Компьютерные инструменты в образовании, 2017
№ 5: 35–44
УДК: 378.147 + 004.41
http://ipo.spb.ru/journal

АВТОМАТИЗАЦИЯ ОБУЧЕНИЯ ЭФФЕКТИВНЫМ ПРОЦЕССАМ ПРОГРАММИРОВАНИЯ С ПОМОЩЬЮ ОБРАТНОЙ СВЯЗИ*

Самочадин А. В.[1], Тимофеев Д. А.[1]

[1]Санкт-Петербургский политехнический университет Петра Великого

Аннотация

В работе рассматривается проблема передачи знаний об эффективных процессах работы программистов при решении отдельных задач. В условиях массового, особенно дистанционного образования использование традиционной модели «учитель-ученик», предполагающей наблюдение за работой опытного наставника и общение с ним, оказывается невозможным. Для решения этой проблемы предлагается моделировать процессы, используемые опытными разработчиками, с помощью математического аппарата скрытых марковских моделей и связывать их с набором подсказок, позволяющих студенту улучшить собственный рабочий процесс. При использовании этого метода система идентифицирует решаемые студентом подзадачи и формирует обратную связь в виде подсказок, соответствующих текущему состоянию процесса.

Ключевые слова: *процесс разработки программного обеспечения, скрытые марковские модели, обратная связь, образовательные технологии.*

Цитирование: Самочадин А. В., Тимофеев Д. А. Автоматизация обучения эффективным процессам программирования с помощью обратной связи // Компьютерные инструменты в образовании. 2017. № 5. С. 35–44.

1. ВВЕДЕНИЕ

Разработка программного обеспечения вовлекает программиста в два вида деятельности. Первая из них, коллективная, связана с управлением проектом и его жизненным циклом. Она включает планирование, постановку задач, распределение их между участниками проекта и контроль выполнения задач исполнителями. Вторая, индивидуальная, состоит в решении назначенных разработчику задач — собственно программирование, проектирование, тестирование, разработка документации.

Вопрос управления проектом был предметом детального изучения с самого начала формирования программной инженерии как дисциплины. К настоящему времени выделяются 10–15 стандартных методологий, из которых на практике наибольшее распространение получили вариации «гибкого» («agile») подхода к разработке программного

*Работа подготовлена в ходе реализации проекта в рамках Постановления Правительства РФ № 218 от 09.04.2010 г. при финансовой поддержке Министерства образования и науки РФ (договор № 03.G25.31.0247 от 28 апреля 2017 г.).

обеспечения. Например, в 2016 году гибкими методологиями пользовались более 80 % компаний, сотрудники которых принимали участие в исследовании, проводимом компанией Atlassian [2]. Подобная унификация процессов существенно снижает для исполнителя сложность вхождения в новый проект. Подготовка специалистов также становится проще за счет стандартизации требований к квалификации выпускника.

Индивидуальные процессы, применяемые программистами при решении задач, исследованы в существенно меньшей степени. Основная часть выпускаемых книг в области программирования посвящена конкретным языкам и парадигмам программирования, отдельным технологиям, а также разработке алгоритмов или архитектуре программного обеспечения. В классических работах, посвященных технологии программирования ([6, 9, 10]), внимание уделяется в большей степени приемам кодирования, чем процессу работы над задачей. В то же время известна оценка, в соответствии с которой производительность труда лучших программистов может превышать производительность труда среднего разработчика в 10 раз и более. Авторы работы [3] связывают это отличие, в первую очередь, со способами мышления («паковщики» и «картостроители»), в то время как автор заметки [5] — с разницей в опыте сравниваемых программистов. В любом случае значительная разница в производительности труда свидетельствует о существовании различий между процессами разработки, которые применяют разные программисты. Эти отличия могут быть количественными, то есть состоять в основном в различных затратах времени на каждом этапе работы над задачей, или качественными. В последнем случае сами процессы, которые применяют более опытные разработчики, могут структурно отличаться от процессов, используемых менее опытными программистами.

Гипотеза о существовании качественных отличий между процессами, которые используют разные разработчики, была экспериментально подтверждена в диссертации Д. Каммы [7]. В описанном в ней эксперименте типовая задача разработки модульных тестов независимо решалась несколькими разработчиками одной компании. Все разработчики на основании мнения их руководителей были разделены на две группы (разработчики с высокой и средней эффективностью, причем оценка эффективности включала как производительность труда, так и качество разрабатываемого программного обеспечения). Действия каждого разработчика фиксировались с помощью видеозаписи. При решении модельной задачи производительность труда в этих двух группах отличалась в два раза в пользу «высокоэффективных» программистов. При этом выяснилось, что при решении задачи «высокоэффективные» разработчики использовали достаточно близкие друг к другу последовательности шагов, которые существенно отличались от процессов в группе программистов «средней эффективности».

Целью образовательного учреждения, подготавливающего специалистов в области информационных технологий и программирования, является формирование специалиста, способного эффективно работать в своей профессиональной области. Как следствие, представляется важным учить студентов не только общим методикам управления программными проектами в сочетании с алгоритмами и приемами кодирования, но и эффективным процессам работы над задачей. Проблема заключается в том, что такие эффективные процессы не формализованы, а их освоение требует личного опыта работы над задачами программного проекта. Традиционным методом передачи навыков такого рода является ученичество: ученик (менее опытный, начинающий специалист) наблюдает за действиями учителя (более опытного специалиста) и в определенной степени подражает им, усваивая, таким образом, не описываемую явно технологию работы. Наблюдение и подражание сочетаются с явным взаимодействием между учителем и уче-

ником, которое выражается, в том числе, в даваемых учителем подсказках и ответах на вопросы.

В условиях массового образования и, в особенности, при использовании дистанционных образовательных технологий традиционное ученичество оказывается невозможным. Из-за этого на практике реализуется подход, сводящийся к проверке результатов работы студента и выдаче рекомендаций на ее основе. Хотя преподаватель, особенно имеющий существенный опыт практической деятельности в области программирования, может на основе анализа результатов работы студента идентифицировать проблемы в организации процесса работы над задачей и дать свои рекомендации по улучшению этого процесса, эти рекомендации носят косвенный характер и существенно отделены во времени от собственно работы над задачей.

В данной работе предлагается новый подход к формированию у студентов навыка организации работы над программным проектом на уровне решения отдельных задач. Подход состоит в построении моделей процессов, реализуемых разработчиками высокой квалификации, моделировании применяемых студентами процессов, и формировании подсказок о возможном улучшении процесса непосредственно в ходе работы студента над задачей. В разделе 2 неформально описывается процесс работы программиста над индивидуальной задачей. Раздел 3 посвящен построению модели процесса разработки на основе анализа действий разработчиков. В разделе 4 описан подход к формированию обратной связи.

2. ПРОЦЕСС РАБОТЫ НАД ИНДИВИДУАЛЬНОЙ ЗАДАЧЕЙ

Рассмотрим в качестве примера деятельность программиста, решающего задачу реализации новой единицы функциональности. Работа над такой задачей может включать, в частности, следующие виды деятельности.

1. Планирование работы.
2. Изучение отдельных аспектов предметной области.
3. Изучение отдельных аспектов используемых языков и технологий программирования.
4. Проектирование, выбор алгоритмов и структур данных.
5. Обсуждение задач и решений, консультации с другими разработчиками, аналитиками, менеджерами, представителями заказчика.
6. Анализ существующего программного кода.
7. Написание программного кода, реализующего новую функциональность.
8. Модификация существующего программного кода.
9. Написание модульных тестов.
10. Модификация существующих модульных тестов.
11. Модульное тестирование программы.
12. Ручное тестирование отдельных функций приложения.
13. Отладка, поиск и исправление ошибок.
14. Профилирование кода, анализ его производительности.
15. Анализ и обсуждение результатов рецензирования кода другими разработчиками.
16. Документирование кода на уровне исходного текста (комментарии, встроенная документация).
17. Разработка внешней документации (описание архитектуры, схемы данных, алгоритмов).
18. Протоколирование выполненных работ, изменения состояния задач в системе управления задачами.

19. Анализ состояния задачи в системе управления задачами, сообщений об ошибках, комментариев.

Каждый конкретный разработчик с учетом задачи, своей роли в проекте и личных особенностей (знаний, опыта) может выполнять все эти виды деятельности или их подмножество. Работа над задачей обычно представляет собой итеративный процесс, в котором некоторые этапы многократно повторяются, а другие могут выполняться один раз или не встречаться вообще. При этом, как показано в работе [7], процессы решения задачи разными разработчиками могут отличаться не только набором и последовательностью выполняемых подзадач, но и подходом к решению каждой подзадачи, в том числе выбором инструментов и метода (алгоритма) решения задачи.

Процесс работы над задачей может прерываться по нескольким причинам.

- Естественные физиологические перерывы, усталость.
- Запланированные перерывы (встречи, совещания).
- Переключение на другие задачи.
- Внешние сигналы (взаимодействие с окружающими людьми, телефонные звонки, уведомления).
- Окончание рабочего дня.

Такого рода паузы являются неизбежной составляющей любого процесса разработки. В ряде случаев они необходимы, так как позволяют работнику отдохнуть и снижают уровень стресса. В фольклоре программистов распространены истории об ошибках, причины которых не удавалось обнаружить в течение длительного времени, но которые становились очевидными на следующей день или даже сразу же после ухода программиста с рабочего места. В то же время следует иметь в виду явление «потока» [1], при котором разработчик полностью погружается в процесс. Для состояния потока характерны высокая производительность труда и состояние удовлетворения от работы. Воздействие отвлекающих факторов часто приводит к выходу из состояния потока, при этом возврат в это состояние требует усилий и времени.

При подготовке студентов-программистов важно научить их использованию эффективных процессов разработки. Традиционно разработчики осваивают такого рода практики в ходе профессиональной деятельности, общаясь с более опытными коллегами и наблюдая за их работой. Хотя практический опыт работы над реальными проектами остается ключевым фактором профессионального становления разработчика, формирование навыков эффективной работы в процессе обучения может дать студентам конкурентное преимущество, а также сократить затраты компаний на обучение начинающих специалистов.

Перенос модели «наставник-ученик» из промышленной среды в учебный класс затруднен, в частности, следующими факторами.

1. В роли наставника должен выступать человек, профессионально занимающийся разработкой программного обеспечения, в то время как профессиональные преподаватели не всегда имеют необходимый и актуальный опыт промышленной разработки (верно и обратное: не все профессиональные разработчики владеют навыком преподавания, что может затруднять обучение начинающих разработчиков).
2. Обучение в промышленном окружении в большинстве случаев носит индивидуальный характер, при этом ученик и наставник взаимодействуют в течение всего рабочего дня. В условиях университета на одного преподавателя приходится большое число студентов, а взаимодействие осуществляется в течение ограниченного времени.

Помимо этого, студенты обычно одновременно работают над несколькими проектами, как учебными (домашними заданиями, курсовыми работами), так и личными. Все эти проекты целесообразно использовать для обучения студента лучшим практикам разработки программ, однако анализ используемых студентами в самостоятельной работе процессов и выдача рекомендаций далеко выходят за рамки возможного для преподавателей отдельных дисциплин.

На более низком уровне аналогичные проблемы возникают при написании исходного кода программы. Современные среды разработки (Microsoft Visual Studio, IntelliJ Idea и другие) производят синтаксический и семантический анализ текста программы непосредственно в ходе ее написания. При этом формируется два вида обратной связи с пользователем: предупреждения об ошибках (в частности, выделение некорректных конструкций) и рекомендации по улучшению кода. В последнем случае рекомендации, помимо способа автоматизации рефакторинга, имеют еще один аспект, более важный с точки зрения рассматриваемой в данной работе проблемы. Анализируя код, для которого была сформирована рекомендация, и предлагаемый способ его модификации, разработчик получает информацию, позволяющую ему улучшить свой уровень владения языком программирования и стиль кодирования. Рекомендацию в таком случае можно рассматривать как способ обратной связи, позволяющий передать разработчику знания о лучших практиках написания кода.

Для обучения студентов эффективным процессам программирования мы предлагаем использовать подобный подход. В его основе лежит сбор данных о действиях разработчика в процессе работы над задачей и его психофизиологическом состоянии.

1. На основе протоколов действий и психофизиологических показателях опытных программистов строится набор моделей (с учетом отличий в предметных областях, классах задач, языках и технологиях программирования), описывающих реализуемые этими программистами процессы работы над задачей.
2. Для основных состояний модели специалистами в области обучения и опытными разработчиками совместно формируется перечень подсказок, позволяющих улучшить результаты выполнения данного этапа работы и выбрать наиболее подходящие пути продолжения работы.
3. На основе построенных моделей строится модуль распознавания основных состояний процесса разработки на основе протокола действий пользователя и показателей его психофизиологического состояния. Этот модуль интегрируется со средами разработки.
4. Среда разработки с подключенным модулем используется студентами при разработке программ. В ходе работы над задачей она идентифицирует состояния процесса разработки и выдает пользователю подходящие с учетом его действий подсказки.

Для получения данных о действиях пользователя используются предоставляемые наиболее распространенными средами разработки возможности обработки событий с помощью загружаемых в среду модулей расширения, а также специализированные приложения, перехватывающие и протоколирущие такие системные события, как переключение фокуса ввода между окнами, запуск и завершение приложений. Так как многие из этих событий являются низкоуровневыми, целесообразно провести предварительную обработку и фильтрацию событий. Например, при отслеживании операции переключения между окнами может быть полезно разделить все переключения на несколько классов, чтобы отделить переключения между окнами одного приложения

от перехода к использованию другого приложения. Для оценки психофизиологического состояния, а также для обнаружения и классификации событий прерывания процесса используются носимые устройства (фитнес-браслет, «умные часы»), предоставляющие доступ к данным о частоте сердечных сокращений.

3. ПОСТРОЕНИЕ МОДЕЛИ ПРОЦЕССА

Процесс работы над задачей характеризуется следующими особенностями.

1. Различные реализации этого процесса отличаются друг от друга. Это относится и к различным задачам, над которыми работает один разработчик, и к одинаковым задачам, над которыми работают разные разработчики. Отличаться могут последовательности состояний, длительности пребывания в этих состояниях, а также конкретные действия, которые в каждом состоянии выполняет разработчик.
2. Состояния процесса невозможно наблюдать непосредственно. Протоколировать можно только конкретные действия, при этом одинаковые действия могут выполняться в различных состояниях. Для идентификации конкретной последовательности состояний необходимо участие разработчика, протокол действий которого используется для создания модели.

С учетом этих особенностей, для моделирования процессов индивидуальной работы над задачей представляется целесообразным использовать аппарат скрытых марковских моделей [4]. Дадим определение скрытой марковской модели с использованием обозначений, принятых в известной работе Л. Рабинера [11].

Скрытая марковская модель имеет N состояний $S = \{s_1, \ldots, s_N\}$. Конкретное состояние, в котором описываемая данной моделью система находится на шаге t, будем обозначать q_t. Для модели задано распределение вероятностей перехода между состояниями $A = \{a_{ij}\}$, $a_{ij} = P(q_{t+1} = s_j | q_t = s_i)$, которое задается в виде матрицы $N \times N$. Таким образом, следующее состояние зависит только от текущего состояния и не зависит от предыдущих состояний (марковское свойство). Состояние q_t неизвестно наблюдателю. Вместо этого ему доступно значение наблюдаемой на шаге t дискретной случайной величины o_t, принимающей значения из множества $V = \{v_1, \ldots, v_M\}$. Распределение этой величины в состоянии s_j задается распределением $b_j(k) = P(o_t = v_k | q_t = s_j)$. Набор таких распределений для всех состояний вектора $B = (b_1(k) \ldots b_N(k))^T$. Начальное состояние системы q_1 выбирается случайным образом, вероятности выбора каждого из состояний в качестве начального задаются вектором $\pi = (\pi_1, \ldots, \pi_N)$, $\sum_{i=1}^{N} \pi_i = 1$. Таким образом, скрытая марковская модель для фиксированных множеств состояний S и наблюдаемых значений V задается тройкой $\lambda = (A, B, \pi)$.

Задача построения скрытой марковской модели на основе набора наблюдений обычно ставится следующим образом. Пусть известны множества S и V, и дана последовательность наблюдаемых на каждом шаге величин $O = (o_1, o_2, \ldots, o_t, \ldots, o_T)$ для достаточно большого значения T. Необходимо найти значения параметров A, B и π, для которых вероятность порождения последовательности O была бы наибольшей. Для решения такой задачи используется алгоритм Баума-Велча [11].

Построение модели процесса индивидуальной работы над задачей выполняется совместно с разработчиками, чьи действия и являются наблюдаемыми величинами O. При этом сами разработчики знают, чем они занимаются в каждый конкретный момент, и могут явно предоставлять информацию о состоянии процесса. Например, в одной из ранних работ, посвященных анализу процесса разработки программ [8], участники экспе-

римента комментировали каждое свое действие и проводили часть рассуждений вслух. В условиях, когда действия пользователя протоколируются неявно, количество необходимых аннотаций может быть существенно сокращено: разработчику достаточно отмечать факт перехода в новое состояние процесса (переход к каждой новой подзадаче или смену вида деятельности). При этом параметры модели A и π становится возможным явно оценить путем подсчета количества переходов между каждой парой состояний. Матрицу B можно восстановить на основе анализа протокола действий каждого разработчика в известных состояниях, соотнеся действия по времени с каждым состоянием процесса.

Таким образом, для построения модели процесса разработки необходимо выполнить следующие действия.

1. Сформировать множество возможных состояний процесса разработки (видов деятельности, которые программист выполняет в ходе работы над задачей).
2. Сформировать множество возможных действий разработчика и обеспечить протоколирование соответствующих событий.
3. Провести серию экспериментов, в которых опытные разработчики работают над задачей в режиме протоколирования действий, комментируя текущее состояние процесса.
4. Провести анализ полученных комментариев, соотнести их с выделенными ранее состояниями процесса разработки, при необходимости скорректировать множество возможных состояний.
5. Оценить параметры скрытого марковского процесса $\lambda = (A, B, \pi)$ на основе данных, полученных на предыдущих шагах.

Предлагаемый подход является расширением метода Д. Каммы [7], в котором процесс разработки моделируется с помощью марковских цепей. Принципиальным отличием марковских цепей от скрытых марковских моделей является то, что непосредственно наблюдаемыми считаются сами состояния процесса. Хотя метод Д. Каммы является более простым, он не может быть применен к нашем случае. Несмотря на то, что в процессе формирования модели последовательность состояний известна благодаря наличию явных аннотаций, при использовании этой модели для обучения студентов состояния процесса разработки неизвестны, а единственным доступным набором данных является протокол действия каждого студента. Таким образом, знание распределения B, связывающего наблюдаемые действия со скрытыми состояниями процесса, оказывается ключевым для идентификации состояний процесса в ходе самостоятельной работы студента.

4. ФОРМИРОВАНИЕ ОБРАТНОЙ СВЯЗИ

После того как модель процесса индивидуальной работы над задачей построена, ее необходимо использовать для формирования обратной связи при самостоятельной работе студента. В качестве обратной связи предлагается использовать подсказки, предлагающие более эффективные подходы к решению конкретных подзадач и рекомендующие дальнейшие действия в ходе работы. Перечень подсказок разрабатывается в ходе построения модели совместно преподавателями и опытными разработчиками, чьи действия ложатся в основу эталонного процесса.

Подсказки должны формироваться при переходе студента к определенному состоянию процесса разработки. Как следствие, ключевой алгоритмической задачей на этапе обучения студента является идентификация последовательности состояний процесса

разработки. Исходными данными при этом являются события, связанные с выполняемыми студентом действиями.

Задача определения наиболее вероятной последовательности состояний процесса, которая могла породить конкретную цепочку значений наблюдаемой величины (декодирование по максимуму правдоподобия) является одной из трех стандартных задач, которые формулируются для скрытых марковских процессов. Оптимальное решение этой задачи может быть найдено с помощью алгоритма Витерби [13], основанного на методе динамического программирования. Ограничением этого алгоритма является необходимость полностью знать последовательность значений наблюдаемой величины $O = (o_1, o_2, \ldots, o_T)$. С точки зрения описываемой в данной работе задачи это означает, что последовательность состояний, достигаемых в ходе работы студента, может быть определена только после завершения сеанса работы.

Тем не менее, существуют модификации алгоритма Витерби, в которых последовательность состояний определяется (с некоторой задержкой) по мере поступления данных (декодирование с ограничением длины). Обзор такого рода алгоритмов приводится, в частности, в работе [12]. В общем случае эти алгоритмы не позволяют получить оптимальное решение, однако ожидаемая вероятность ошибки декодирования может варьироваться за счет выбора ограничения длины.

Кроме этого, сценарий использования модели в ходе обучения позволяет получать дополнительную информацию о состоянии процесса разработки. Для получения дополнительной информации о состоянии процесса можно использовать еще один вид обратной связи — диалоговые окна, в которых пользователь должен выбрать один из вариантов ответа. На основе выбранного пользователем варианта можно более точно определить вид деятельности, которым занимается в данный момент пользователь (состояние процесса). Эти диалоговые окна также должны быть реализованы в виде подсказок, чтобы как можно меньше отвлекать пользователя от работы.

С учетом этих соображений процесс формирования обратной связи выглядит следующим образом.

- Пользователь запускает среду разработки с загруженным модулем распознавания основных состояний процесса разработки и дополнительные программы для протоколирования действий пользователя (они могут быть также запущены автоматически модулем распознавания).
- По мере работы пользователя модуль распознавания определяет наиболее вероятные состояния процесса разработки, соответствующие последовательности действий пользователя. Для этого предлагается использовать алгоритм декодирования с ограничением длины.
- Если для определенного состояния процесса определены подсказки, модуль выдает их пользователю.

Возможны различные модификации этого процесса. В частности, выбор подсказок может зависеть от дополнительных условий, как статических (операционной системы, набора используемых приложений, языка программирования), так и динамических (реакция пользователя на предыдущие подсказки). Действия пользователя могут также запоминаться, и после завершения сеанса работы на их основе может быть выполнено декодирование по максимуму правдоподобия для определения наиболее вероятной последовательности для всего сеанса работы, что может позволить уточнить модель работы конкретного пользователя и улучшить точность распознавания состояний в будущем.

5. ЗАКЛЮЧЕНИЕ

В данной работе предлагается метод обучения начинающих разработчиков процессам индивидуальной работы над задачами с помощью моделирования процессов, используемых опытными разработчиками, и применения обратной связи в виде подсказок в среде разработки. Для моделирования процессов используется математический аппарат скрытых марковских процессов. Выбор подсказок осуществляется с учетом текущего состояния процесса, которое идентифицируется путем анализа действий пользователя и определения на их основе наиболее вероятной цепочки состояний. Модель деятельности опытных разработчиков строится методом обучения с учителем на основе явных аннотаций. Набор возможных подсказок также определяется в процессе создания модели. Таким образом, сочетание модели процесса и набора подсказок позволяет, возможно, приблизиться к реализации модели «учитель-ученик», позволяя отразить как явно выражаемые в виде подсказок знания опытных разработчиков, так и неявные процедурные знания, реализуемые в виде эффективных процессов.

Предполагается, что использование данного метода позволит приблизить процессы, используемые начинающими разработчиками, к процессам, характерным для опытных разработчиков. При этом обучение происходит в ходе самостоятельной работы, что особенно важно в условиях массового профессионального образования, особенно дистанционного обучения, когда принципиально невозможно отследить порядок действий студента при выполнении задания и выдать необходимую обратную связь без применения средств автоматизации.

Список литературы

1. *Чиксентмихайи М.* Поток: Психология оптимального переживания / Пер. с англ. 7-е изд. М.: Смысл; Альпина нон-фикшн, 2017.
2. Atlassian 2016 Report — Software Development Trends and Benchmarks. 2016. Available at https://www.atlassian.com/whitepapers/software-trends-2016 (дата обращения: 25.10.2017).
3. *Carter A., Sangler C.* The Programmer's Stone. 1997. Available at http://www.programmersstone.com (дата обращения: 25.10.2017).
4. *Baum L. E., Petrie T.* Statistical Inference for Probabilistic Functions of Finite State Markov Chains, 1966 // The Annals of Mathematical Statistics. № 37 (6). P. 1554–1563.
5. *Guzdial M.* Is There a 10x Gap Between Best and Average Programmers? And How Did It Get There? ACM Blog, 2014. Available at https://cacm.acm.org/blogs/blog-cacm/180512-is-there-a-10x-gap-between-best-and-average-programmers-and-how-did-it-get-there/fulltext (дата обращения: 25.10.2017).
6. *Hunt A. Thomas D.* The Pragmatic Programmer: From Journeyman to Master. Addison Wesley Longman Inc, 2000.
7. *Kamma D.* Study of Task Processes for Improving Programmer Productivity. PhD thesis. Indraprastha Institute of Information Technology, Delhi, 2007. Available at https://repository.iiitd.edu.in/jspui/bitstream/handle/123456789/513/PhD1004.pdf (дата обращения: 25.10.2017).
8. *Jeffries R. et al.* The processes involved in designing software // Cognitive skills and their acquisition. 1981. P. 255–283
9. *McConnell S.* Code Complete. 2nd edition. Microsoft Press, 2004.
10. *Martin R. C. et al.* Clean Code: A Handbook of Agile Software Craftmanship. Pearson Education, 2009.
11. *Rabiner L. R.* A tutorial on hidden Markov models and selected applications in speech recognition. // In: Proceedings of the IEEE, 1989. Vol. 77, issue 2. P. 257–286.
12. *Šrámek R.* The on-line Viterbi algorithm. Master's Thesis, Dept. of Computer Science, Comenius University, Bratislava, 2007.
13. *Viterbi A. J.* Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. // IEEE Trans. Inform. Theory. April 1967. Vol. IT–13, P. 260–269.

Поступила в редакцию 26.09.2017, окончательный вариант — 26.10.2017.

Computer tools in education, 2017
№ 5: 35–44
http://ipo.spb.ru/journal

AUTOMATED TEACHING OF EFFICIENT PROGRAMMING PRACTICES USING FEEDBACK

Samochadin A. V.[1], Timofeev D. A.[1]

[1]Peter the Great St. Petersburg Polytechnic University

Abstract

In this paper, we discuss the question of transferring knowledge about the efficient processes software developers use when solving specific tasks. The traditional way of learning informal processes, the apprenticeship, requires watching the master at work and asking questions. This method is not possible when dealing with the mass education, especially the distant one.

We propose to model the process of programming practiced by experts using a hidden Markov model associated with a set of hints. A plugin to an integrated development environment uses the model to detect the student's activity and implements the feedback by proving relevant hints.

Keywords: *software development process, hidden Markov model, feedback, educational technology.*

Citation: A. V. Samochadin and D. A. Timofeev, "Automated teaching of efficient programming practices using feedback," *Computer tools in education*, no. 5, pp. 35-44, 2017 (in Russian).

Acknowledgements: *This research is financially supported by the Ministry of Education and Science of the Russian Federation (state contract 03.G25.31.0247 by 28.04.2017).*

Received 26.09.2017, the final version — 26.10.2017.

Alexander V. Samochadin, Professor, Higher School of Software Engineering
samochadin@gmail.com
Dmitrii A. Timofeev, Lead programmer, Mobile Device Management Systems Lab dtim@comitative.com

Самочадин Александр Викторович, кандидат технических наук, профессор, Высшая школа программной инженерии, samochadin@gmail.com

Тимофеев Дмитрий Андреевич, ведущий программист, лаборатория «Системы управления мобильными устройствами», dtim@comitative.com