КОГНИЦИИ И ДИСКУРСЫ

УДК 004.43

В.Е. Зюбин

ЧЕЛОВЕКО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Обсуждаются когнитивные процессы, протекающие при использовании языков программирования, и предлагаемая гипотеза об информационной сложности. Результат работы представлен в виде набора принципов, который может быть использован при оценке прагматики языков программирования на различных этапах разработки программ и при создании новых языков программирования.
Ключевые слова: *человеко-ориентированное программирование; гипотеза об информационной сложности; оценка прагматики.*

1. Введение

Вопросы о роли и месте психологии в программировании стали возникать много лет назад, при становлении программирования как профессиональной деятельности. Удовлетворительных ответов получить не удалось, по прошествии времени вопросы стали восприниматься как риторические, и в настоящее время большинство исследователей предпочитают даже не вспоминать о них.

Научное сообщество концентрирует свое внимание только на тех аспектах человеко-машинных систем, которые могут быть исследованы чисто математическими методами, а небольшое число работ по прагматике формальных языков и психологии программирования – это не более чем те самые исключения, которые придают означенному правилу законченную форму.

Но насколько справедливо рассматривать программирование только с точки зрения вычислительной математики, как простое манипулирование битами? Какой вред компьютеру может нанести программа, содержащая строки с GOTO, о которых писал Дейкстра [1]? Можем ли мы ограничить наши исследования только устройством компьютера, в то время как программирование немыслимо без участия человека? И наконец, сможем ли мы вообще изучать программирование, если мы исключили из рассмотрения значительную и неотъемлемую часть проблемы?

В основе настоящей статьи лежит твердое убеждение, что программирование – это человеческая деятельность, где люди создают программы исключительно для людей, используя для этой цели компьютер лишь как инструмент и на некоторых стадиях. А значит, программирование должно рассматривать не только компьютерные стороны проблемы, но и проводить всестороннее исследование человеко-машинных систем. Это особенно важно в современных компьютерных и информационных системах при рассмотрении вопросов системного анализа, формировании требований, разработке общей концепции кодирования (как использования языковых средств) и сопровождения. Практикуемый механистический подход – это верный путь к срыву проекта.

К сожалению, сломать сложившуюся практику весьма непросто. И дело тут в следующем.

Исторически программирование и кибернетика возникли как ветви математики, царицы точных наук. Студенты, математики и физики, специализирующиеся на информационных технологиях, даже не имеют курсов по основам психологии. С другой же стороны, психология рассматривается скорее как гуманитарная наука, которая слабо интересуется проблемами построения формализмов.

Последствия этой драматичной ситуации очень болезненны. Достаточно вспомнить проект АЛГОЛ. Несмотря на громадные интеллектуальные усилия научного сообщества, произошел коллапс проекта. Неимоверная сложность (психологическая сложность!) в изучении и сопровождении языка, обилие выразительных средств привели к повторению печальной истории Вавилонской башни.

Любая более или менее успешная попытка интегрировать достижения психологии в практику программирования если и не будет означать грандиозного прорыва, так хотя бы позволит сократить число досадных и болезненных провалов. Мы твердо убеждены, что программирование не имеет будущего без ассимиляции основ психологии в практику программирования и процесс обучения программированию.

В статье на основе обсуждения когнитивных процессов, структуры человеческой памяти и психологических ограничений на обработку информации формулируется гипотеза информационной сложности, базирующаяся на идее взаимодействия основных элементов человеческой памяти. Приводится общий подход к проблеме сложности и вскрываются ее психологические корни. Далее, с целью продемонстрировать пользу созданного понятийного аппарата, рассматривается несколько широко известных примеров из области программирования.

Изложенный подход образует компактный понятийный аппарат, чрезвычайно полезный при обсуждении различных проблем программирования, применимый на различных этапах жизненного цикла программы и при создании новых формальных языков. Создание программ на основе предложенного подхода было названо человеко-ориентированным программированием (Human-Oriented Programming).

2. Человек и программирование

Наиболее известная в программировании работа по психологии – это статья Дж. Миллера «Магическое число семь, плюс-минус два: некоторый предел наших возможностей обработки информации» [2]. Основная идея этой статьи полувековой давности отражена в названии: возможности человека обрабатывать информацию, увы, ограничены, и это ограничение специфицируется числом семь. У нас возникают серьезные ментальные проблемы, когда мы имеем дело со смысловыми объектами в количестве, превышающем пороговое значение.

Это «магическое» число достаточно часто встречается в компьютерной литературе, но цитирование носит сугубо механический характер. Авторы приводят точное число, в то время как оригинальное число, указанное в за-

коне Миллера, «магическое». Точнее, обсуждаемый порог специфицируется областью значений, поскольку сильно зависит от сопутствующих обстоятельств, которые мы сейчас обсудим.

Во-первых, Дж. Миллер сообщает о существенных отклонениях от своего закона. Число может превышать границы области, если подопытный хорошо знаком с областью, которой принадлежат смысловые элементы.

Во-вторых, эксперименты показали, что порог может быть преодолен, если смысловые элементы различаются по нескольким признакам (например, цвет, размер, масса). «Лучше знать немного о многом, чем много о немногом», – пишет сам Дж. Миллер, интерпретируя полученный результат.

Таким образом, с одной стороны, закон Миллера отражает объективную реальность, а с другой – наблюдаемые нарушения закона говорят о необходимости его коррекции, которая может быть выявлена, если проанализировать структуру памяти человека [3].

Память человека состоит из двух частей – кратковременной и долговременной памяти [4], см. таблицу их свойств. При этом следует сделать важную ремарку: эти части просто абстрактные понятия, отражающие функционирование, а не физическую структуру человеческого мозга.

Свойства кратковременной и долговременной памяти

Тип памяти	Трудоемкость загрузки	Время хранения информации	Семантическая емкость
Кратковременная	Незначительная	До 30 секунд	7 ± 2 сущностей
Долговременная	Существенная	Длительное	Неограниченная

Кратковременная память не требует усилий на загрузку, но ограничивает время операции и число операндов. Ограничение на время выполнения операции означает, что длительные операции вызывают усталость, которая ведет к необходимости разгружать память. Эта проблема также известна как *проблема закрытия* [3]. Оба ограничения напрямую связаны с понятием *информационной сложности*, которая может интерпретироваться как степень загрузки кратковременной памяти.

С другой стороны, человек имеет долговременную память с единственным ограничением – она предполагает существенные усилия при загрузке. Эти усилия связаны с деятельностью кратковременной памяти.

Таким образом, кратковременная и долговременная память используются только в тесной связке.

Обсуждаемые факты позволяют сделать целый ряд весьма интересных заключений.

3. Гипотеза об информационной сложности

Закон Миллера описывает лишь ограничения на работу с незнакомой информацией. Для знакомой области закон Миллера не работает. С этой точки зрения знания – это просто долговременная память, содержащая информацию по предмету, а сама загрузка данных в долговременную память – это процесс обучения.

Также мы можем обойти закон Миллера, если долговременная память загружена *похожей* информацией. Этот трюк нашел широкое применение в

языках программирования, которые используют искусственные метафоры [5], т.е. переносят понятия одной предметной области, хорошо известной пользователю, на другую. Кстати, слово метафора и означает не что иное, как перенос. В качестве примера можно вспомнить язык релейно-контактных схем – графический язык, который использует метафору реле для программирования алгоритмов управления. Несмотря на метафорические артефакты (паразитные аналогии в сознании пользователя) и другие недостатки метода, он весьма эффективен, поскольку существенным образом сокращает время и усилия, затрачиваемые на начальные стадии овладения языком.

Другие способы, которые мы повсеместно используем для преодоления закона Миллера, основываются на декомпозиции исходной проблемы на слабозависимые части [6]. Это может быть сделано путем:

- исключения несущественных деталей и последующего обобщения;
- выделения семантически обособленных сущностей;
- разложения проблемы по ракурсам;
- создания иерархических зависимостей.

Необходимо отметить, что эти приемы ведут к появлению новой информационной структуры, в которой любая мысленная операция удовлетворяет закону Миллера. Процесс создания таких информационных структур хорошо известен как построение моделей.

Отличие между оригинальной проблемой и ее моделью заключается в следующем. Модель содержит только элементы с ограниченным числом семантических связей, в то время как причин, которые бы обусловили подобные свойства оригинала, нет.

Также следует отметить, что построение модели порождает погрешность, несоответствие оригинала и модели, однако при этом достигается новое качество, называемое *информационной изоляцией*.

Если после исключения несущественных деталей нам все же не удается построить модель, совместимую с законом Миллера, мы вынуждены выделять и временно устранять из рассмотрения *условно несущественные детали*, т.е. строить *ракурсную модель*. Таким образом оригинал может иметь несколько моделей, отражающих лишь определенные аспекты оригинала и структурно несовместимых.

Вышесказанное ведет к следующим заключениям:

- Мысленные операции, предполагающие одновременную манипуляцию с большим числом семантических элементов, чрезвычайно трудны или даже невозможны для человека.
- Динамические операции со сложной информационной сущностью требуют построения такой модели, работа с которой не нарушает закон Миллера.
- Качество модели, удобство ее использования определяется характером взаимодействия кратковременной и долговременной памяти пользователя.

Этот набор утверждений и составляет гипотезу об информационной сложности (ГИС).

ГИС создает компактную, но чрезвычайно полезную концептуальную основу для рассуждений на тему удобства использования формальных спе-

цификаций и вообще вопросов прагматики в целом. Принципы, составляющие ГИС, применимы на самых разнообразных стадиях создания программного обеспечения, в частности при создании языков программирования. Программирование, основанное на предложенном подходе, получило название *человеко-ориентированное программирование* (*Human-Oriented Programming)*, или сокращенно ЧОП.

Хорошим примером использования ЧОП на практике может служить язык Рефлекс, диалект языка Си, предназначенный для программирования сложных управляющих алгоритмов [7].

4. Известные факты под новым углом зрения

Рассмотрение известных фактов и проблемных вопросов, их простое объяснение в рамках ГИС показывают чрезвычайную полезность созданной основы для рассуждений.

4.1. Графические и текстовые языки программирования

Какой же язык лучше? Появление этого вопроса в сетевом форуме практически гарантированно ведет к баталиям, так называемым «религиозным войнам» (*holy wars*). Графические языки часто награждают самыми лестными эпитетами: простые, интуитивно понятные, дружественные, привлекательные и т.п. Подходы WYSIWYG и WIMP стали неотъемлемым атрибутом современных пользовательских интерфейсов. Но все же находятся ярые противники манипуляторов типа «мышь», джойстиков, меню, окон и иконок, у которых на каждый эпитет о полезности графики заготовлено по десятку контраргументов. Да и эксперименты показывают, что утверждение о несомненном превосходстве графики весьма дискуссионно [8, 9]. В чем же дело?

Исследование вопроса в рамках ГИС обнаруживает следующие ключевые моменты. Графика имеет два очень привлекательных свойства: большое число возможных признаков и простая возможность использования искусственных метафор. Признаки графики – это цвет, насыщенность, размер, текстура, положение, ориентация. Хотя признаки имеют специализацию (например, цвет, форма, текстура подходят только для отражения качественных характеристик, а остальные признаки – только количественных), они обеспечивают компактное и непротиворечивое совмещение разнородной информации.

Искусственные метафоры снижают нагрузку на кратковременную память во время изучения. Побочный эффект графики – метафорические артефакты, погрешность, сложность определения формальной семантики языка и т.д. [5, 6].

Текстовые языки хотя и допускают использование дополнительных графических признаков [10], но все же слабо приспособлены для этого. Преимущества текстовой формы записи в том, что она обеспечивает единство внешнего и внутреннего (машинного) представления данных, простоту стандартизации кодирования, модифицируемость, быстрое создание трансляторов. Текстовые языки позволяют использовать произвольную метафору, обеспечивающую чрезвычайную лаконичность записи, однако предъявляют повышенные требования к профессионализму пользователя, или, другими словами, специализированную загрузку долговременной памяти.

В качестве способа примирить достоинства и недостатки графики и текста был предложен двухэтапный подход к разработке программ [6], когда на

первом этапе предпочтение отдается графике, а на втором – текстовому языку. Ключевое преимущество подхода очевидно. Графические средства снижают нагрузку на кратковременную память, когда программист занимается построением модели, изучает задачу. Это позволяет:

- упростить работу за счет искусственной метафоризации;
- установить единое видение проблемы внутри команды;
- организовать конструктивный диалог с пользователем.

Возникающая при этом погрешность вполне разумная цена за возможность комфортной работы. Ошибки имеют локальный характер и могут быть скорректированы на втором этапе при программировании на текстовом языке, к началу которого программист будет иметь и общий план решения, и, что не менее важно, вполне подготовленную долговременную память.

Существенная проблема подхода – обеспечение бесшовного перехода между двумя представлениями. *Проблема бесшовного перехода,* по-видимому, не имеет общего решения и должна рассматриваться для каждого конкретного случая отдельно, например так, как это сделано для языка Рефлекс [7].

4.2. Аспект-ориентированное программирование

В качестве исходной посылки авторы аспект-ориентированного программирования (АОП) [11] используют справедливое утверждение (полученное из практики), что универсальных языков нет и для каждой из сторон проблемы лучше всего использовать собственный язык. На основе этого заключения авторы предлагают стиль-АОП, направленный на то, чтобы разделить описание системы на независимые части (аспекты) и описывать эти части в отдельных модулях наиболее подходящими для этого средствами («natural shapes»). Описанные таким образом аспекты затем «сшиваются» на этапе трансляции в исполняемый код. Например, в одном модуле описывается функциональная часть, в другом отражаются вопросы устойчивого функционирования, в третьем – структура данных и т.п. Преследуемая цель – повысить качество создаваемой системы за счет снижения порога сложности.

АОП показало на практике неплохие результаты и этим подтвердило свою эффективность, но при этом возникают несколько вопросов, на которые авторы не могут дать ответ. Почему аспекты противоречат друг другу? Чем определена неравноправность аспектов? Можно ли выработать ясное понимание естественной формы «natural shapes», которую имеют различные аспекты?

Отдавая отчет в том, что АОП – это один из примеров интуитивного использования ЧОП, в котором построение модели производится через аспектную декомпозицию, проведем краткий анализ АОП на основании гипотезы информационной сложности.

Аспектная декомпозиция, построение аспектных моделей производится через различные метафоры (не обязательно искусственные). Это обстоятельство уже само по себе означает структурную несовместимость моделей, которая может быть разрешена только через иерархию аспектов. То есть через создание цепочки вложенных аспектов типа матрешки, когда один основной аспект определяет глобальную структуру алгоритма, а влияние зависимых

аспектов имеет локальный характер. Так называемые «естественные формы» – это просто аспектные модели, которые, с одной стороны, удовлетворяют закону Миллера (просты для восприятия), а с другой – не могут по отдельности отразить целевую проблему во всей полноте из-за отброшенных ранее условно несущественных деталей.

4.3. Вопросы и вызовы времени

Было бы интересно обсудить на базе гипотезы информационной сложности следующие темы:

- Является ли UML [12] языком или это просто набор языков, называемых диаграммами?
- Почему языки Си++ and Java сильно схожи с языком Си?
- Что заставляет корпорацию Майкрософт – общепризнанного лидера в области объектно-ориентированных технологий – по-прежнему использовать Си для написания операционных систем?
- Почему промышленность сделала свой выбор в пользу языков стандарта IEC 61131-3 [13] и отвергла многообразие альтернатив как практических, например Си++, так и большое число теоретических моделей?
- Можем ли мы найти реальное различие между так называемыми языками общего назначения и проблемно-ориентированными языками? Действительно ли объектно-ориентированные языки – это языки третьего поколения? Не случилось ли ошибки, а мы попросту не замечаем проблемной ориентированности объективно ориентированных языков в силу широкого распространения персональных компьютеров?
- Какие психологические проблемы решаются с помощью концепций полиморфизма, наследования, сокрытия информации и т.д.?

Полезность ГИС выражается не только в том, что она задает направление поиска и предоставляет необходимый инструментарий для рассуждений об известных проблемах. Появляется возможность ставить новые вопросы и получать ответы на более высоком уровне понимания.

При этом не стоит воспринимать ЧОП как очередную панацею. ЧОП – это в первую очередь инструмент для трезвой оценки человеческих ограничений и, скорее, обоснование естественного ограничения на сложность информационных систем. В результате использования инструментария ЧОП можно не только выработать эффективное решение, но и прийти к отрицательному заключению о перспективах проекта.

В этой связи уместно затронуть основной вопрос, который мы будем вынуждены решить в ближайшем будущем, – какой язык мы будем использовать для программирования многоядерных процессоров. Это очень серьезный, а возможно, и основной вызов времени. Отсутствие приемлемого языка означает, что мы будем отброшены назад в эпоху низкоуровневого программирования, когда структура алгоритма зависела от физической топологии вычислительной платформы. Более того, кризис может привести к стагнации или даже коллапсу компьютерной индустрии как таковой [14]. Преодоление кризиса возможно только через трезвое понимание законов программирования и выработку подхода, который учитывает в себе и человеческий аспект программирования, и физический параллелизм машинной архитектуры.

5. Заключение

Гипотеза об информационной сложности обеспечивает компактный и вместе с тем эффективный понятийный аппарат для оценки свойств языков программирования и создаваемых информационных систем. Анализ взаимодействия кратковременной и долговременной памяти человека и психологических ограничений при работе с этими типами памяти позволяет дать ответы на многие вопросы прагматики, исторически не подлежащие рассмотрению в рамках классических информационных технологий.

Автор считает, что идеи человеко-ориентированного программирования представляют интерес для комплексных проектов по созданию новых языков программирования. Понимание природы и использование прямых методов оценки информационной сложности могут оказать существенную помощь при создании различных типов компьютерных и информационных систем.

Автор ожидает появления исследовательских работ в рамках человеко-ориентированного программирования, которые обеспечат решение насущных вопросов, стоящих перед компьютерной индустрией, и создание эффективных методик программирования.

Литература

1. *Dijkstra E.W.* GOTO Statement Considered Harmful // Communication of the ACM. 1968. Vol. 11. № 3. P. 147–148.

2. *Miller G.A.* The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information // Psychological Review J. 1956. Vol. 63. № 2. P. 81–97.

3. *Shneiderman B.* Software Psychology: Human Factors in Computer and Information Systems. Cambridge: Winthrop Publishers, 1980.

4. *Ericsson K.A., Kintsch W.* Long-Term Working Memory // Psychological Review J. 1995. Vol. 102. № 2. P. 211–245.

5. *Авербух В.Л.* Метафоры визуализации // Программирование. 2001. № 5. С. 13–17.

6. *Зюбин В.Е.* Графика или текст: какой язык нужен программисту? // Открытые системы. 2004. № 1. С. 54–58.

7. *Зюбин В.Е.* «Си с процессами»: язык программирования логических контроллеров // Мехатроника. 2006. № 12. С. 31–35.

8. *Green T.R.G., Petre M.* When Visual Programs are Harder to Read than Textual Programs. Human-Computer Interaction: Tasks and Organisation // Proc. ECCE-6 (6th European Conference on Cognitive Ergonomics). Rome, 1992, http://citeseer.nj.nec.com/green92when.html

9. *Whitley K.N.* Visual Programming Languages and the Empirical Evidence For and Against // J. of Visual Languages and Computing. 1997. Vol. 8. № 1. P. 109–142.

10. *Куст Т.С.* Графическая организация текста электронных учебных пособий // Вестн. Том. гос. ун-та. 2009. № 324. С. 37–41.

11. *Kiczales G., Irwin J., Lamping J., Loingtier J.-M., Lopes C.V., and Maeda C.* Aspect-oriented programming // LNCS. Vol. 1241. Springer-Verlag. P. 220–242.

12. *Larman C.* Applying UML and Patterns: An Introduction on Object-Oriented Analysis and Design and Interactive Development / Prentice Hall, 2005.

13. IEC 61131-3. Programmable Controllers. Pt. 3: Programming Languages. 2nd ed. International Electrotechnic Commission, 1998.

14. *Зюбин В. Е.* Многоядерные процессоры и программирование // Открытые системы. 2005. № 7–8. С. 12–19.