7 Почему так трудно научиться программировать

Марк Гуздьял

Большинство вопросов, рассматриваемых в этой книге — оптимальные способы разработки программных продуктов, затраты на разработку, организация общения между разработчиками — так или иначе относится к программистам. Однако стать программистом не так легко. Мало кто пытается вступить в эту область, и еще меньше людей добиваются успеха. В этой главе мы попробуем разобраться, почему же так трудно научиться программировать?

Прежде всего необходимо понять, действительно ли нам нужно больше программистов? Бюро трудовой статистики США недавно предсказало огромный рост спроса на профессионалов в компьютерных областях. Согласно отчету за ноябрь 2008 года [Association, 2008], спрос на IT-профессионалов за период с 2006 по 2016 год вдвое превысит темпы роста трудовых ресурсов. В обновленной оценке за ноябрь 2009 года говорится: «Компьютерные и математические профессии составляют наиболее быстрорастущую профессиональную подгруппу в пределах самой быстрорастущей профессиональной категории» [Consortium, 2010]. Но что следует понимать под «IT-профессионалом»? «Компьютерной и математической» профессией? Опыт многих безработных IT-профессионалов, особенно во время текущей рецессии, наводит на мысль, что, возможно, в США сейчас слишком много программистов [Rampell, 2010].

Даже если ответ на вопрос о том, нужно ли нам больше программистов, не так очевиден, совершенно ясно, что многие люди вступают на путь программирования и относительно рано терпят неудачу. Слухи о высоком проценте неудач на вводных компьютерных курсах (часто обозначаемых сокращением «CS1» в соответствии с ранними стандартами оформления резюме) постоянно встречаются в литературе и в кулуарных беседах на конференциях — таких как симпозиум ACM SIGCSE (Special Interest Group in Computer Science Education). Йенс Беннедсен и Майкл Касперсен сделали первую осмысленную попытку определить действительный процент неудач [Bennedsen and Caspersen, 2007]. Они запросили данные из учебных учреждений по всему миру по образовательным спискам рассылки. 63 учреждения предоставили информацию о проценте неудач на своих вводных курсах; таким образом, решения об отборе и предоставлении данных принимались самими учебными заведениями (например, участники с особенно неудачными результатами могли отказаться от

предоставления данных или предоставить неверные данные, что могло привести к искажению результатов из-за нарушения случайности выборки).

В целом 30% студентов «проваливаются» или уходят с первого курса, причем в колледжах процент неудач выше, чем в университетах (40% вместо 30%). Получается, что примерно треть студентов, поступающих на курсы CS1 в разных учебных учреждениях по всему миру, терпит неудачу или отказывается от продолжения учебы. Почему?

Результаты Беннедсена и Касперсена сообщают об успехе или неудаче в учебе, однако критерии учителей CS1 не являются единственно возможным определением успеха. Многие программисты вообще никогда не ходили ни на какие курсы, но добились успеха. Таким образом, сначала необходимо удостовериться в том, что у людей действительно возникли трудности с изучением программирования, не привлекая к этому доказательства в виде оценок. Если нам удастся установить этот факт, можно задать вопрос «Почему?» Может, программирование противоречит человеческой природе? Не станет ли оно проще в другой форме? Можно ли учить программированию так, чтобы упростить обучение? А может, мы просто понятия не имеем, как измерить реальные познания студентов в программировании?

Действительно ли у студентов возникают трудности

В 1980-х годах в Йельском университете Эллиот Соловей (Elliot Soloway) регулярно давал на занятиях по программированию на Pascal следующее задание [Soloway et al. 1983]:

> *Напишите программу, которая в цикле читает положительные числа до тех пор, пока не прочитает целое число 99 999. После чтения 99 999 программа выводит среднее арифметическое прочитанных чисел.*

Эта задача, названная «дождевой задачей»[1], чаще других изучалась на ранних порах исследований в преподавании программирования. В статье от 1983 года, из которой взята эта формулировка задачи (другие формулировки исследовались в других статьях), группа из Йельского университета выясняла, улучшает ли возможность использования команды `leave` (`break` в C или Python) эффективность решения задачи. Задача была предложена трем группам студентов:

- первокурсникам CS1 после изучения и практического применения конструкций `WHILE`, `REPEAT` и `FOR` — 3/4 семестра;
- CS2 (второй семестр, обычно курс структур данных) — 3/4 семестра;
- студентам предпоследнего и последнего курса, специализация «системное программирование».

В каждой категории половина студентов использовала традиционный синтаксис Pascal, а другая имела возможность использовать Pascal с добавлением команды

[1] По аналогии с задачей вычисления среднего уровня осадков — *Примеч. перев.*

`leave`. Результаты, представленные в табл. 7.1, покажутся поразительными каждому, кто успешно начал карьеру в области программирования. Только 14% первокурсников смогли решить эту задачу в традиционном синтаксисе Pascal? А 30% *самых опытных* студентов задачу вообще не решили? Исследование повторялось несколько раз (например, работа студентов над «дождевой задачей» изучалась в диссертациях Джима Спорера [Spohrer, 1992] и Льюиса Джонсона [Johnson and Soloway, 1987]), а также многократно воспроизводилось неформально — и каждый раз с удивительно похожими результатами.

Таблица 7.1. Результаты решения «дождевой задачи» студентами Йельского университета

Группа	% правильных решений в традиционном синтаксисе Pascal	% правильных решений в синтаксисе Pascal с leave
CS1	14	24
CS2	36	61
Системное программирование	69	96

Задача требует относительно сложного условного управления циклом. Если программа читает отрицательное число, она игнорирует его, но продолжает принимать данные. Если число положительно и отлично от 99 999, оно прибавляется к накопленной сумме, а счетчик увеличивается на 1. Если число равно 99 999, то оно игнорируется, а выполнение цикла прерывается. Ошибка в реализации логики легко приводит к тому, что к накопленной сумме будет прибавлено отрицательное число или 99 999.

Эти результаты были получены в Йельском университете. Может, в нем просто не умеют учить программированию? Лишь немногие студенты в наше время изучают программирование до поступления в высшее учебное заведение, так что полученные ими сведения были в основном получены с курсов CS1. Ученые уже много лет пытаются найти способ провести исследование, из которого был бы исключен усложняющий фактор возможности плохого обучения в конкретном учебном заведении.

Группа Маккракена

В 2001 году Майк Маккракен [McCracken et al. 2001] организовал встречу группы исследователей на конференции ITICSE (Innovation and Technology in Computer Science Education) в Кентерберийском университете (Кент). ITICSE — европейская конференция, на которую приезжают участники со всего мира. Учителя из группы Маккракена должны были провести одинаковое исследование в своих учебных группах CS1 и CS2: поставить одну и ту же задачу и дать студентам 90 минут на ее выполнение на бумаге. Все результаты анализировались участниками на конференции. В этом многоцентровом многонациональном исследовании участвовали четыре учебных учреждения из трех стран. Сравнивая студентов из разных стран

и учреждений, исследователи надеялись получить представление о том, чему же студенты реально учатся на начальных этапах.

Задача заключалась в вычислении результатов логических выражений (префиксных, инфиксных или постфиксных) с использованием только чисел, бинарных операторов (+, –, /, *) и унарного отрицания (~ для предотвращения сложностей, связанных с перегрузкой знака –). Всего ответы были получены от 216 студентов. Средний показатель по независимой от языка шкале до 110 баллов составил всего 22,89 (21%). Студенты справились с этой задачей просто ужасно. Один преподаватель даже «сжульничал», проведя лекцию по вычислению выражений до назначения задачи. Результаты этой группы были не лучше, чем у других.

Группа Маккракена проанализировала полученные данные. Было обнаружено, что эффективность решения задачи сильно изменялась между учебными группами. Также были обнаружены доказательства «эффекта двух подгрупп», который многие преподаватели заметили, а некоторые научные статьи попытались объяснить. Некоторые студенты просто «врубились» и отлично справились с задачей. Более многочисленная подгруппа показала куда худший результат. Почему одни студенты «врубаются» в программирование, а у других это не получается? Были исследованы разные переменные факторы — от прошлого опыта до математической подготовки [Bennedsen and Caspersen, 2005], — однако убедительного объяснения этого эффекта до сих пор не найдено.

Группа Листера

Возможно, некоторые студенты плохо реагируют на конкретного преподавателя или стиль преподавания, но почему так много студентов в нескольких учреждениях показали такие низкие результаты? Неужели плохо учат *повсюду*? Мы переоцениваем возможности своих студентов? Или оцениваем что-то не то? Рэймонд Листер в 2004 году организовал вторую рабочую группу ITICSE для изучения некоторых вопросов [Lister et al. 2004].

Группа Листера решила проверить, не слишком ли много ожидала группа Маккракена от студентов. Для разработки и реализации решения задачи требовался относительно высокий уровень мышления. Группа Листера решила сосредоточиться на низкоуровневых способностях чтения и отслеживания кода. Была разработана анкета с множественным выбором ответов, которая предлагала студентам выполнять такие задачи, как определение результатов по фрагменту кода или заполнение пропусков в фрагменте программы. Вопросы в основном относились к операциям с массивами. Исследователи попросили своих участников из разных стран мира опробовать ту же анкету на своих студентах и доставить результаты на конференцию ITICSE.

Результаты группы Листера были лучше, но и они разочаровывали. У 556 студентов средняя оценка была около 60%. Хотя из этих результатов можно было сделать вывод, что группа Маккракена переоценила возможности студентов, Листер и его группа ожидали куда более высоких результатов.

Из работ Маккракена и Листера исследователи узнали, что на первом курсе трудно понять, что студенты понимают в программировании. Бесспорно, они узнают намного меньше, чем нам кажется. Но одни студенты учатся, а большинство — нет. Что же такого сложного в программировании, что многие студенты не могут освоить его?

Естественное понимание

Лингвисты обычно соглашаются с тем, что люди «настроены» на язык. Наш мозг эволюционировал таким образом, чтобы освоение языка проходило быстро и эффективно. Но если говорить точнее, мы «настроены» на естественный язык. Программирование представляет собой манипуляции с искусственным языком, придуманным для конкретной, относительно неестественной цели: точно сообщить компьютеру (не человеку!), что ему нужно сделать. Возможно, программирование не является для нас естественной деятельностью, и лишь немногие люди способны к сложным умственным упражнениям, необходимым для достижения успеха в этом неестественном деле.

Как ответить на этот вопрос? Можно попытаться использовать метод, аналогичный модифицированным Листером методом Маккракена: выбрать меньшую часть задачи и сосредоточиться на ней. «Программировать» значит указать машине, что она должна сделать, на неестественном языке. А если мы попросим участников приказать другому человеку выполнить некоторую задачу на *естественном* языке? Как участники определят свои «программы»? Если убрать искусственный язык, станет ли программирование более «естественным» или «основанным на здравом смысле»?

Л. А. Миллер попросил участников своего исследования написать указания для задачи, которую должен выполнять кто-то другой [Miller, 1981]. Участники получали наборы данных (например, информацию о работниках, должностях и зарплате) и задачи вида:

> *Составьте список работников, удовлетворяющих хотя бы одному из следующих критериев.*
>
> *(1) Работник занимает должность техника и зарабатывает не менее 6 долларов в час.*
>
> *(2) Работник не состоит в браке и зарабатывает менее 6 долларов в час.*
>
> *Список должен быть упорядочен по имени.*

Миллер узнал много нового о том, что было трудно, а что было просто для участников. Прежде всего все участники эксперимента справились со своей задачей. Он не говорит, что 1/3 участников сдались или потерпели неудачу, как это постоянно происходит на курсах программирования. Похоже, проблема связана не со сложностями описания процесса.

Ключевое различие между решениями задач Миллера на естественном языке и задачами по программированию, изучавшимися предыдущими исследователями, относится к структуре решения. Участники эксперимента Миллера определяли не итерации, а операции. Например, они не говорили: «Возьмите каждую запись и проверьте фамилию. Если она начинается с буквы ‘A’...» Вместо этого они использовали формулировки вида: «Для всех фамилий, начинающихся с буквы ‘A’...» Миллера это удивило: никто не указывал условие завершения цикла. Некоторые участники говорили о проверяемых `IF`-подобных конструкциях, но никто и никогда не использовал `ELSE`. Уже одни этих результаты указывают на возможность определения языка программирования на *естественном* уровне более понятного для новичков.

Миллер провел отдельный эксперимент, в котором он выдал другим участникам инструкции из первого эксперимента с нечеткими циклами. Ни у кого не возникло проблем с их выполнением. Все отлично понимали, когда нужно остановиться при завершении обработки данных. Участники обрабатывали данные из имеющегося набора, а не увеличивали индекс.

Джон Пейн повторил это исследование в конце 1990-х и начале 2000-х [Pane et al. 2001]. Пейна интересовала возможность создания языка программирования, приближенного к естественному языку, которым люди описывают выполняемые процессы друг другу. Он повторил эксперимент Миллера с другой задачей и другим набором исходных данных. Его беспокоило то, что концепции «наборов данных» и «списков» в описании Миллера могли подсознательно влиять на действия участника. Вместо этого Пейн показывал участникам графику и ролики из видеоигр, а затем спрашивал, как бы они описали компьютеру выполнение соответствующих действий, например: «Напишите команду, которая описывает, как я (компьютер) должен перемещать Pac-Man в присутствии или отсутствии других объектов».

Пейн, как и Миллер, обнаружил, что участники не пытались явно определять итеративные действия. Он сделал следующий шаг и попытался охарактеризовать написанные ими инструкции в контексте парадигм программирования. Он обнаружил использование ограничений («Эта штука всегда ведет себя так»), событийных («Когда Pac-Man собирает все точки, он переходит на следующий уровень») и императивных конструкций. Никто и никогда не говорил об объектах. Участники говорили о характеристиках и поведении сущностей видеоигры, но не о группировке этих сущностей (например, в классах). Они никогда не говорили о поведении с точки зрения самих сущностей; все происходящее представлялось с точки зрения игрока или программиста.

На основании экспериментов Миллера и Пейна можно утверждать, что люди способны формулировать задачи для других людей, но современные языки программирования не позволяют описывать задачу так, как о ней думает сам программист. Если сделать языки программирования более естественными, станет ли программирование более доступным? Смогут ли люди решать сложные задачи с содержательными алгоритмами на более естественном языке? Подойдет ли более естественный язык для задач как новичков, так и профессионалов? А если не подойдет, придется ли новичкам в какой-то момент своей карьеры изучать язык профессионалов?

Группа исследователей, называющая себя CCG (Commonsense Computing Group), занялась изучением подобных вопросов. Студентам, не прошедшим даже начальный курс программирования, было предложено решить нетривиальные алгоритмические задачи (такие как сортировка или параллелизация процессов) на естественном языке, до изучения каких-либо программных конструкций. Участники эксперимента на удивление успешно справлялись с этими задачами.

В одном из исследований [Lewandowski et al. 2007] студентам было предложено создать модель театра с двумя продавцами билетов:

> *Предположим, билеты заказываются по телефону следующим образом: когда покупатель звонит и заказывает n билетов, продавец (1) находит n лучших свободных мест, (2) помечает эти n мест как зарезервированные и (3) обсуждает*

с клиентом условия покупки (запрашивает номер кредитной карты, отправляет билеты в окно выдачи и т. д.). В театре одновременно работают сразу несколько продавцов. Какие проблемы могут возникнуть, и как предотвратить эти проблемы?

Задача была предложена 66 участникам из 5 образовательных учреждений — и они решили ее с поразительным успехом! Как видно из табл. 7.2, почти все студенты распознали суть проблемы, а 71% предложил работоспособное решение. В большинстве предложенные решения были неэффективными (в них использовался централизованный арбитраж), так что студентам еще предстоит многому научиться. Тем не менее, сам факт решения задачи параллельной обработки наводит на мысль, что трудности с программированием у студентов возникают из-за неподходящего инструментария. Возможно, студенты в большей степени способны к компьютерному мышлению, чем нам кажется.

Таблица 7.2. Количество решений и проблем, идентифицированных студентами (n = 66), по данным [Lewandowski et al. 2007]

Результат	**Процент студентов**
Проблемы:	
1. Многократная продажа билетов	97
2. Другие	41
«Разумное» решение проблемы параллельного доступа	71

Совершенствование инструментария и визуальное программирование

Как улучшить существующие инструменты? Один из очевидных ответов — переход на визуальные технологии. Еще со времен создания Дэвидом Смитом языка программирования Pygmalion, основанного на пиктограммах [Smith 1975], существует теория, что визуальное мышление проще для студентов. Многие исследования показывают, что визуализация в целом упрощает изучение программирования [Naps et al. 2003], но действительно серьезных исследований было относительно немного.

Затем Томас Грин и Мэриан Петр провели параллельное сравнение языка программирования, основанного на традиционных потоках обработки данных, с текстовыми языками программирования [Green and Petre, 1992]. Они написали программы на двух визуальных языках, хорошо сработавших в предыдущих исследованиях, и на текстовом языке, который тоже хорошо показал себя при тестировании. Исследователи ненадолго выдавали участникам визуальную или текстовую программу, а затем задавали вопросы о ней (например, о входных данных или результатах выполнения). На то, чтобы разобраться в графическом языке, участникам всегда требовалось больше времени. Причем результат не зависел от предыдущего опыта работы участника с визуальными или текстовыми языками или разновидностью

визуального языка. Визуальные языки воспринимались участниками эксперимента медленнее, чем текстовые.

Грин и Петр опубликовали несколько статей по разновидностям этого исследования — [Green et al. 1991], [Green and Petre, 1996], — но настоящей проверке это утверждение подверглось, когда Том Моэр со своими коллегами [Moher et al. 1993] попытались склонить чашу весов в пользу визуальных языков. Том со своими аспирантами использовал для обучения студентов программированию особую систему визуальных обозначений, так называемые *сети Петри*. Он раздобыл копию материалов Грина и Петра и создал версию, в которой из визуальных языков использовалась только модель сетей Петри. Затем Том повторил это исследование на себе и на своих студентах. И снова выяснилось, что текстовые языки в любых ситуациях проще воспринимаются участниками эксперимента.

Выходит, мы зря доверились своей интуиции по поводу визуальных языков? И на самом деле визуализация только затрудняет понимание программнного кода? А как же исследования группы Нэпса [Naps et al. 2003] — все они были ошибочными?

Для сравнения нескольких исследований применяется стандартный метод, называемый *метааналиэом*. Барбара Китченхэм описывает эту процедуру в главе 3. Крис Хундхаузен, Сара Дуглас и Джон Стаско провели такой анализ для исследований в области визуализации алгоритмов [Hundhausen et al. 2002]. Они обнаружили, что многие исследования, обладающие статистической значимостью, действительно демонстрировали преимущества визуализации алгоритмов для студентов. Также было много исследований со статистически незначимыми результатами. Некоторые исследования демонстрировали статистически значимые результаты, но из них было неясно, как алгоритмические визуализации способствовали усвоению материала (рис. 7.1). Хундхаузен и его коллеги обнаружили, что многое зависело от использования визуализаций. Например, использование визуализаций на лекциях мало влияло на обучение студентов. С другой стороны, самостоятельное построение визуализаций студентами оказывало значительное влияние на процесс их обучения.

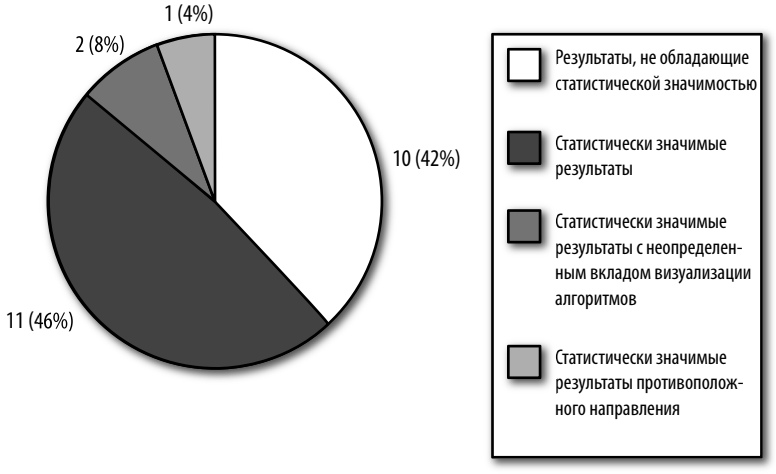

Рис. 7.1. Сводные данные по 24 исследованиям из статьи Хундхаузена, Дуглас и Стаско [Hundhausen et al. 2002]

Лишь немногие исследования изменяли применение визуализации, сохраняя другие переменные постоянными (тип обучения, тип студентов, преподаватель и т. д.). Хундхаузен с коллегами после анализа 24 исследований предполагают, что способ использования визуализации важен, однако предположить и доказать предположение — совсем не одно и то же. В ходе своих исследований в области преподавания мы выяснили, что предсказать результат всегда довольно сложно. Поведение людей куда менее предсказуемо, чем поведение летящего камня или химической смеси. Мы должны проверять свои подозрения и гипотезы, причем иногда делать это многократно в разных условиях, пока не убедимся в правильности своих предположений.

Роль контекстуализации

Итак, мы знаем, что новички узнают о проектировании и программировании намного меньше, чем можно ожидать, а процент неудач на первом курсе относительно велик. Отсутствие стабильного улучшения результатов при переходе с текстовых моделей на визуальные наводит на мысль о том, что в данной ситуации может действовать другая переменная. В предыдущем разделе рассматривалась возможность того, что такой переменной может быть способ использования визуализаций. Какие еще переменные можно изменить для улучшения результатов и успешности обучения студентов?

В 1999 году Технический университет штата Джорджия решил ввести обязательный вводный курс компьютерных технологий для всех студентов. За первые пять лет это требование было выполнено только одним курсом. В целом процент успешной сдачи экзаменов по этому курсу составлял 78%, что, в соответствии с данными Беннедсена и Касперсена, было неплохо [Bennedsen and Caspersen, 2007]. Но если как следует проанализировать это число, ситуация выглядит уже не столь привлекательной. Среди студентов общеобразовательных, архитектурных и управленческих колледжей процент успешной сдачи был ниже 50% [Tew et al. 2005a]. Среди женщин «провалы» встречались почти вдвое чаще, чем у мужчин. Если курс предназначен для всех студентов, а успешно проходят его в основном мужчины с технической специализацией, это говорит о системных проблемах в преподавании программирования.

В 2003 году был начат эксперимент по преподаванию новой разновидности вводного курса в контексте обработки цифровых материалов [Forte and Guzdial, 2004]. Студенты работали практически над теми же задачами, как и в других вводных курсах компьютерных технологий. Нам пришлось изрядно потрудиться, чтобы включить в курс все темы [Guzdial, 2003], рекомендованные действовавшим на то время стандартом ACM and IEEE Computing Standards [IEEE-CS/ACM 2001]. У нас это получилось, и во всем тексте учебников, примерах кода и домашних заданиях речь шла об обработке цифровых материалов. Например, задача перебора элементов массива рассматривалась на примере преобразования всех пикселов графического изображения в оттенки серого — а вместо конкатенации строк студенты выполняли конкатенацию звуковых буферов в операциях цифрового монтажа. Перебор поддиапазона массива изучался на примере удаления эффекта «красных глаз» без ущерба для красного цвета на фотографии.

Реакция студентов была положительной и очень заметной. Студентам новый курс показался намного более актуальным и интересным — особенно женщинам, у которых процент успешной сдачи превзошел аналогичный показатель среди мужчин (хотя и не в пределах статистической значимости) [Rich et al. 2004]. За следующие три года средний процент успешной сдачи экзаменов увеличился до 85%, причем даже в тех специализациях, в которых он ранее находился на уровне ниже 50% [Tew et al. 2005a].

Казалось бы, успех налицо, но о чем реально говорится в этих статьях? Можно ли утверждать, что все остальные условия остались неизменными, а изменился только новый метод? Возможно, в университетские колледжи вдруг стали поступать более умные студенты. Или технический университет штата Джорджия принял на работу нового обаятельного преподавателя, который увлек студентов своим энтузиазмом. Социологи называют такие факторы, которые мешают нам утверждать желаемое, *угрозами валидности*[1]. В защиту своей оценки изменений мы можем сказать, что в статье [Tew et al. 2005a] приводятся данные по нескольким семестрам с разными преподавателями, результаты обобщаются за три года исследований, а внезапное изменение уровня студентов кажется маловероятным.

Даже если мы с уверенностью заключим, что успех в Техническом университете штата Джорджия был обусловлен переходом на контекстную методологию, а все остальные факторы, относящиеся к студентам и преподаванию, остались неизменными, какие же выводы из всего этого следуют? Университет — хорошее учебное заведение, и туда поступают одаренные студенты. Он нанимает хороших преподавателей. Добьется ли ваше учебное заведение того же успеха во вводных курсах программирования, перейдя на контекстную методологию?

Первое испытание контекстной методологии в другом учебном заведении было проведено Чарльзом Фаулером в колледже Гейнсвиля (штат Джорджия), государственном колледже с двухгодичной формой обучения. Результаты представлены в той же статье [Tew et al. 2005a]. Фаулер также обнаружил кардинальный рост успеваемости среди своих студентов из разных специализаций, от компьютерных технологий до медицины. Впрочем, и в техническом университете штата Джорджия, и в колледже Гейнсвиля учились в основном белые студенты. Сработает ли этот метод для этнических меньшинств?

В Иллинойском университете в Чикаго (UIC) Пэт Трой и Боб Слоан ввели контекстную методологию на своих курсах CS 0.5 [Sloan and Troy, 2008]. Их занятия предназначались для студентов, которые хотели специализироваться на компьютерных технологиях, но не имели предварительной подготовки в области программирования. Вводный курс CS 0.5 должен был подготовить их к первому курсу (CS1). За несколько семестров процент успешной сдачи экзаменов среди этих студентов также возрос. UIC отличается более разнородным этническим составом, а большинство студентов принадлежит к этническим меньшинствам.

Вы еще не убеждены, что контекстную методологию стоит использовать с вашими студентами? Кто-то скажет, что эти случаи могли быть аномальными. Исследования в техническом университете штата Джорджия и в Гейнсвиле проводились со

[1] *http://www.creative-wisdom.com/teaching/WBI/threat.shtml*