Материалы IX Всероссийской конференции по проблемам науки и высшей школы
"ФУНДАМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ В ТЕХНИЧЕСКИХ УНИВЕРСИТЕТАХ".
СПб.: изд-во Политехнического университета. 2005, с. 44-52.

АВТОМАТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Шалыто А.А.
Санкт-Петербургский государственный университет
информационных технологий механики и оптики

Излагаются основные положения новой технологии программирования, названной автором "автоматно-ориентированное программирование". При использовании предложенной технологии графы переходов конечных автоматов применяются при спецификации, реализации, отладке и документировании поведения программ.

Что такое автоматно-ориентированное программирование?

В последние годы большое внимание уделяется разработке технологий программирования для встроенных систем и систем реального времени, к которым предъявляются высокие требования по качеству программного обеспечения. Одним из наиболее известных подходов в этом направлении является синхронное программирование [1].

Параллельно с развитием в Европе синхронного программирования, в России создается подход к разработке программного обеспечения, названный "автоматно-ориентированное программирование (автоматное программирование)" [2–4], который можно рассматривать в качестве разновидности синхронного программирования.

В настоящей работе описываются основные положения "автоматно-ориентированного программирования". Оно поддерживает проектирование, реализацию, отладку и документирование программ в части обеспечения корректности их поведения.

Если в традиционном программировании в последнее время все шире используется понятие "событие", то предлагаемый стиль программирования базируется на понятии "состояние". Добавляя к нему понятие "входное воздействие", которое может быть входной переменной или событием, вводится термин "автомат без выхода". Добавляя к последнему понятие "выходное воздействие", вводится термин "автомат" (конечный, детерминированный).

Поэтому область программирования, базирующаяся на понятии "автомат", в работе [4] была названа "автоматное программирование", а процесс создания таких программ — "автоматное проектирование программ".

Особенность рассматриваемого подхода состоит в том, что при его использовании автоматы задаются графами переходов. Для различения однотипных вершин вводится понятие "кодирование состояний". При выборе "многозначного кодирования" с помощью одной переменной можно различить состояния, число которых совпадает с числом возможных значений выбранной переменной. Это позволило ввести в программирование такие понятие, как "наблюдаемость" и "управляемость" программ, широко используемые в теории управления.

В рамках предлагаемого подхода программирование выполняется "через состояния", а не "через переменные" (флаги), что позволяет лучше понять и специфицировать задачу и ее составные части.

При этом необходимо отметить, что в автоматно-ориентированном программировании проектирование, реализация и отладка проводятся в терминах

автоматов. В рамках предлагаемого подхода от графа переходов к тексту программы предлагается переходить формально и изоморфно, что позволяет сохранять в тексте программы "визуальный образ" этого графа.

В работе [4] было предложено реализовывать автоматы на языках программирования высокого уровня с помощью оператора `switch` языка *C*, либо его аналогов в других языках программирования. Поэтому предложенная технология была также названа *Switch-технология*. Ниже будет показано, что для объектно-ориентированного программирования могут быть использованы и другие подходы к реализации автоматов.

В настоящее время эта технология разрабатывается в нескольких вариантах, различающихся как классом решаемых задач, так и типом вычислительных устройств, на которых осуществляется программирование.

Логическое управление

В 1996 г. Российский фонд фундаментальных исследований (РФФИ) в рамках издательского проекта № 96-01-14066 поддержал издание работы [4], в которой предлагаемая технология была изложена применительно к системам логического управления, в которых события отсутствуют, выходные воздействия являются двоичными переменными, а операционная система работает в режиме сканирования. Системы этого класса реализуются обычно на программируемых логических контроллерах, которые имеют относительно небольшой объем памяти, а их программирование выполняется на таких специфических языках, как, например, язык функциональных блоков [5]. В работе [4] предложены методы формального написания программ для таких языков при задании спецификации для разрабатываемого проекта системой взаимосвязанных графов переходов. Показаны преимущества использования языка графов переходов по сравнению с языком *Графсет*.

Программирование с явным выделением состояний

В дальнейшем автоматный подход был распространен на событийные системы, которые называются также "реактивными" или "событийными" [6]. В них указанные выше ограничения сняты. Как следует из названия этих систем, в них среди входных воздействий используются события, в качестве выходных воздействий применяются произвольные процедуры.

Для программирования событийных систем с применением автоматов был использован процедурный подход. Поэтому в работе [7] такое программирование было названо "программирование с явным выделением состояний".

При этом выходные воздействия "привязаны" к дугам, петлям или вершинам графов переходов (применяются смешанные автоматы — автоматы Мура-Мили). Это позволяет в компактном виде представлять последовательности действий, которые являются реакциями на соответствующие входные воздействия.

Особенность предлагаемого подхода к программированию этого класса систем состоит в том, что в них повышается централизация логики за счет переноса ее из обработчиков событий и формирования системы взаимосвязанных автоматов, которые вызываются из обработчиков [8]. Автоматы между собой могут взаимодействовать по вложенности, вызываемости и за счет обмена номерами состояний.

Последний вид взаимодействия рассматривался также в работе [9], в которой утверждается, что "указанное взаимодействие может оказаться мощным средством при проверке программ".

Система взаимосвязанных автоматов образует системонезависимую часть программы, а функции входных и выходных воздействий, обработчиков событий и другие компоненты программы зависят от используемой аппаратно-программной платформы (системы).

Другая важнейшая особенность описываемого подхода состоит в том, что при его применении автоматы используются триедино: при спецификации, при программировании (сохраняются в программном коде) и при отладке. Отладка автоматных программ может выполняться как в графическом режиме (при наличии соответствующих инструментальных средств), так и по протоколам. Отметим, что протоколирование выполняется автоматически по построенной программе и может использоваться для задач большой размерности при сложной логике программы.

При этом каждый построенный протокол может рассматриваться в качестве соответствующего сценария. Отметим, что для "больших" задач невозможно применение диаграмм последовательностей и диаграмм кооперации, входящих в состав языка *UML* [10], так как при использовании этого языка указанные диаграммы предлагается строить вручную на этапе проектирования, в то время как в автоматном программировании протоколы строятся автоматически при выполнении программы.

Протоколы позволяют наблюдать за ходом выполнения программы и демонстрируют тот факт, что автоматы являются не "картинками", а реально действующими сущностями.

Автоматный подход предлагается применять не только при создании системы управления, но и при моделировании объектов управления.

Этот подход был апробирован при разработке ряда систем управления ответственными объектами, в том числе судовыми дизель-генераторами [11]. Система была специфицирована более чем тридцатью взаимодействующими автоматами. Для описания модели дизеля также использовались автоматы. При проектировании на каждый автомат выпускалось четыре документа:

- словесное описание ("декларация о намерениях");
- схема связей (задает интерфейс автомата, в том числе, поясняя на русском языке символы входных и выходных воздействий);
- граф переходов (с символьными обозначениями входных и выходных воздействий);
- текст программного модуля, который формально и изоморфно реализует граф переходов (также без использования смысловых идентификаторов и комментариев).

Эти документы заменяют самодокументирующиеся программы, содержащие смысловые идентификаторы и комментарии, так как эти средства при сложной логике программы не обеспечивают должного понимания программ и их пригодности для дальнейшей модификации [12]. Эту проблему при сложной логике не решают и самодокументирующиеся графы переходов [10].

Опыт проектирования с применением автоматного подхода подтвердил целесообразность использования протоколов для проверки корректности взаимодействия большого количества автоматов и каждого из них в отдельности.

Объектно-ориентированное программирование с явным выделением состояний

Для решения широкого круга задач весьма эффективен подход, основанный на совместном использовании объектной и автоматной парадигм, который в работе

[13] был назван "объектно-ориентированное программирование с явным выделением состояний".

Особенности этого подхода состоят в следующем. Также как и в машине Тьюринга [14] явно выделены управляющие (автоматные) состояния объекта, число которых значительно меньше числа остальных состояний, например, "вычислительных".

Как и при использовании любого другого подхода, применение предлагаемого связано с множеством эвристик, возвратов назад, уточнений и параллельно выполняемых работ. Однако после завершения создания программы предлагаемый подход может быть сформулирован (по крайней мере, для ее документирования) как "идеальная" технология, фиксирующая принятые решения [15].

1. На основе анализа предметной области выделяются классы, и строится диаграмма классов.
2. Для каждого класса разрабатывается словесное описание, по крайней мере, в форме перечня решаемых задач.
3. Для каждого класса создается структурная схема, отражающая его интерфейс и структуру. При этом атрибуты и методы разделены на автоматные и остальные.
4. При наличии в классе нескольких автоматов строится схема их взаимодействия.
5. Для каждого автомата разрабатываются словесное описание, схема связей, граф переходов.
6. Каждый класс реализуется соответствующим модулем программы. Его структура должна быть изоморфна структуре класса, а методы, соответствующие автоматам, реализованы по шаблону, например, приведенному в работе [8].
7. Производится отладка полученной системы, например, путем построения протоколов выполнения, в которых функционирование объектов, содержащих автоматы, описывается в терминах состояний, переходов, событий, входных и выходных воздействий.
8. Выпускается проектная документация, составной частью которой является программная документация.

Описанный подход был использован при создании системы управления "танком" для игры *Robocode* [15]. В отличие от систем управления сотнями других "танков", на этот "танк" выпущена подробная проектная документация, содержащая, в частности, графы переходов и схемы связей автоматов, реализующих функциональность "танка". Детальные протоколы поведения "танка" позволяют проследить все течение боя. Метод построения протоколов, возможно, является основой для новой концепции построения "черных ящиков".

В описанной технологии автоматы применялись как методы классов. В рамках этой технологии могут быть использованы и другие подходы к объектной реализации автоматов, изложенные, например, в работах [16–18]. Автоматы могут выступать, в частности, как объекты-наследники определенного класса, реализующего базовую функциональность автоматов, обусловленную семантикой Switch-технологии.

Возможно также использование классов, реализующих понятия "состояние" или "группа состояний".

При проектировании автоматных программ может быть использован паттерн *State* или, например, его модификация *StateMachine* [19].

Особенности автоматной реализации параллельных процессов на основе механизма обмена сообщениями рассмотрены в работе [20]. Еще один подход к автоматной реализации параллельных процессов описан в работе [21].

Наличие качественной проектной документации резко упрощает осуществление рефакторинга программы (изменение ее структуры при сохранении функциональности). Последнее подтверждается рефакторингом упомянутой выше системы управления танком, выполненным с целью повышения "объектности" программы [22].

Создано инструментальное средство *UniMod* (http://unimod.sf.net), которое обеспечивает разработку и выполнение автоматно-ориентированных программ. Этот пакет позволяет использовать UML - нотацию при построении диаграмм в рамках Switch-технологии. При этом схемы связей, определяющие интерфейс автоматов, строятся в нотации диаграмм классов языка *UML*, а графы переходов — в UML-нотации диаграмм состояний. В состав пакета *UniMod* [23] входит встраиваемый модуль (*plug-in*) для платформы *Eclipse* (http://www.eclipse.org), позволяющий:

- создавать и редактировать *UML*-диаграммы классов и состояний, которые соответствуют схеме связей и графу переходов;
- проверять полноту и непротиворечивость условий на переходах, выделять на найденные ошибки и автоматически исправлять их;
- выполнять автоматическое завершение ввода условий на переходах;
- выполнять запуск и визуальную отладку моделей.

Процесс построения моделей с использованием предлагаемого средства состоит в следующем:

- на основе анализа предметной области разрабатывается концептуальная модель системы в виде диаграммы классов;
- в отличие от традиционных для объектно-ориентированного программирования подходов, из числа сущностей выделяются источники событий и объекты управления. Для их взаимодействия вводятся автоматы. Источники событий активны — они по собственной инициативе воздействуют на автомат. Объекты управления пассивны — они выполняют действия по команде от автомата. Объекты управления также формируют значения входных переменных для автомата. Автомат активируется источниками событий и на основании значений входных переменных и текущего состояния воздействует на объекты управления, переходя в новое состояние;
- используя нотацию диаграммы классов, строится схема связей автомата, задающая его интерфейс. На этой схеме слева отображаются источники событий, в центре — автомат, а справа — объекты управления. Источники событий с помощью *UML*-ассоциаций связываются с автоматом, события которому они поставляют. Автомат связывается с объектами, которым он управляет;
- каждый объект управления содержит два типа методов, реализующих входные переменные (`xj`) и выходные воздействия (`zk`);
- для автомата с помощью нотации UML-диаграммы состояний строится граф переходов типа *Мура-Мили*, в котором дуги могут быть помечены событием (`ei`), булевой формулой, состоящей из входных переменных, и выходными воздействиями, формируемыми на переходах. В вершинах могут указываться выходные воздействия и имена вложенных автоматов. Автомат имеет одно начальное и произвольное количество конечных состояний;
- состояния на графе переходов могут быть простыми и сложными. Если в состояние вложено другое состояние, то оно называется сложным. В противном случае состояние простое. Основной особенностью сложных

состояний является то, что наличие дуги, исходящей из такого состояния, заменяет однотипные дуги из каждого вложенного состояния;

- все сложные состояния неустойчивы, а все простые, за исключением начального — устойчивы. При наличии сложных состояний в автомате появление события может привести к выполнению более одного перехода. Это происходит в связи с тем, что сложное состояние является неустойчивым и автомат осуществляет переходы до тех пор, пока не достигнет первого из простых (устойчивых) состояний. Отметим, что если в графе переходов сложные состояния отсутствуют, то, как и в *SWITCH*-технологии, при каждом запуске автомата выполняется не более одного перехода;
- каждая входная переменная и каждое выходное воздействие являются методами соответствующего объекта управления, которые реализуются **вручную** на целевом языке программирования;
- использование символьных обозначений в графах переходов позволяет весьма компактно описывать сложное поведение проектируемых систем. Смысл таких символов задает схема связей. При наведении курсора на соответствующий символ на графе переходов во всплывающей подсказке отображается его текстовое описание;
- в общем случае схема связей может быть расширена введением в нее дополнительных автоматов, взаимодействующих как между собой за счет вложенности и посылки событий друг другу. Это позволяет отказаться от схемы взаимодействия автоматов, используемой в Switch-технологии;
- система может содержать несколько схем связей, если необходимо запускать автоматы параллельно.

Из изложенного следует, что запускаемыми моделями являются построенные указанным образом схемы связей и диаграммы переходов.

Указанные модели могут быть реализованы двумя способами:

- интерпретационным;
- компилятивным.

В первом случае по моделям строиться XML-описание, которое интерпретируется. Интерпретатор входит в состав пакета *UniMod* и в настоящее время реализован только для языка *Java*.

Во втором случае модель преобразуется в код на целевом языке программирования для используемой операционной системы. Код компилируется и запускается. Этот подход целесообразно применять для устройств с ограниченными ресурсами, таких, например, как мобильные телефоны.

Описанные метод и инструментальное средство были успешно использованы для разработки ряда проектов в компании *eVelopers* (http://www.evelopers.com).

Вычислительные алгоритмы

Автоматный подход используется в настоящее время и при реализации вычислительных алгоритмов [24–27].

Так, в частности, показано, что произвольный итеративный алгоритм может быть реализован конструкцией, эквивалентной циклу do–while, телом которого является оператор switch.

На основе автоматов предложен новый подход к построению визуализаторов алгоритмов, используемых на кафедре "Компьютерные технологии" СПбГУ ИТМО при обучении программированию и дискретной математике [28, 29]. Подход позволяет представить логику работы визуализаторов системой взаимосвязанных

конечных автоматов. Система состоит из пар автоматов, каждая из которых содержит "прямой" и "обратный" автоматы, обеспечивающие пошаговое выполнение алгоритма вперед и назад соответственно.

На сайте http://is.ifmo.ru введен раздел "Визуализаторы", в котором публикуются визуализаторы, выполняемые в рамках "Движения за открытую проектную документацию".

Движение за открытую проектную документацию

27 ноября 2002 г. на открытии полуфинальных соревнований командного чемпионата мира по программированию *ACM* (Association for Computing Machinery) в Северо-восточном Европейском регионе было объявлено об организации "Движения за открытую проектную документацию" [30]. В рамках этого движения на сайте http://is.ifmo.ru создан раздел "Проекты", в котором размещено более 60 проектов разработки программного обеспечения на основе автоматного подхода. Перечислим некоторые из них:

- автоматная реализация интерактивных сценариев образовательной анимации с использованием *Macromedia Flash*;
- совместное использование теории построения компиляторов и Switch-технологии;
- скелетная анимация;
- управление различными технологическими процессами и объектами (упрощенная модель цеха холодной прокатки, дизель-генератор, турникет, кодовый замок, светофор, кофеварка, телефон, банкомат, лифт, система безопасности банка и т. д.);
- игры (*Terrarium*, *Robocode*, *CodeRally*, *Lines*, *Bomber*, *Однорукий бандит*, *Завалинка* и т.д.);
- управление роботами *LEGO Mindstorms*;
- XML-формат для описания внешнего вида видеопроигрывателя (www.crystalplayer.com);
- примеры клиент-серверных приложений;
- построение пользовательских интерфейсов;
- реализация сетевого протокола *SMTP*.

"Коллекция" проектов пополняется, и будет пополняться в дальнейшем.

Заключение

Одна из целей настоящей работы состоит в том, чтобы показать, что автоматы в программировании служат не только "для распознавания цепочек символов" [31] и управления "стиральными машинами". Кроме того, в работе показано, что автоматы являются не просто одной из математических моделей дискретной математики, а могут применяться при реализации любых программ, обладающих сложным поведением.

Использование автоматов упрощает формализацию спецификации программы, определяющей ее поведение и играющей "ключевую роль в вопросе сдерживания программных ошибок" [32].

Предлагаемая технология должна ответить на вопрос, поставленный в [33]: "теорию конечных автоматов мы проходили, но, причем здесь программирование? "

В заключение отметим, что в работе [34] утверждается, что программирование "от состояний" является первым из известных стилей программирования, который, как указано в этой работе, в настоящее время возрождается и приводится ссылка на

книгу [4]. В работе [35] указанный стиль программирования переименован в "автоматное программирование" также со ссылкой на книгу [4] и сайт http://is.ifmo.ru.

Предлагаемый в настоящей работе подход является развитием классической теории автоматов [31] и подхода Д. Харела [6, 36], основанного на диаграммах *Statechart*. Основная особенность предлагаемого подхода состоит в том, что он базируется на понятии "состояние" и предназначен для описания и реализации сложного поведения управляющих "устройств" при использовании различных языков программирования и различных вычислительных устройств (программируемые логические контроллеры, микроконтроллеры и ЭВМ).

Автоматно-ориентированное программирование создается при поддержке Министерства образования и науки Российской Федерации (http://is.ifmo.ru/science/1/) и Российского фонда фундаментальных исследований (гранты № 02-07-90114 (http://is.ifmo.ru/science/2/) и № 05-07-90011).

Литература

1. *Benveniste A., Caspi P., Edwards S. et al.* The Synchronous Languages 12 Years Later // Proceedings of the IEEE. Vol. 91. 2003. № 1, p. 28-35.

2. *Шалыто А.А.* Алгоритмизация и программирование для систем логического управления и "реактивных" систем. Обзор //Автоматика и телемеханика. 2001. № 1, с. 3–39 http://is.ifmo.ru

3. *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000. 780 С.

4. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 С.

5. *Шалыто А.А.* Реализация алгоритмов логического управления программами на языке функциональных блоков //Промышленные АСУ и контроллеры. 2000. № 4, с. 45–50 http://is.ifmo.ru

6. *Harel D., Politi M.* Modeling Reactive Systems with Statecharts. NY: McGraw-Hill, 1998. 260 P.

7. *Шалыто А., Туккель Н.* Программирование с явным выделением состояний //Мир ПК. 2001. № 8, с. 116–121; № 9, с. 132-138. http://is.ifmo.ru

8. *Шалыто А.А., Туккель Н.И.* SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5, с. 45-62. http://is.ifmo.ru

9. *Дейкстра Э.* Взаимодействие последовательных процессов // Языки программирования. М.: Мир, 1972, с. 9-86.

10. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. М.: ДМК, 2000. 429 С.

11. *Шалыто А.А., Туккель Н.И.* Проектирование программного обеспечения системы управления дизель - генераторами на основе автоматного подхода // Системы управления и обработки информации. СПб.: ФГУП "НПО "Аврора", 2003, вып. 5, с. 66-82. http://is.ifmo.ru

12. *Безруков Н.* Повторный взгляд на "собор" и "базар" //BYTE/Россия. 2000. № 8, с. 60–78.

13. *Шалыто А.А., Туккель Н.И.* Объектно-ориентированное программирование с явным выделением состояний //Материалы международной научно-технической конференции "Искусственный интеллект – 2002". Т.1. Таганрог - Донецк: ТГРУ - ДИПИИ, 2002, с. 198-202.

14. *Шалыто А., Туккель Н.* От тьюрингова программирования к автоматному //Мир ПК. 2002. № 2, с. 144-149. http://is.ifmo.ru

15. *Шалыто А.А., Туккель Н.И.* Танки и автоматы //BYTE/Россия. 2003. № 2, с. 69-73. http://is.ifmo.ru

16. *Гуров В.С., Нарвский А.С., Шалыто А.А.* Автоматизация проектирования событийных объектно-ориентированных программ с явным выделением состояний //Труды X Всероссийской научно-методической конференции "Телематика–2003". Т.1. СПб.: СПбГИТМО (ТУ), 2003, с. 282-283. http://tm.ifmo.ru.

17. *Шопырин Д.Г., Шалыто А.А.* Применение класса "STATE" в объектно-ориентированном программировании с явным выделением состояний //Труды X Всероссийской научно-методической конференции "Телематика–2003". Т.1. СПб.: СПбГИТМО (ТУ), 2003, с. 284-285. http://tm.ifmo.ru.

18. *Корнеев Г.А., Шалыто А.А.* Реализация конечных автоматов с использованием объектно-ориентированного программирования //Труды X Всероссийской научно-методической конференции "Телематика–2003". Т.2. СПб.: СПбГИТМО (ТУ), 2003, с. 377–378. http://tm.ifmo.ru.

19. *Шамгунов Н.Н., Корнеев Г.А., Шалыто А.А.* State Machine — новый паттерн объектно-ориентированного проектирования // Информационно-управляющие системы. 2004. № 5, с. 13–25. http://is.ifmo.ru.

20. *Гуисов М.И., Кузнецов А.Б., Шалыто А.А.* Интеграция механизма обмена сообщениями в Switch-технологию // Проектная документация. http://is.ifmo.ru.

21. *Любченко В.* О бильярде с Microsoft Visual C++ 5.0 // Мир ПК 1998. № 1, http://www.osp.ru/pcworld/1998/01/202.htm.

22. *Кузнецов Д., Шалыто А.* Система управления танком для игры "Robocode" // Проектная документация. http://is.ifmo.ru.

23. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6, с.12-17.

24. *Корнеев Г.А., Шамгунов Н.Н., Шалыто А.А.* Обход деревьев на основе автоматного подхода // Компьютерные инструменты в образовании. 2004. № 3, с. 32-37. http://is.ifmo.ru.

25. *Шалыто А., Туккель Н., Шамгунов Н.* Задача о ходе коня //Мир ПК. 2003. № 1, с. 152-155. http://is.ifmo.ru

26. *Шалыто А.А., Туккель Н.И.* Преобразование итеративных алгоритмов в автоматные //Программирование. 2002. № 5, с. 12-26. http://is.ifmo.ru

27. *Туккель Н.И.,* Шалыто А.А., Шамгунов Н.Н. Реализация рекурсивных алгоритмов на основе автоматного подхода // "Телекоммуникации и информатизация образования", 2002, № 5, с.72-99.

28. *Корнеев Г.А., Казаков М.А., Шалыто А.А.* Построение логики работы визуализаторов алгоритмов на основе автоматного подхода //Труды X Всероссийской научно-методической конференции "Телематика-2003". Т.2. СПб.: СПбГИТМО (ТУ), 2003. http://tm.ifmo.ru.

29. *Казаков М.А., Шалыто А.А.* Использование автоматного программирования для реализации визуализаторов // Компьютерные инструменты в образовании. 2004. № 2, с. 19-33.

30. *Шалыто А.А.* Новая инициатива в программировании. Движение за открытую проектную документацию //Мир ПК. 2003. № 9, с. 52-56. http://is.ifmo.ru

31. *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002. 528 с.

32. *Аллен Э.* Типичные ошибки проектирования. СПб.: Питер, 2003. 224 с.

33. *Чижов А.* chizh@irk.ru.

34. *Непейвода Н.Н., Скопин И.Н.* Основания программирования. Ижевск: Научно-издат. центр "Регулярная и хаотическая динамика", 2003. 868 с. http://ulm.udsu.ru/~nnn/index.html

35. *Непейвода Н.Н.* Стили и методы программирования. М.: Интернет-Университет Информационных Технологий – ИНТУИТ.РУ, 2005. 320 с.

36. *Harel D.* Statecharts: A visual formalism for complex systems // Science of Computer Programming. 1987. vol. 8, pp. 231–274.