир. И тут с блеском проявляет себя Боб Гласс. В книге «Креативное программирование 2.0» он вдумчиво, методично и убедительно излагает свои мысли. Очень часто он делает это творчески, демонстрируя в книге ту изобретательность, за которую ратует в создании программ. Например, обратите внимание на пример с трапециевидной призмой – это что-то вроде чудесного озарения, неожиданного, впечатляющего и глубокого.

Поскольку для XXI века характерны глобализация, коренные перемены и сильная конкуренция, никто не сомневается в усилении роли творчества. В результате теперь гораздо больше говорят о поддержке творчества. На словах все «за». Едва ли найдется компания, руководство которой не разглагольствовало бы о творчестве и его *огромной* роли. Но часто этим оно и ограничивается. Надеюсь, эта книга поспособствует тому, чтобы поддержка творчества в нашем деле продвинулась хотя бы немного дальше слов.

Том Демарко
Кэмден, Мэн

Предисловие к первому изданию

Книга называется «Креативное программирование». Она написана на основании глубокой личной убежденности автора в том, что:

- создание программного обеспечения – это в первую очередь решение задач;
- решение задач всегда требует творческого подхода;
- решать задачи в программировании очень сложно – возможно, сложнее, чем в других видах деятельности;
- следовательно, решение задач в программировании требует максимально творческого подхода.

Я начал писать эту книгу из других побуждений. Меня утомили и разозлили многочисленные утверждения о том, что задачи разработки ПО решаются с помощью дисциплины, формализма и количественного подхода. Мой 40-летний опыт практического решения задач программирования и 6 лет научной работы, в течение которых я размышлял о связи этого опыта с тем, что пишут о программировании, привели меня к противоположному выводу: задачи разработки ПО решаются с помощью гибкости, творчества и качественного подхода. Разумеется, для крупных проектов, над которыми работает много людей, дисциплина – это неизбежное зло; и все же я считал, что прогресс осуществляют выдающиеся люди, а не выдающиеся технологические процессы.

Но стоило мне приступить к работе над книгой и в поисках материалов в поддержку творчества обратить внимание на другие области деятельности, как началась моя собственная одиссея. Я стал замечать, что такое значение дисциплине, формализму и количественному подходу придается не зря. Как бы ни нуждалась какая-то сфера деятельности в творчестве,

всюду необходима более формальная основа для его поддержки и укрепления.

Эта мысль часто проскальзывала в довольно неожиданных местах. В книге, многозначительно озаглавленной «The Discipline of Curiosity» (Дисциплина и любознательность) [Groen 1990]), автор писал: «Наука – не просто любознательность и творчество, это их дисциплинированная форма... Именно странный на первый взгляд союз дисциплины и любознательности обеспечивает научный прогресс. Любознательность творческого ума, заставляющая непрерывно задавать вопросы „как?“ и „почему?“, и дисциплина, чтобы понять: наука – это часть мира, и она влияет на него».

В другой книге, посвященной решению задач [Judson 1980], автор передает свой разговор с Мюрреем Гелл-Манном, открывателем кварков: «Со всяким искусством, если оно заслуживает этого названия, связана некоторая дисциплина. Даже если речь идет не о сонетах, симфонической музыке или классической живописи, а о весьма раскрепощенном современном искусстве, то и там действуют те или иные законы. И задача в том, чтобы донести то, что вы хотите, соблюдая эти законы».

И еще, из интервью с виолончелистом Ма Йо-Йо [Shapiro 1991]: «Творчество... не инстинктивно. Скорее это непрерывная борьба дисциплины с интуицией. Дисциплина – годы упражнений, музыкальное образование – основа исполнительского мастерства. Но в какой-то момент музыкант должен довериться своим чувствам...»

Одна и та же мысль повторялась снова и снова. В науке требуется как дисциплина, так и творчество. В изобразительном искусстве требуется как дисциплина, так и творчество. В музыке требуется как дисциплина, так и творчество. Очевидно, к моему разочарованию, и в программировании также требуются и дисциплина, и творчество!

Моя одиссея была почти завершена. Теперь я понимал, почему там, где творчество представлялось столь важным для достижения успеха, постоянно вставал вопрос дисциплины. Но тогда возникла новая дилемма: есть ли вообще смысл писать книгу?

Я перечитал компьютерную литературу, вновь отыскивая те места, где говорилось о дисциплине, формализме и количественном подходе. И снова, несмотря на мою одиссею, во мне поднялось возмущение. Я увидел, что защитники дисциплины и формализма делали ту же ошибку, что и я. Они полагали, что формализм продуктивен, а творчество нет. Совсем, как я, когда считал необходимым условием только творчество, а формализм – помехой.

Возможно, книгу все же стоило писать! Она могла бы оказаться полезной тем, кто слишком далеко отклонялся вправо, подобно тому как я откло-

нялся влево, помогая им увидеть, что правильный подход к решению задач расположен где-то между этими крайностями. А тем, кто не был столь предвзят, как мои «оппоненты» и я сам, она могла бы помочь разобраться в сути весьма важной проблемы, существующей в нашей области.

Мне не понадобилось менять собственные коренные убеждения. Тем не менее пришлось расширить их, включив в них этот «странный союз», эту «бесконечную борьбу», взаимодействие между одним существенным компонентом, творчеством, и другим существенным компонентом, дисциплиной. Моя одиссея наконец-то завершилась.

Мое пожелание и моя надежда – чтобы, читая эту книгу, вы проделали то же путешествие, которое совершил я, и остались довольны тем, что отправились в него.

Роберт Л. Гласс
Лето 1994 г.

Ссылки

GROEN 1990 – *The Discipline of Curiosity*, Elsevier Science, 1990; Groen, Jenny; Eefke Smit; Juurd Eijsvoogel.

JUDSON 1980 – *The Search for Solutions*, Holt, Reinhart and Winston, 1980; Judson Horace Freeland.

SHAPIRO 1991 – «Yo-Yo and Manny», *World Monitor*, Aug., 1991; Shapiro Michael.

Предисловие ко второму изданию

Забавно получилось с этой книгой.

Первое издание поначалу оказалось коммерчески неудачным. Книга плохо продавалась, авторский гонорар был невелик, отклика читающей публики почти не было.

Я был огорчен, но не пал духом. Чтобы написать эту книгу, я проделал большую исследовательскую работу, и то, что я узнал в результате, имело для меня большое значение. Я доволен тем, что, работая над книгой, сильно вырос интеллектуально. Из нескольких написанных мной книг – на сегодня – этой я горжусь, пожалуй, больше всего. В ней достигнут правильный баланс – по крайней мере, по моим представлениям – легко воспринимаемого стиля изложения и должного уровня научных исследований.

Спросите, что тут забавного? Дело в том, что, несмотря на слабое начало, количество проданных экземпляров стало расти. На конференциях и по электронной почте люди сообщали мне, как им понравилась книга. Все чаще и чаще. У меня появилось ощущение, что книга становится культовой. На рубеже тысячелетий я понял, что в мире программирования ее начинают воспринимать во многом так же, как я сам.

Поклонников книги становилось все больше. Остававшиеся в издательстве экземпляры были распроданы. Теперь, когда книги уже не было ни в печати, ни на складе, подорожали подержанные экземпляры. Из жадности я продал на *Amazon.com* свой запасной экземпляр почти за 100 долларов! Цена продолжала резко подниматься. Прямо сейчас, сочиняя новое предисловие, я открыл окно Amazon и смотрю на цену – 1196 долларов! Трудно поверить, что написанное мной может столько стоить! Жизнь явно требовала новой и улучшенной версии 2.0 «Креативного программирования». Работа началась.

Я привел имевшиеся заметки в соответствие с сегодняшним днем (за 10 лет в мире компьютеров и программирования многое произошло!). Я добавил несколько новых статей, чтобы отразить изменения и новые веяния.

Я постарался отразить гибкую методологию разработки (Agile), поскольку она близко связана с понятиями гибкости, творчества и качественного подхода, с которых началась моя одиссея (спешу добавить, что им может понадобиться пройти отчасти тот же путь, что и мне, чтобы приспособиться к представлениям о дисциплине и формализме, ставшим результатом моей одиссеи).

Я также добавил материал о методике Open Source – возможно, одном из последних бастионов представления об удовольствии как о мотивирующем факторе создания программного обеспечения.

Однако сердцевину книги обновлять не потребовалось. Одиссея, описанная мной в предисловии к первому изданию, все так же актуальна.

Замечательно, что все так забавно получилось. Кстати, благодарю всех поклонников книги за то, что исчезнувшая, не издаваемая и отсутствующая на прилавках книга стала пользоваться таким высоким спросом, который превзошел мои самые смелые мечты!

А тем новым читателям, которые впервые встречаются с «Креативным программированием», желаю радости от прочтения!

Роберт Л. Гласс
Лето 2006 г.

Почему «Креативное программирование»?

Для творческого мышления нужна позиция, сочетающая поиск идей и умелое обращение со своими знаниями и опытом. Встав на эту точку зрения, вы опробуете разные подходы,... часто не приводящие никуда. Вы используете безумные, дурацкие и непрактичные идеи, чтобы достичь практичных новых идей. Время от времени вы нарушаете правила и ищете новые идеи в непривычных местах.

Рождер фон Эйк

«A Whack on the Side of the Head»,
Warner Books, 1990

Миру программирования присущ принципиальный конфликт, иногда принимающий характер войны.

С одной стороны – руководители, которые стремятся навязать разработчикам более высокий уровень дисциплины и контроля, и исследователи, которые с теми же целями предлагают, а порой даже пытаются узаконить формальные методы.

С другой стороны – разработчики программного обеспечения, спокойно продолжающие писать программы обычным путем, то есть свободно выбирая методы и применяя творческие решения.

Время от времени появляются «революционные» методы, которые какое-то время пользуются популярностью и затем исчезают, мало что изменив в существующей практике разработки программ.

Методологии дробятся, языки программирования приходят и уходят, CASE-средства пылятся на полках, ориентация на объектность вечно остается «задачей следующего года». Попытки упростить и регламентировать разработку программного обеспечения в целом проваливаются одна за другой.

В чем причина такой ситуации? Может быть, в неподатливости разработчиков, которые настолько закоснели в своих привычках, что не способны воспринимать новое? Похоже, руководители и исследователи именно так и полагают.

А может, дело в невежестве руководителей и исследователей, которые настолько озабочены поиском новых решений, что перестали понимать старые? Такая точка зрения характерна для практиков.

Думаю, главная проблема этого конфликта – творчество. Если при создании программ творчество почти или совсем не нужно, то правы руководители и исследователи, заявляющие, что процесс изготовления программ можно упростить и регламентировать. Если же, напротив, творчество было и остается важной составной частью процесса создания программ, то правы практики: нам по-прежнему будут нужны свободные методы и творческие решения. Вот эту-то совокупность проблем, возникающих из представлений о творчестве в программировании, я и собираюсь рассмотреть в книге и, если удастся, решить. Потому я и назвал ее «Креативное программирование».

Рассмотрению этих диаметрально противоположных точек зрения и возможных путей выхода из конфликта посвящена первая часть книги. Я разделю проблему на составные части. Вместо того чтобы рассматривать творчество в целом, мы последовательно изучим его с разных сторон, взяв такие его аспекты, как дисциплина, формальные методы, оптимизация решений, количественный подход, технологический процесс, теория и занимательность. Эти и некоторые другие темы будут изучены в ряде неформальных очерков. Попутно мы рассмотрим пару дефиниций творчества и даже результаты исследования, имевшего целью выяснить, какие же задачи разработки программного обеспечения действительно требуют творческого подхода.

Затем в части II мы перейдем к несколько более формальной и практической точке зрения на творчество. Мы возьмем две области, в которых творчество играет особо важную роль – творчество в организационной сфере и творчество в технологиях программирования, – и изучим значение творчества в них существенно глубже. А после, чтобы более полно

осветить творчество в программировании, мы взглянем на наш предмет с исторической точки зрения: где и как возникла проблема творчества? Для каждой из этих тем мы рассмотрим пути стимуляции творчества. Для этого есть чертова дюжина приемов, которые мы опишем и обсудим.

Чтобы не ограничиваться при рассмотрении творчества точкой зрения одних лишь программистов, в части III мы рассмотрим то, что узнали о творчестве, применительно к другим областям. В частности, нас ждет несколько удивительных ответов на вопрос: «Могут ли компьютеры и программы повысить уровень творчества в других (некомпьютерных) областях?».

В части IV мы увяжем свободные концы всех нитей книги.

Это обзор структуры книги, но как она построена с эмоциональной точки зрения? Куда она приведет нас в итоге?

Надеюсь, по мере чтения книги вы станете по-новому воспринимать роль творчества и тот конфликт, с рассказа о котором я начал. Где-то я подам идею, где-то подскажу – все это поможет вам сформировать собственное мнение относительно важности творчества в разработке программного обеспечения. Изучение материалов для этой книги стало для меня путешествием чувств и ума, уведя далеко от того места, где все началось. Такое может случиться и с вами.

Попутно я призываю на помощь некоторых коллег. Например, у известного программиста П. Дж. Плогера (P. J. Plauger) мы позаимствуем понятие «индекса сложности» («the falutin' index») – легкомысленного способа оценки программных проектов, весьма отличающегося от формальных метрик, о которых можно прочесть в формальной литературе. Аналогично Брюс Блюм (Bruce Blum) из Лаборатории прикладной физики имени Джона Хопкинса поможет нам изучить соотношение между развлечением и скукой в программных проектах, а Дуэйн Филипс (Dwayne Phillips) будет руководить нашим исследованием забавной стороны некоторого проекта. Деннис Галетта (Dennis Galletta), профессор из Питта, расскажет о творчестве в стратегических системах. Покойный Дэн Кугер (Dan Couger) из Университета Колорадо (и директор Центра изучения творчества в Колорадо-Спрингс) – автор ряда идей о творчестве в программировании, которые я позаимствовал из его книг. С Айрис Весси (Iris Vessey) из Университета Пенсильвании мы вместе исследовали природу задач программирования и описали результаты. Таким образом, данная книга отчасти является коллективным трудом.

Часть I

Два типа мышления: исследование творчества в области программирования

Введение

1. Дисциплина и гибкость
2. Формальные методы и эвристики
3. Оптимизация и разумная достаточность
4. Количественный и качественный подходы
5. Процесс и продукт
6. Интеллектуальные и канцелярские задачи
7. Теория и практика
8. Наука и производство
9. Забавность и серьезность

Введение

Наиболее успешно решают задачи те, кто обладает редким умением объединять режимы сознательного функционирования, свойственные разным полушариям мозга, переключаясь в обоих направлениях между целостным и последовательностным, между интуицией и логикой, между расплывчатым пространством проблемной области и маленьким четким конкретным участком этого пространства. Это могут быть выдающиеся художники и ученые, потому что они сочетают в себе сильные стороны как... Леонардо да Винчи ... так и Эйнштейна.

Моше Ф. Рубинштейн

«Tools for Thinking and Problem-Solving»,
Prentice-Hall, 1986

К написанию книги о творчестве в программировании можно подойти по-разному:

- слегка коснувшись проблем творчества, привести массу советов по поводу того, как проявлять больше творчества, и множество примеров их успешного применения;
- достаточно серьезно и академически, переработав массу литературы, дословно описать все, что известно по различным аспектам данной темы;
- изложить неофициальный взгляд на важность творчества для разработки программного обеспечения.

В этой книге нашли отражение все три подхода, хотя автор явно склоняется к последнему. В части I ему отдано предпочтение. Следующие статьи в целом представляют собой неофициальное собрание мыслей о важности творчества в программировании.

Как мне кажется, данная проблема описывается рядом известных и считающихся ключевыми проблем в области программирования:

- роль дисциплины;
- роль формальных методов;
- роль оптимизации решений;
- роль количественного подхода;
- роль технологического процесса;
- роль теории;

- роль науки;
- роль занимательности.

Думаю, в действительности каждая из перечисленных проблем имеет отношение к творчеству. Уверен, что при творческом подходе к созданию программ мы:

- освобождаемся (хотя бы частично) от дисциплины;
- переходим от более формальных методов к менее формальным;
- выбираем решения, достигающие цели (возможно, не самым оптимальным образом);
- отдаем предпочтение не количественному, а качественному подходу;
- интересуемся как процессом, так и его результатом;
- понимаем, что иногда успешная практика опережает успешную теорию и что некоторые секреты производства науке неизвестны;
- и получаем больше удовольствия от работы!

В духе этой дихотомии – что творчество означает разрыв со стандартными шаблонами – я и выбрал слова «типы мышления» для названия этой части книги. Каждый из следующих заголовков означает противопоставление: формальных методов – эвристикам, количественного подхода – качественному, практики – теории. Уверен (и книга подтверждение тому), что в современном мире дисциплинированному, формальному подходу к разработке программного обеспечения придается чрезмерное значение. Я хочу сказать, что есть не только альтернативы общепринятому здравому смыслу, но и веские основания перейти от него к более взвешенной точке зрения на рассматриваемую область деятельности.

В этой части не только даются ответы, но и ставятся новые вопросы. Всегда ли дисциплина – правильный метод для программного проекта? Права ли компьютерная наука, делая столь сильный упор на формальные методы? Верно ли, что «нельзя управлять тем, чего не можешь измерить»? Всегда ли теория опережает практику? Должны ли серьезные размышления вытеснить элемент развлекательности в нашем деле? Пытаясь ответить на эти вопросы, я сталкиваюсь с другими и ставлю новые.

Однако есть и однозначные ответы. Научное исследование задач программирования позволило выяснить, в какой мере эти задачи являются интеллектуальными, канцелярскими и творческими. Некоторые из результатов этого исследования могут удивить вас.

Глава 1 | Дисциплина и гибкость

Введение

Наука – не просто любознательность и творчество, это их дисциплинированная форма... Именно странный на первый взгляд союз дисциплины и любознательности обеспечивает научный прогресс. Любознательность творческого ума, заставляющая непрерывно задавать вопросы «как?» и «почему», и дисциплина, чтобы понять, что наука – это часть мира, и она оказывает на него влияние.

Дженни Гроэн

«The Discipline of Curiosity», Elsevier Science, 1990;
Janny Groen, Eefke Smit, Juurd Eijsvoogel

Работая над этой книгой, я с трудом подобрал понятие, противоположное «дисциплине». Очевидный кандидат «недисциплинированность» был отвергнут из-за негативного оттенка. При том, что никто и не собирается предлагать полный отказ от дисциплины при разработке программного обеспечения. Всякий раз, возвращаясь к этой теме в надежде на озарение, я вновь оказываюсь у пустого колодца. Я бы хотел различать управляемое поведение в едином русле (дисциплинированное) и целенаправленное, но не связанное ограничениями поведение (творческое).

Я не придумал ничего лучше, чем «гибкость». Имея в виду такие аспекты дисциплины, как подчинение и контроль, я выбрал «гибкость» в противоположность им. В моем словаре «гибкий» (flexible) означает «способный легко гнуться, не ломаясь; приспосабливающийся, способный меняться в зависимости от обстоятельств».

Меня устраивает это определение, хотя я не вполне уверен, что правильно противопоставил его дисциплине. То есть этот вариант меня не совсем удовлетворяет, но, думаю, сгодится и он. Если найдете более удачное слово, мысленно подставьте его в заголовок этой главы!

Закончим на этом игры со словами. Почему в книге о творчестве в программировании появилась глава о дисциплине и (...э-э...) гибкости? Потому что одно из древнейших противостояний в нашей профессии связано именно с этим вопросом: какой должна быть команда разработчиков программного обеспечения – дисциплинированной, как на заводе, или гибкой и с высокой личной мотивацией?

Разумеется, это противостояние старше программирования. Когда мир перешел от кустарного производства к фабричному, стало совершенно яс-

но – по крайней мере, на тот момент, – что будущее принадлежит дисциплинированным, а не гибким способам организации. Век автомобилей, конечно, не настал бы без Генри Форда с его идеей неквалифицированной, но дисциплинированной рабочей силы.

Но это было давно, а сейчас времена переменились. Многие согласятся, что эру фабричных труб сменила эра информации. Общество, которое раньше зависело от умения что-то производить из дорогого сырья на огромных фабриках, теперь зависит от информации, обрабатываемой с помощью дешевого кремния. Будущее принадлежит работникам сферы знаний, а не заводским рабочим. Вопрос в том, как найти оптимальные способы организации и управления командами работников сферы знаний.

Несомненно, разработка программного обеспечения – это вершина обработки информации. Если вы согласны хотя бы отчасти поверить заявлениям Дэвида Парнаса и Фреда Брукса, что «программировать тяжело» и что «это самый сложный вид деятельности, которым когда-либо занимались люди», то нетрудно предположить, что прежние модели организации труда на фабрике оказываются непригодными в эпоху информации.

Любопытно отметить, что сегодня успешные компании-производители программного обеспечения финансируют разработку программ по бюджетной статье «Исследования и разработки», то есть рассматривают создание программ не как вид промышленного производства, а как деятельность, более близкую к интеллектуальному труду исследователя.

А теперь задумаемся об исследованиях и разработках. Характерны ли для них «подчинение» и «контролируемое поведение»? Разумеется, нет. Дисциплина может иметь значение для исследований и разработок (мы глубже изучим эту тему в последующих разделах), но гораздо важнее здесь гибкость – или другое слово, которое вы подберете для этого понятия.

В таком случае соотношение ролей дисциплины и гибкости должно задаваться моделью наших представлений о разработке программного обеспечения. Что правильнее: модель фабричного типа, когда рабочие целый день штампуют детали и собирают из них программу на конвейере, или что-то вроде мастерской или лаборатории, когда разработчики целый день решают интеллектуальные проблемы?

Этой проблемой мы и займемся в данном разделе. Вопрос не нов: чтобы показать это, я включил в книгу (на нашем профессиональном языке можно сказать «повторно использовал»!) статью «Генри Форд от программирования, пожалуйста, встаньте!» примерно тридцатилетней давности, но, как мне кажется, хорошо описывающую проблему. А заключительный раздел «„Странная парочка“ – дисциплина и творчество» отражает мое личное разрешение данного противоречия.

Вперед! И заодно подумайте о том, как бы вы сами разрешили противоречие между дисциплиной и гибкостью.

1.1. Генри Форд от программирования, пожалуйста, встаньте!

Расстояние между штаб-квартирами фирм Alchemy Chemical и Softli Paper не больше мили, если ехать по Прямому шоссе в Кремниевом содружестве (Массачусетс).

Но их производственные мощности с таким же успехом могли располагаться на Марсе и Венере, потому что их организационную философию разделяют световые годы. И занимаемые ими крайние позиции характерны для современных дилемм в программировании.

В Alchemy твердой рукой правит Стэн Сорсер. Его служащие придерживаются установленного дресс-кода, работают от звонка до звонка, пьют кофе в отведенные для этого перерывы и строго выполняют правила поведения. Для программистов установлены многочисленные и обязательные стандарты. Документация и листинги подвергаются периодическому контролю и внезапным проверкам. Все приложения пишутся только на C# и Java, а для добровольно принудительного соблюдения стандартов программирования применяются средства статистического анализа. Руководство строго следит за выполнением графика работ, и если программист не укладывается в срок, он должен ликвидировать отставание, добровольно работая сверхурочно. Обстановка в Alchemy напряженная, деловая и продуктивная. Стэн любит демонстрировать данные, согласно которым производительность труда у его программистов втрое выше, чем в среднем по стране. По всем внешним признакам Стэн построил фабрику, производящую программное обеспечение, и руководит ей, как сборочным конвейером.

А глава Softli Paper Херб Бонд смотрит на все иначе. Его программисты одеваются в стиле «неохиппи». Двери Softli открыты круглые сутки, программисты приходят и уходят. Инструкция по стандартам программирования здесь тоньше предметного указателя Alchemy и вместо «должен» и «обязан» в ней говорится «может» и изредка «следует». У Херба принято, чтобы код рецензировали коллеги-программисты, а не проверяли начальники. Из языков здесь преобладают C# и Java, но один из асов-программистов пишет сейчас генератор отчетов на Python и очень доволен. Управление заданиями осуществляется с помощью отчетов о ходе работы, причем качеству продукта уделяется больше внимания, чем соблюдению графика. Свободная, неказенная атмосфера в Softli напоминает семинары на старших курсах. Херб никогда не приводит статистические

показатели по продуктивности, но как-то обмолвился, что не доверяет им, зная, откуда берутся эти фиктивные числа. По всей видимости, Херб Бонд пасет табун диких необъезженных программистов, направляя его движение в сторону выпуска готового продукта.

Совершенно очевидно, что Стэн Сорсер и Херб Бонд не могут быть оба правы относительно того, как надо управлять программным производством. Программисты должны работать либо как рабочие на конвейере, либо как художники. И все же – как?

Ответить на этот вопрос не так легко. Десять лет назад казалось, что философия сборочного конвейера – единственно верная. Программные продукты отличались чрезмерной сложностью, непомерной дороговизной и отставанием от графика. Программные модули изготавливались и собирались вручную мастерами. Казалось очевидным, что характерные для программных продуктов проблемы обусловлены кустарным способом производства.

Исходя из такого предположения руководство выделило немалые средства на разработку технологий конвейерной сборки программных продуктов. Первые надежды возлагались на Visual Basic – язык, поддерживающий перетаскивание и мастера для разработчика, с помощью которого даже любители и начинающие программисты могли создавать приложения с невиданной ранее скоростью.

Однако расчеты не оправдались. Как ни крути, программы остаются программами. Их индивидуальные особенности и сложность таковы, что одним перетаскиванием не обойтись. Visual Basic не смог изменить ситуацию (как позже не смогли Java и C#).

Но руководство не теряло надежд. Поощрялись технологии, предлагавшие стандартизацию. Значение индивидуальности принижалось, превозносилась групповая работа. Возник термин «неэгоистичное программирование». Были формализованы управление настройками и контроль качества. А когда появилось автоматизированное управление жизненным циклом, сулившее чудеса, руководство с готовностью ухватилось за него. Казалось, что земля обетованная – поточная линия программирования – уже на горизонте.

Но в этом стремлении к максимальной автоматизации была изрядная доля иронии. В то время как исследователи отдавали все свои силы разработке технологий конвейера, почти никто не уделял внимания мастерству технологов. Несмотря на запуск конвейеров программные продукты по-прежнему создавались мастерами-умельцами. И никто не занимался разработкой и продвижением усовершенствованных инструментов для этих умельцев. Ну, то есть не совсем так. Были технологи, которые создавали более эффективные инструменты. Но почти никто не обращал на

них внимания. Если новая технология не предвещала быстрого приближения к поточной линии, редкий руководитель удосуживался вникнуть в нее. А без поддержки руководителей новые инструменты не могли попасть к программистам-практикам.

Хочу внести полную ясность в один вопрос. Я не отрицаю, что в один прекрасный день программирование может встать на поток. Не исключено, что где-то уже сейчас некий Генри Форд от программирования разрабатывает совершенный метод изготовления программ. Но пока исследователи и инвесторы заняты этим делом, кто-то должен решать более близкие задачи. Глупо поспешно переводить отрасль на конвейерное производство, когда никто не знает, как проектировать этот конвейер. И глупо прекращать разрабатывать инструменты и применять их, оттого что когда-нибудь они, возможно, больше не понадобятся.

Однако хватит разглагольствовать. Я, пожалуй, слишком отклонился от Прямого шоссе, Стэна Сорсера и Херба Бонда.

А может быть, и нет. Фирма Стэна Сорсера – это высшее современное достижение в организации конвейера. А фирма Херба Бонда – вершина подхода, основанного на привлечении творцов. И мои разглагольствования были попыткой объяснить, почему в действительности нельзя утверждать, что Стэн с его цифрами прав, а Херб – нет. Или наоборот.

Но если вернуться к Alchemy и Softli, то очевидно, что противоречие между этими двумя компаниями не прошло незамеченным для их сотрудников. Дело в том, что в пятницу вечером в баре на Прямом шоссе бывало очень интересно, если там сходились люди из Alchemy и Softli. В один из таких пятничных вечеров и произошло описываемое ниже событие.

Представьте картину. Ровно в 16:43 у бара останавливается пара машин с программистами Alchemy, возвращающимися с работы. Одетые строго по дресс-коду в рубашки поло и модельные джинсы, они с некоторым презрением поглядывают на бородатых парней из Softli в мешковатых шортах и сандалиях, потягивающих в уголке свои напитки. Взгляд не остается незамеченным, и один из двоих программистов Softli, засидевшийся в баре дольше, чем позволяет истекающий завтра срок сдачи работы, презрительно отзывается о стиле одежды и родословной программистов Alchemy в целом и данной их группы в частности.

Если бы в баре собирались заводские ребята, уже мелькали бы кулаки. Но заведению присуща некоторая интеллектуальная атмосфера, и вместо этого возникла изящная перебранка. В ход пошли разные идиоматические выражения. Сражение еще только началось, а дресс-код Alchemy уже был назван *гитлеровским*, манера одеваться в Softli – *стилем бездельников*; распорядок дня Alchemy – *бухгалтерским*, продуктивность Softli – *высо-*

кой, но отрицательной, дисциплинированное программирование в Alchemy – *упорядоченным кретинизмом*, а характер труда в Softli – *детсадовским*. Перепалка разгоралась.

Не знаю, кто первым предложил провести турнир. И не уверен, что в конечном счете это была верная идея. Но остановить сражение удалось. Высказанная идея прозвучала как колокол в суде, заставив всех умолкнуть, после чего началось бурное обсуждение условий. Идея была крайне проста, типа «мой папа побьет твоего папу». «Сделай, что можешь, или заткнись!» – вот и весь разговор. «Если вы такие способные программисты, посмотрим, сможете ли вы нас переплюнуть».

Идею подхватили все. Когда вечером эта разношерстная публика покинула бар, основные правила были уже оговорены.

Турнир был достаточно прост. Команды от обеих компаний должны были написать две программы. Первая – генератор отчетов, относительно эффективный и незамысловатый. Вторая – небольшой интерпретатор языка с парой сложных типов данных и мудреной семантикой. Оцениваются время написания программы, стоимость написания и качество готового продукта. Качество, естественно, определяется субъективно с учетом некоторой таблицы атрибутов качества, призванной способствовать объективности оценки и составленной судейской бригадой из сотрудников обеих компаний. Проводить соревнования планировалось по утрам в баре в течение четырех суббот подряд.

В эти субботы в баре не смотрели кубковые футбольные матчи. Помещение было набито болельщиками из обеих компаний, потому что весть о турнире разнеслась по всем ИТ-отделам. Перфораторщицы из Alchemy в одинаковых свитерах и брюках подобающей скромной расцветки смотрелись как импровизированная группа поддержки. Группа штатных аналитиков из Softli организовала блог и RSS-трансляцию, чтобы сообщать о ходе соревнований в реальном времени. На третью субботу пришло местное отделение Ассоциации по вычислительной технике в полном составе, а на четвертую явились репортеры из «Прямошоссейного вестника» и «Трубного гласа». Событие стало крупнейшим в Кремниевом содружестве после урагана Гизельда.

Результаты турнира – которых вы ждали, затаив дыхание, – разочаровали. Если не придираться к подсчету очков, то получилась ничья. Конвейерщики из Alchemy легко справились с генератором отчетов и показали хорошие результаты по времени и стоимости. Зато умельцам из Softli лучше дался интерпретатор, и баллы за качество у них были выше. Для такого рода соревнований исход оказался, видимо, лучшим из всех возможных – проигравших не оказалось и никто не испытывал неловкости, когда в последнюю субботу толпа расходилась из бара.