ОТ РЕМЕСЛА — К ИНЖЕНЕРИИ; ОТ МАСТЕРСКОЙ — К ЗАВОДУ ПРОГРАММНЫХ СИСТЕМ

И.Е. Ермаков,
ООО «Метасистемы», директор по науке и образованию,
Технологический институт ОрёлГТУ, преподаватель, зав. лаб.

I. Программная инженерия: желаемое и действительное

1. Термин «программная инженерия» впервые был употреблён в качестве названия конференции NATO Software Engineering, состоявшейся в 1968 г., в период первого из так называемых «кризисов программирования». Термин был введён в качестве «программного», то есть задающего некоторый желательный вектор развития отрасли создания ПО. Эту нагрузку он несёт и по сей день. Можно сказать так: программная инженерия — это фаза зрелости, которой должна достичь деятельность по созданию программного обеспечения.

Что отличает инженерную стадию развития любой отрасли от незрелой, кустарной? В первую очередь, предсказуемость результатов деятельности, то есть свойств продуктов, создаваемых этой отраслью. Такая предсказуемость обеспечивается применением систематических, научно обоснованных методов (которые обычно в той или иной степени используют математический аппарат, в виде подходящих формальных моделей и нотаций). Такие методы, как правило, подкреплены арсеналом инструментальных средств.

Применение методов и инструментов для инженера-конструктора — не самоцель. Очевидно, что проектирование и конструирование являются творческой деятельностью, опирающейся на опыт и интуицию автора. Однако результатом творчества инженера является реальная система, которая имеет фазу эксплуатации, и автор несёт ответственность за поведение системы на этой фазе. Систематические методы для инженера являются способом изучения свойств создаваемых конструкций, дают возможность гарантировать эти свойства, обеспечивать требуемые эксплуатационные качества изделия и, таким образом, отвечать за результат своего труда.

2. Особенности программной отрасли — нематериальность создаваемых систем и свобода программиста от большинства физических ограничений, характерных для традиционных отраслей, неоднократно отмечены и общеизвестны. Однако кажется, что этой особенности стали придавать слишком большое значение, а часто — просто использовать её для оправдания недобросовестности и кустарщины в разработке ПО.

На самом деле, имеют место два упущения в рассмотрении. *Во-первых*, при сравнении программной отрасли с традиционными сопоставляются неправильные этапы: разработка и производство. Очевидно, термин «производство ПО» вообще имеет мало смысла. Если же говорить об этапе разработки, то продукт труда инженера-конструктора — проект технической системы, является таким же нематериальным, как и продукт программиста, хотя и требует дальнейшей трудоёмкой фазы материального производства системы. В этом плане отрасль ИТ возможно рассматривать как инженерию с вырожденной фазой производства, то есть просто как частный случай. *Во-вторых,* во многих задачах программист разрабатывает только часть большой технической системы, которая является продуктом традиционной инженерии и материального производства. Заметим, что в настоящее время остро стоит противоречие между всё большим увеличением роли встроенного ПО и тем, что именно программисты постоянно оказываются слабейшим звеном в разработке технических систем (широко известны примеры из области автомобильной и бытовой техники).

В различных источниках (например, в статьях классика — Э. Дейкстры [1, 2]) справедливо отмечается, что сложность современных информационных систем превышает сложность, с которой сталкивается обычный инженер. Это связано с целой совокупностью факторов — с особенностями самой деятельности, огромным числом областей её приложения и стоящих задач ([3, 4]), а в области встроенного ПО — с тем, что усложнение поведения и функциональности технических систем сегодня ложится именно на программную часть. Однако в значительной степени эта сложность накоплена внутри самой отрасли, является искусственной, паразитной (в терминах Информатики-21 [5] — «огромный пузырь избыточной сложности»). Эта паразитная сложность подпитывается незрелостью методов и инструментов, отсутствием методик профессионального образования, наконец, просто невежеством (профессиональным, общенаучным и общекультурным) среднестатистического деятеля современного ИТ.

Кроме того, ситуация в области ИТ на протяжении последних 20 лет плотно связана с общеэкономической ситуацией. Ещё памятен «бум доткомов» и последовавший затем «кризис ИТ» конца 90-х гг., который позднее дошёл и до российского программирования. Область информационных технологий ввиду своей «нематериальности» оказалась удобной для спекулятивного вкладывания финансов и потому оказалась полигоном для таких экспериментов (с вовлечением большого числа новых трудовых ресурсов, которые затем оказались безработными). Видимо, проблемы ИТ-отрасли как самой верхней «надстройки» экономики могут рассматриваться в качестве модельных для некоторых проблем экономических систем в целом. Можно ожидать, что поворот ИТ лицом к задачам «реального сектора» экономики способен положительно повлиять на развитие программирования в направлении зрелой инженерии. Давно известно, что именно область встроенного ПО и ответственных управляющих систем является наиболее дисциплинированной, оснащённой систематическими методами, математическим аппаратом, качественным инструментарием.

3. Отрасль ВТ и ИТ имеет двойственную природу: она появилась как следствие одновременно и достижений счётной техники (с переходом на фундамент электронных технологий), и развития математических моделей (от проблем Гильберта и исследований понятия алгоритма Тьюрингом, Чёрчем и др. математиками — к широкому полю теоретической информатики). Сама по себе эта двойственность, опять-таки, не исключительна для ИТ («непостижимая эффективность математики» — выражение Е. Вигнера [6] — отмечена в большом числе приложений к практическим отраслям; так же, как и обратное определяющее влияние прикладных задач на развитие математики).

Однако качественной особенностью вычислительной техники является то, что входным сырьём и выходным продуктом её работы является математический, абстрактный объект — число. Человек систематизирует опыт реальности в абстрактных концепциях, уточняет их до вида строгих математических моделей, а далее имеет возможность «замкнуть» эти модели обратно на физическое устройство (ЭВМ), сделать их исполняемыми. То есть имеет место «обратное» моделирование математических абстракций физическими процессами; программирование же можно назвать «технически исполняемой

математикой».

Несмотря на такую принципиальная связь математики и программирования, существует острое разделение между теоретическими областями и практической деятельностью: между Computing Science (CS) и Software Engineering (SE) (массовое же программирование вообще далеко и от того, и от другого). Российские компании часто жалуются на плохую подготовленность выпускников ВУЗов к работе в индустрии, однако, судя по некоторым публикациям (например, недавняя статья Б. Страуструпа [7, 8]), в американских университетах такой разрыв с реальной инженерией не менее серьёзен, а взаимная изоляция CS и SE часто культивируется сознательно.

Однако и эта проблема скорее в головах, чем в природе отрасли. Полезно вспомнить, что А. Тьюринг был не только основоположником теоретической информатики, но и руководителем разработки нескольких ЭВМ, В.М. Глушков создал алгебраическую теорию цифровых автоматов в качестве обобщения конструкторского опыта, а теория информации вообще имеет первоначальное название от своего основателя К. Шеннона — «математическая теория связи», которое наиболее точно отражает суть предмета (заметим, что попытки расширить эту прикладную дисциплину до рассмотрения некоей «информации вообще» являются примером наиболее зыбких увлечений теоретической информатики).

Отмеченные разногласия в профессиональных кругах ни в коей мере не могут служить оправданием необразованности массовых разработчиков, не желающих применять прикладные математизированные методы, которые позволяют качественно увеличить глубину мышления программиста и дать ему реальный контроль над свойствами создаваемого программного обеспечения. Показательны незатихающие споры относительно *азбуки*: необходимости грамотного построения алгоритмов на основе применения утверждений (предусловий, постусловий, инвариантов циклов — метод Дейкстры, который предполагает умеренную формализацию, однозначно понижающую общую трудоёмкость разработки корректных блоков программ) [9, 10].

4. Отличия программирования от чистой и даже прикладной математики принципиальны — и об этом нельзя забывать. Как уже было отмечено в п.1, продукт инженера — эта система, которая будет иметь фазу эксплуатации. Таким образом, работа программиста — это не игра внутри знаковой системы и не построение статического текста-описания, а конструирование той реальной системы, которая выполняется на машинах, имеет динамическое поведение, взаимодействует с окружающей средой. Это поведение, вообще говоря, позволяет считать программную систему гораздо более материализованной, чем символический продукт, создаваемый математиком.

Создание и эксплуатация программной системы подразумевают постоянную её эволюцию. Возможность такую эволюцию обеспечивать — безболезненно, долговременно, эффективно — стала главным требованием к ПО, наряду с надёжностью. С таким понятием, как эволюция системы, тем более — в условиях коллективной разработки, математик сталкивается в лучшем случае эпизодически.

Всё это является причиной известного в профессиональных кругах мнения, что худшие программные инженеры получаются из математиков (в то время как лучшие — часто из физиков). Анализ и коррекция такой ситуации, безусловно, необходимы для тех образовательных учреждений, которые строят подготовку программистов-профессионалов на фундаменте математических факультетов и кафедр. Кроме объективных отличий программной инженерии от математики в чистой форме, играет роль неприученность математиков к хорошей организации формальных текстов — к глубокому структурированию, простоте и ясности выражения (а из-за хорошо развитых навыков «борьбы с искусственными сложностями» значение такой организации математиками, как и программистами такого склада, часто вообще не ощущается и не признаётся). Эти проблемы являются в то же время и внутренними для математики (отправляем читателя по этому поводу, например, к статьям В.А. Арнольда [11, 12] и С.П. Новикова [13]).

5. Работа с текстом или работа с конструкцией?

Для автора данной статьи стало открытием то, что большинство программистов воспринимают работу над программой исключительно как символическую, текстовую деятельность (о чем было написано в [10], в конце). Более того, в личной переписке на наши аргументы относительно преимущественной инженерной составляющей в программировании выдвигались однотипные возражения: «Конечно, инженерная составляющая присутствует. Но как можно не видеть, что труд программиста сродни литературному — выражение своих мыслей текстом».

Конечно, умение строить строгие и ясные описания является основополагающим для деятельности любого культурного человека, и в том числе программиста (что позволяет, в частности, проекту Информатика-21 [5] справедливо акцентировать внимание на взаимосвязях между изучением языков, литературы, математики и программирования в школьном образовании [14, 15]). Понятно, что любая нотация, применяемая в инженерии, является в некотором смысле формальным языком; это — очевидная, но поверхностная особенность конструкторской деятельности. Так же, как работа с чертежом — это только рутинная составляющая конструирования, так и написание программы как текста — только механическая составляющая программирования.

Одним из показателей зрелости программиста служит то, какой процент его внимания направлен не на сам текст, а «между строк» — ведь именно там можно увидеть главное: свойства программной системы и её поведения при выполнении. Для алгоритма эти свойства заключены в утверждениях относительно состояния между операторами, соотношениях между этими утверждениями — инвариантах; для архитектурного уровня ПО — в коммутации компонентов через их разъёмы (интерфейсы) и протоколах взаимодействия, и т. п. Вся эта информация оказывается первостепенным дополнением к тому, что явно выражено в тексте программы на языке программирования; роль её такая же, как роль расчёта прочности и динамических свойств для технических конструкций.

Между тем подход к работе у среднестатистического программиста в точности следующий: компьютер рассматривается как некий внешний субъект, которому надо «объяснить, что он должен делать», программист же должен построить свою коммуникацию с этим «субъектом» посредством написания текста программы. Методом проб и ошибок текст доводится до того уровня, когда «компьютер понял, как себя вести» [16]. Ничего общего с научно-технической деятельностью такой подход не имеет, а является одной из форм современного «магического мышления».

Здесь мы не удержимся от того, чтобы процитировать Дж. Вейценбаума [17]: *«Программист волен каждое очередное затруднение трактовать как некоторый специальный случай, для которого следует специально написать особую подпрограмму, и таким способом он включает этот случай в свою систему. Прибегая к подобному неограниченному наращиванию в своих системах эпициклов, программисты приобретают неисчерпаемый резерв объяснений, позволяющих*

им обходить даже самые серьезные трудности. Одержимый программист на большинство проявлений неудач отвечает дальнейшими программными ухищрениями и, таким образом, не дает им возможности сгруппироваться в его сознании вокруг соответствующих теорий. Эти три механизма, названные Поляни логическим кругом, авторасширением и подавлением образования ядер, составляют основу оборонительного оружия истого приверженца магических систем мышления и, в частности, одержимого программиста. Литература по психиатрии свидетельствует о том, что эта патология тесно связана с бредом всемогущества».

Понимание того, что программирование — далеко не просто символическая деятельность, а программная система — отнюдь не текст, а инженерная конструкция, столь важно, что на этом следует специально акцентировать внимание в самом начале обучения программированию. Не стоит забывать, что студент или школьник, впервые видящий показываемые ему программы, не может ещё увидеть за ними ничего, кроме последовательности литер, которую потом должен «понять» компьютер. При этом, если явно не форсировать понимание программы как конструкции, то оно не имеет шансов сформироваться у начинающего программиста вообще, разве что долгим окольным путём будут вырабатываться какие-то «приблизительные ощущения». Яркой частной проблемой здесь является приучение новичков к строгому оформлению текста («исходный текст — это ваш чертёж»), к выделению структуры отступами и т.п.

Вообще, полезно рассматривать плоскую текстовую форму программ просто как технически обусловленную (возможностями ранних терминалов и способов ввода) частность (вспомним ранние алгоритмические нотации и широкое распространение блок-схем в то время, когда терминалов вообще ещё не существовало). Кроме исторических причин, текстовое представление само по себе достаточно удобно и практично, но не является единственно возможным. В области инструментария программирования (в частности, CASE-технологий реального времени) широко распространено представление программы в виде чертежа на графическом языке, неплохо проработаны понятия графического синтаксиса и формальных графических грамматик [76, 77]. Даже для текстовой формы возможно не плоское, а так называемое семантическое редактирование, когда программа вводится и редактируется целыми структурными блоками. В конце концов, конструирование ПО — всего лишь более обобщённое и гибкое в сравнении с конструированием электроники, и никто не мешает применять аналогичные последнему чертежи и САПР. Интересным движением в этом направлении является когнитивная эргономика В.Д. Пароннджанова [18, 19].

6. Развитие программной инженерии интенсивно происходило в 60-е-80-е гг., когда и в западном, и отечественном программировании был накоплен опыт выполнения очень крупных и ответственных программных проектов, в том числе масштабного распределённого характера (военные, космические системы; управление транспортом, связь, банковское дело и т. п.). Приобретённый опыт доступен профессиональной общественности из многих книг, авторы которых — Ф. Брукс [20], Дж. Фокс [21], И. Соммервилл [22], Л. Басс [23] и др. К сожалению, отечественный опыт решения задач такого масштаба менее доступен для читателя; в последние годы по тематике программной инженерии опубликованы книги таких российских экспертов в области программной инженерии, как В.В. Липаев [24], А.Н. Терехов [25], Е.А. Жоголев [26].

Кроме всего, в 80-х гг. сложился консенсус относительно того, каким требованиям должен удовлетворять промышленный язык программирования, что воплотилось в принятии языков Ada и Modula-2 в качестве де-факто-стандартов для серьёзных проектов [27, 28, 29, 30, 31, 32, 33]. Уровень проработанности программных архитектур того периода можно видеть, например, из книги Р. Бара «Язык Ada в проектировании систем» [35]; из материалов по отечественным аппаратно-программным комплексам «Эльбрус» [35, 36, 37]**,** по проекту МАРС — «Модульные асинхронные развиваемые системы» [38, 39].

Начиная со второй половины 80-х гг. начинается широкое распространение персональных компьютеров и систем на их основе. Отрасль ИТ оказывается под ударом сразу нескольких факторов: 1) растущая массовость применения и стоящих задач; 2) резкий приток новых кадров, преимущественно самоучек; 3) ограниченные возможности настольного оборудования, не позволяющие применить в полной мере накопленный опыт программной инженерии и качественные инструменты (например, специалисты по языку Ada отмечают как главный фактор потери популярности этого языка отсутствие до середины 90-х гг. полноценных реализаций на ПК и игнорирование военно-промышленным сектором этой проблемы). Все эти факторы способствовали полному «одичанию» массового программирования, распространению методов и инструментов, не выдерживающих никакой критики с позиций грамотной инженерии (например, C-семейство языков). В странах бывшего СССР это оказалось особенно ярко выражено в силу того, что новые программисты в принципе не сталкивались с наработанными в предыдущие десятилетия технологиями (ср. с США, где огромное количество бизнес-задач и по сей день обеспечивается мейнфреймами и унаследованными системами на старых языках типа COBOL).

В этих условиях трудно переоценить влияние разработок тьюринговского лауреата Н. Вирта и его учеников — языков и операционных систем семейства Oberon [40, 41, 42]. Будучи созданными на базе ПК собственной разработки (Ceres), они уникальны тем, что одновременно оказались и промышленными (наследуя языку Modula-2), и настольными инструментами (на базе наследованной от Smalltalk концепции интегрированной операционной среды). Идеи и опыт Оберонов послужили основой для более качественных и приближённых к инженерным требованиям, нежели C/C++, mainstream-языков: Java, C#, наконец, Google Go [43, 44].

В целом, в последние годы в распространении в ИТ идей и целей программной инженерии заметен некоторый подъём, примером которого могут служить инициативы IEEE SWEBOK [45, 46] — документ «Свод знаний по программной инженерии», и SEMAT [47] — «Software Engineering Method and Theory» — интернет-площадка, координирующая усилия ведущих мировых специалистов в данной области.

В любом случае, преодоление «болезней роста» в нашей молодой отрасли связано с созданием качественного и устойчивого контура «наука — образование — производство — наука». И важнейшая роль в этом контуре ложится на программную инженерию, которая вовлечена в решение сложных и ответственных прикладных задач, значимых для общества. Не следует забывать, что «чистые исследования» — это прерогатива естественных наук, познающих мир; в области же искусственной деятельности человека логика исследований диктуется решаемыми практическими задачами, иначе деятельность рискует выродиться в бесплодные игры с виртуальными артефактами.

II. «Завод программных систем»?

1. Фабрика или лаборатория — два основных цвета, которые останутся от пёстрой мозаики отечественной ИТ-

индустрии, если исключить многочисленные «любительские мастерские», находящиеся на ремесленной стадии, проблемы которых рассмотрены в предыдущем разделе данной статьи.

Ряд крупных ИТ-компаний, выполняющих краткосрочные проекты аутсорсингового характера, представляют собой конвейер по изготовлению однообразного несложного ПО, преимущественно характера бизнес-приложений — информационных систем. Заметим, что под «несложным» мы понимаем отсутствие «сложности в глубину», что не исключает наличия количественной, «экстенсивной» сложности. Такие проекты ориентируются на типовые архитектурные шаблоны (тонкий HTTP-клиент, трёхслойная архитектура, центральная монолитная база данных), опираются на готовый сторонний инструментарий — каркасы (frameworks), несущее ПО (middleware), СУБД. Интенсивно используются CASE-технологии и средства проектного документирования. Хорошо организовано разделение труда и управленческий контроль. Хорошую методическую, технологическую и организационную оснащённость можно считать сильной стороной ИТ-фабрик.

Однако типовой характер решаемых задач и конвейерный стиль работы способствует сужению кругозора программистов. Ещё более этому способствует опора только на готовые технологии, привычка руководствоваться краткосрочными соображениями и приоритетами в ущерб стратегическим — в силу этого практически всегда предпочтение отдаётся использованию готовых сторонних компонентов, даже если очевидно, что это разрушает целостность архитектуры системы и провоцирует далеко идущие отрицательные последствия. В итоге, фабрики ПО и специалисты, задействованные в их работе, вносят значительную лепту в «надувание пузыря избыточной сложности» и в порождение конструктивно несовершенных систем, закрепление этого как нормы. Кроме того, из-за тех же краткосрочных приоритетов культивируется «умеренное» и даже «прохладное» отношение к эксплуатационным качествам продуктов.

Фактически, имеет место контур: компании-монополисты, производящие устаревшее, избыточно сложное, конструктивно дефектное системное и инструментальное ПО — фабрики типовых проектов — факультеты, готовящие программистов, которые пытаются из конъюнктурных соображений ориентироваться на первые две группы. Интересная беседа по поводу проблем программной индустрии в России состоялась в 2006 г. с С.И. Рыбиным [48].

Полная противоположность фабрикам ПО — инновационные коллективы-лаборатории, часто образованные на базе или под влиянием университетов. Такие коллективы готовы решать сложные наукоёмкие задачи, тяготеют к созданию собственных технологий и инструментов, обладают большой эрудицией в области Computing Science. Это позволяет им находить прорывные, интересные решения (на Западе такие компании охотно скупаются крупными корпорациями, подобными Microsoft или Google). Однако часто предлагаемые решения не имеют успеха. Кроме прочего, причиной тому иногда является склонность таких «венчурных» коллективов в своих идеях выполнять чрезмерное обобщение прежде накопления достаточной эмпирической базы. Полученные обобщения оказываются любопытными умозрительными построениями, но имеют малое пространство практической применимости.

Проблемой для малых коллективов, которую многие отмечают, является «увязание» на поддержке и развитии выполненных крупных заказов. Тому есть две причины. Во-первых, из-за неналаженности систематического инженерного процесса наблюдается эксклюзивность («custom-характер») внутреннего устройства и подходов к проектированию для каждой системы в каждом коллективе, что делает практически невозможной или экономически неприемлемой передачу сложных систем на сопровождение кому-то, кроме их создателей. Во-вторых, из-за такой же неналаженности систематического образования в области ИТ затруднено вхождение в серьёзные проекты новых людей «с улицы», для организации же специальной подготовки кадров на базе университетов у большинства малых коллективов нет ни времени, ни ресурсов, ни, наконец, методического опыта преподавания.

Вызывает огорчение, что некоторые из идейно интересных проектов категории онлайн-систем (социальные сети, игры и т.п.) производятся откровенно халтурными методами, на основе устаревших и гнилых инструментов (что можно видеть из рассказов их авторов о внутреннем устройстве своих систем). Такой стиль «свободного творчества» распространяется среди новичков. Это, во-первых, закрывает многим проектам начинающих коллективов возможность к долгосрочному устойчивому развитию (система неизбежно и быстро «обваливается»). Во-вторых, это препятствует переносу накопленного опыта (опыта внешней концепции, продвижения и развития) в область социально значимых проектов, поскольку терпимый для развлекательных проектов уровень качества оказывается совершенно неприемлем в области других задач.

2. Заводом программных систем мы хотели бы назвать такую форму организации разработки программных систем, при которой решены проблемы, подробно раскрытые выше в этой статье, то есть налажены и обеспечены следующие составляющие:

- применение систематических инженерных методов (в том числе формальных, в той мере, в какой это необходимо);
- предсказуемость свойств создаваемых продуктов и возможность инженеров доказательно отвечать за них;
- обеспечение длительного цикла эволюции систем;
- способность как решать сложные наукоёмкие задачи, значимые для общества (в частности, для реального сектора экономики), так и выпускать на основе этих решений серийные продукты для задач исследованного класса;
- наличие устоявшейся схемы передачи проектов от исследовательских и конструкторских групп к «серийно-производственным», то есть интеграция в одной организации описанных выше форм лаборатории и фабрики ПО;
- система устойчивого пополнения организации новыми, должным образом подготовленными кадрами и быстрого включения их в работу;
- способность выстраивать линейки продуктов, накапливать фонд базовых средств [23], самостоятельно обеспечивать себя необходимым инструментарием.

Такой уровень организации дела встречается достаточно редко. Характерно, что на него чаще других поднимаются коллективы, изначально связанные со встроенным ПО критического назначения — как уже отмечалось выше, именно в этой области накоплен большой опыт программной инженерии. Хорошим примером может служить петербургская группа компаний «Ланит-Терком» [49], созданная проф. СПбГУ, зав. каф. системного программирования А.Н. Тереховым (автором книги [25]). В Санкт-Петербурге создана уникальная система подготовки кадров на базе СПбГУ [50, 51] и ИТМО [52]. Также следует отметить вклад проф. ИТМО А.А. Шалыто в развитие систематических методов — SWITCH-технологию, и выдвинутую им концепцию «За открытую проектную документацию» [53]. Сотрудниками этих коллективов выпущен ряд книг по программированию и программной инженерии (например, [54]).

Конечно, для развития любой организации важно правильное и устойчивое целеполагание. Как известно из

менеджмента, определяющим фактором для долгого успешного развития коллектива является наличие собственной системы интересов, внутренней инициативы, а не следование за конъюнктурой. Основой для такой инициативы могут стать линейки продуктов и инвестирование в собственную инфраструктуру. Процитируем авторов [23] (выделение наше): *«... в качестве долгосрочного проекта в CelsiusTech рассматривается не отдельная судовая система, изготовленная по специальному заказу, и даже не совокупность размещённых к текущему моменту систем.* ***Своей основной задачей компания видит ведение линейки продуктов как таковой****. Ведение линейки продуктов подразумевает сопровождение фонда базовых средств, обеспечивающее возможность регенерации любого существующего члена линейки (в конце концов, по мере изменения требований они обнаруживают тенденцию к развитию и росту) и построения новых членов. В определённом смысле,* ***развитие линейки продуктов означает поддержание способности к производству на основе базовых средств новых продуктов****. Для достижения этой цели нужно сделать так, чтобы повторно используемые модули всегда были современными и универсальными. Ни один продукт не должен развиваться отдельно от линейки продуктов. ... С точки зрения внешнего наблюдателя, CelsiusTech поставляет судовые системы.* ***Сами же сотрудники компании трактуют свою деятельность как развитие и наращивание фонда общих средств, который, в свою очередь, обеспечивает возможность изготовления судовых систем.*** *Такое различие в осмыслении деятельности компании (а это именно осмысление), хоть и едва уловимо, всё же проявляет себя в политиках управления конфигурациями, организации предприятия и методиках продвижения новых продуктов»*.

Мы склонны считать, что для выхода на уровень завода программных систем необходимо концентрировать усилия на трёх приоритетных направлениях: 1) развитие собственного инструментального и системного ПО — создание в некотором смысле «натурального хозяйства», позволяющего экранировать калейдоскоп сменяющихся ИТ-технологий и рыночной конъюнктуры; 2) построение целостной системы обучения системных программистов, с привитием им инженерной и естественнонаучной культуры решения задач; 3) решение реальных задач, из которых для нас представляют наибольший интерес связанные с реальным сектором и управляющими системами, а также масштабные распределённые информационные системы.

На основе такого видения идёт работа над организацией завода программных систем в Орловской области, на базе ООО «Метасистемы» [55], Технологического института ОрёлГТУ [56] (Факультет среднего профессионального образования [57, 58] рассматривается как узловое звено в подготовке кадров) и физико-математического факультета ОГУ [59].

3. Технологии программной инженерии.

По точному замечанию Э. Дейкстры [3], программист занимается строительством моста через пропасть, по одну сторону которой находятся машины (и простейшие примитивы, начиная с 0 и 1), а по другую — бесконечное множество прикладных задач. Построение этого моста возможно только путём поэтапной аккуратной абстракции, которая ведётся от обеих сторон пропасти. Задача абстрагируется посредством концептуальных, математических, информационных, алгоритмических моделей, которые затем замыкаются разработчиком на некоторую техническую платформу, на которой будет возводиться и функционировать программная система. Техническая платформа давным-давно представляет собой не просто оборудование, а многослойную структуру из виртуальных машин (на этом хорошо акцентируется внимание у авторов [60]). Инженер-программист обязан уверенно ориентироваться в таких многослойных структурах и столь же уверенно выполнять абстрагирование, создавая новые необходимые слои.

Необходимо понимать, что абстрагирование — не просто удобство или прихоть, но необходимость, без которой задачу может быть в принципе невозможно решить, о чём в математической логике существует теорема Оревкова [61], говорящая, что при попытке построить формальное доказательство без введения специальных понятий высокого уровня длина доказательства может вырастать в башню экспонент раз (и напротив, высокоуровневый подход может столь же эффективно сокращать сложность формальных систем; это касается, например, доказательно построенных алгоритмов, в сравнении с написанными «как попало» [10]). Также известен принцип нетранзитивности научного объяснения [63], который заключается в том, что если «на объяснении А основано объяснение В» и «на объяснении В основано объяснение С», то совершенно необязательно С может быть построено только на основе А без среднего уровня В.

Любая технология, по сути, представляет собой некоторый способ получения требуемых результатов из доступных исходных объектов. Она включает в себя методы работы и инструменты, их поддерживающие. При использовании любой технологии нужно чётко понимать её сообразность задаче и учитывать её качество: соответствие структурам реальных задач, а не умозрительным фантазиям программистов, концептуальную стройность, отсутствие избыточной сложности, дефектов дизайна и следований массовым предрассудкам.

Один из важнейших интеллектуальных навыков для инженера-программиста — умение находить подобие во внешне непохожих конструкциях и свободно выполнять эквивалентные преобразования между разными базисами. Ярким примером является эквивалентность по выразительной мощности архитектурных моделей на основе объектов-методов и процессов-сообщений (см., например, модель рандеву языка Ada [34] или модель ОС Plan-9). Вот что по этому поводу пишет логик, программист, педагог Н.Н. Непейвода [62]: *«Изоморфизмы между различными структурами — одно из наиболее ценных знаний, которое можно получить из математики. Ведь если две структуры изоморфны, на практике это означает, что у нас есть как минимум два внешне совершенно различных представления данных для решения одних и тех же задач. А выбор правильного представления данных — больше чем полпути к хорошему решению практической задачи. ... Если вы не можете избавиться от терминов, скорее всего, вы неглубоко понимаете вопрос. Один из критериев глубокого понимания сути дела — умение выразить её совершенно разными словами»*.

4. Технологическая инфраструктура для завода программных систем.

Не претендуя на полноту охвата, перечислим слои инфраструктуры, на которую может опираться разработка программных систем. Мы выделяем эти слои на основе своего опыта, который связан с системным ПО и распределёнными программными комплексами.

1) **Инструменты программирования** — языки программирования, окружения разработки и выполнения, библиотеки и программные каркасы (frameworks). Отметим, что большинство представлений о требованиях к этим инструментам полностью сложились ещё в 80-х гг. и «открывать Америку» тут следует в основном путём отказа от наслоенного мусора и выделения аккуратных базовых ядер.

2) **Внутрипрограммные конструктивные схемы** («паттерны объектно-ориентированного проектирования») —

наработанные на решении типовых проблем схемы соединения нескольких компонентов и распределения ролей между ними. Впервые систематически применены, видимо, в языке и системах Smalltalk; разобраны и систематизированы в книге «банды четырёх» — Э. Гаммы и соавторов [65]. Современное развитие этого направления — компонентно-ориентированное программирование, сформировавшееся на базе систем Oberon Н. Вирта и его научно-инженерной школы [65, 66, 67, 68, 69, 70]. Инструментально этот подход впервые поддержан в системе Component Pascal / BlackBox Component Framework [71, 72]. Принципиальной особенностью является отказ от неявных либо усложнённых языковых механизмов, в число которых попадает и наследование реализации (которое в Компонентном Паскале хотя и возможно, но практически не применяется). Любым неявным механизмам опытный инженер предпочитает декомпозицию на примитивные компоненты, распределение между ними ролей и явную их коммутацию через абстрактные интерфейсы-разъёмы [69, 70].

3) **Несущее программное обеспечение** (middleware, мы вводим свой вольный перевод этого термина в целях более точного, на наш взгляд, отражения роли этого ПО), которое обеспечивает выполнение, взаимодействие и управление в распределённых системах. Выход в распределённую среду вносит ряд качественных особенностей, которые не возникают на уровнях 1-2 (например, в локальной системе во многих случаях можно опираться на предположение, что оборудование безотказно; в распределённом приложении нужно исходить из обратного предположения о возможности в любой момент столкнуться с отказом узла или линии связи). Распространённый подход 90-х гг., предполагающий простые клиент-серверные архитектуры (трёхзвенные с центральной монолитной СУБД), должен быть переосмыслен в связи с новыми задачами распределённых Интернет-служб и «облачных вычислений» (cloud computing). Это требует обращения к опыту создания распределённых систем специального назначения, который долгое время оставался невостребованным в широком программировании (в частности, этот опыт в значительной мере сконцентрирован в технологиях, основанных на языке Ada) [23, 34, 73].

4) **Межпрограммные конструктивные схемы**, применяемые в распределённых системах.

5) **Средства проектирования архитектуры, системного анализа, спецификации требований.** Данная тема столь обширна, что мы отметим только частные аспекты, кажущийся наиболее значимым в настоящее время. В связи с расширением задач управляющего ПО (системное программирование, встроенные системы, сложные распределённые службы и т. п.) возникает острая потребность в средствах описания поведения и взаимодействия. Если для разработки «обычных» информационных систем были необходимы преимущественно средства статического и потокового моделирования (например, IDEF-диаграммы), то сегодня необходимы инструменты для проектирования управляющих систем. Такие инструменты существуют и применяются в области встроенных систем, систем реального времени (например, аэрокосмическая отрасль). Часто они основаны на подходе конечных автоматов и ансамблей параллельных автоматов, менее распространены, но также применяются сети Петри. Стоит обратить внимание на такие технологии, как SWITCH [53], ДРАКОН [18, 19] (результат «конверсии» системы ГРАФИТ-ФЛОКС, применяющейся во ФГУП НПЦ АП для создания ПО ракет-носителей и разгонных блоков), систему РЕФЛЕКС [74]; существуют также много закрытых технологий, подобных системе ГРАФКОНТ [75] или циклу разработки бортовых программ НПО им. Решетнева [32, 33]. Получают распространение формальные методы верификации управляющих систем, в частности, подход Model Checking [78], Рекомендуем также по этому вопросу книги А.А. Тюгашева [76, 77].

По проблеме системного анализа и моделирования мы считаем одним из перспективных подходов методологию УФО С.И. Маторина [79], основанную на функциональной системологии Г.П. Мельникова [80, 81]. Эти концепции отличает отход от теоретико-множественных формальных методов (которые многими специалистами в самой математике считаются почти бесплодными для реальных задач [11, 12, 13]) в пользу формально-семантических подходов.

5. Роль языка программирования.

Может показаться странным, что мы уделяем так много внимания, в контексте самых разных вопросов, применяемому языку программирования. Роль этого основного инструмента часто недооценивается («какая разница, на чём реализовывать, если основные проблемы лежат на уровне проектирования»), на самом же деле, именно он является основным звеном, связывающим программирование как проектную деятельность с физической машиной. Именно язык программирования ответственен за превращение проекта в исполняемую систему, а следовательно, он имеет влияние на большинство эксплуатационных качеств системы (так же, как в «традиционной» инженерии на это влияет материал и его свойства). В силу этого к языку возникает очень жёсткий набор требований, которым он должен удовлетворить. Эти требования столь давно известны, что, право слово, стыдно их повторять в среде профессионалов. Они были ясно сформулированы ещё во времена конкурса на единый базовый язык, проведённого Министерством обороны США, в результате которого в финал вышли 4 языка, основанные на Паскале (Алголе) (итогом стало принятие «Зелёного» языка, позднее названного Ada) [27, 28].

Если руководствоваться долгосрочными приоритетами и учитывать всю совокупность факторов (в частности, необходимость максимально продуктивной профессиональной подготовки в ВУЗах; технологическую независимость как от крупных корпораций, так и от неформальных сообществ; желательность хотя бы гипотетической возможности поддерживать инструментарий самостоятельно, и т.п.), то вопрос выбора основного языка и системы программирования оказывается центральным. В калейдоскопе разных платформ, разных задач, экспериментов с разными проектными методами и языками спецификаций именно язык программирования должен быть устойчивым ядром для коллектива. Выбор языка «под задачу», или, что ещё хуже, «в соответствии с требованиями заказчика» — пример доминирования сиюминутных соображений. Самое плохое здесь то, что метание между языками препятствует накоплению фонда базовых средств в коллективе, не даёт сформироваться арсеналу инструментов, а **ведь такой фонд является одним из главных активов любой компании**.

Хороший инструмент программирования не должен требовать к себе постоянного внимания. **Принцип «выбрать один раз и забыть» — вот что здесь требуется.**

Отметим, что многие современные задачи (например, распределённые сетевые службы, научные задачи, игростроение и т.п.) требуют от языка программирования сочетания эффективности (прозрачной компилируемости в машинный код, без потерь быстродействия и непредсказуемых побочных эффектов) и динамичности (автоматического управления памятью, динамической загрузки и связывания модулей, рефлексии и метапрограммирования). Распространение получил биязыковый подход, когда эти роли возлагаются на разные языки (например, C/C++ и Python, Lua, собственные скриптовые языки). Если провести две оси (эффективность и динамичность), то большинство языков будут группироваться вдоль одной из них.

Хорошими показателями по обоим осям обладают языки семейства Oberon (такая плоскость сравнения предложена Ф.В. Ткачёвым, из опыта применения Оберона в сложных задачах символической алгебры).

Мы, из собственного опыта, выделяем следующие три требования к универсальному промышленному языку программирования (в отличие от описанной выше плоскости «эффективность-динамичность», эти свойства — не этапа выполнения, а этапа разработки):

- **простота** (требование диктуется «принципом Калашникова»: «Избыточная сложность — всегда уязвимость» [82]. Хорошо известна метафора В.Ш. Кауфмана [83] — «языки-сундуки» и «языки-чемоданчики». Интересно отметить, что при сравнении Модулы-2 и гораздо более сложной Ады многие специалисты отмечают преимущества первой [27]);
- **жёсткость** (для написания краткосрочных маленьких проектов или прототипов иллюзорное удобство даёт гибкость языка, отсутствие статической типизации и проверок этапа компиляции, синтаксическая свобода. Но эти же «свободы» являются неодолимым препятствием для инженерии крупных систем. Этот вопрос имеет конкретное экономическое значение: хорошо известен принцип «чем раньше обнаружена ошибка, тем дешевле её исправление», при этом зависимость далеко не линейная, говорящая однозначно в пользу механизмов раннего обнаружения. Процитируем, вслед за Информатикой-21 [16], Конрада Лоренца: *«Функция всех структур — сохранять форму и служить опорой — требует, по определению, в известной мере пожертвовать свободой. Можно привести такой пример: червяк может согнуть свое тело в любом месте, где пожелает, в то время как мы, люди, можем совершать движения только в суставах. Но мы можем выпрямиться, встать на ноги — а червяк не может»*);
- **расширяемость** (в простом и жёстком языке должны быть заложены базовые механизмы для неограниченного расширения систем, для обеспечения безболезненной и надёжной эволюции. В Оберонах такое расширение обеспечено полноценной модульностью, с раздельной компиляцией и динамической загрузкой, расширяемыми записями и сопряженными с ними средствами полиморфизма, автоматическим управлением памятью).

Можно перечислить достаточно много языков, удовлетворяющих отдельно одному или двум из названных критериев. Например, простые, изящные и одновременно расширяемые LISP, Scheme, Smalltalk, Forth и многие другие. Жёсткая, соответствующая взыскательному «инженерному вкусу», но достаточно объёмная и «многовариантная» Ada, которая, кроме того, имеет проблемы с динамической расширяемостью. Достаточно простой и жёсткий (а кроме того, предельно производительный), «старина FORTRAN». Но сочетанием всех названных качеств, видимо, обладают только Обероны — и их промышленный представитель Компонентный Паскаль, который в силу этого и может считаться современным универсальным языком промышленного программирования [84, 85, 69, 70].

Язык программирования должен быть поддержан гибкой и расширяемой системой программирования, которая стирает границы между этапами разработки и выполнения. Это характерно для интерпретируемых и динамических языков, но также возможно и для компилируемых, что убедительно доказано системой BlackBox Component Builder [71]. Именно стирание названной границы между этапами жизненного цикла системы является основным фактором для производительной разработки. При таком подходе язык программирования оказывается одновременно скриптовым и макроязыком, помогающим разработчику автоматизировать свою деятельность. Другое мощное средство обеспечения процесса разработки — интерфейс на основе активных составных документов (текстово-графический, командно-графический интерфейс) [71]. В итоге, жизненный цикл программной системы оказывается основан на расширении базовых каркасов новыми компонентами и организации постоянной эволюции (часто прямо во время выполнения).

6. Организационная инфраструктура.

Имеет место эмпирический принцип: **порождаемая система наследует свойства от порождающей**. Плохо организованная программная система порождает плохой сервис для своих пользователей. Плохо организованная голова разработчика порождает плохие программные системы. Плохо организованный процесс коллективной разработки точно так же оказывает отрицательное влияние на результат.

Не имея возможности углубляться в данный вопрос, отметим, что попытки жёстко зафиксировать процесс создания системы обречены на провал, и это очевидно для любого человека, понимающего природу умственного труда, в отличие от тех, кто усиленно продвигает «мёртвые» стандарты, подобные «системам менеджмента качества», ISO-9000, и т. п.

Организация проектной работы должна заключаться в структурировании информационных фондов проекта — проектной документации, моделей, исходных текстов и всех других необходимых материалов. Суммируя опыт, изложенный различными авторами [20, 21, 22, 23, 24, 25, 26] (например, И. Соммервиллом [22]), организация может выработать для себя подходящую структуру проектных фондов.

7. Образовательная инфраструктура.

Проблема общей и профессиональной подготовки в сфере программирования наиболее полно рассматривается Международным общественным научно-образовательным проектом «Информатика-21» [5]. Наша позиция по этим вопросам отражена в ряде публикаций [84, 9, 86, 87].

Литература

1. Дейкстра, Э. Смиренный программист. EWD340. // Эдсгер Дейкстра: избранные статьи. Web: **http://store.oberoncore.ru/lib/book/dijkstra.pdf**.
2. Дейкстра, Э. Два взгляда на программирование. EWD540. // Эдсгер Дейкстра: избранные статьи. Web: **http://store.oberoncore.ru/lib/book/dijkstra.pdf**.
3. Дейкстра, Э. Почему программное обеспечение такое дорогое? EWD648. // Эдсгер Дейкстра: избранные статьи. Web: **http://store.oberoncore.ru/lib/book/dijkstra.pdf**.
4. Дейкстра, Э. Научная фантастика и научная реальность в информатике. EWD952. // Эдсгер Дейкстра: избранные статьи. Web: **http://store.oberoncore.ru/lib/book/dijkstra.pdf**.
5. Международный общественный научно-образовательный проект «Информатика-21». Web: **http://www.inr.ac.ru/~info21/**
6. Вигнер, Е. Непостижимая эффективность математики в естественных науках. // Е. Вигнер. Этюды о симметрии. Пер. с англ. Ю.А.Данилова, под ред. Я.А.Смородинского. — М.: Мир, 1971. — 318 с.
7. Stroustrup, B. What Should We Teach New Software Developers? Why? // Communications of the ACM. Vol. 53 No. 1,

Pages 40-42. — 10.1145/1629175.1629192. Web: http://cacm.acm.org/magazines/2010/1/55760-what-should-we-teach-new-software-developers-why/fulltext

8. Перевод статьи Бьярна Страуструпа “What Should We Teach New Software Developers? Why?”. Web: http://dbarashev.habrahabr.ru/blog/80623/

9. Ермаков, И.Е.; Рюмшин, Б.В. О грамотной алгоритмизации. // Сборник докладов секции «Информатика образования» VII межд. конф-и памяти акад. А.П. Ершова «Перспективы систем информатики». — Новосибирск, 2009. Web: http://metasystems.ru/download/edu/I21-a-057a-Novosib-PSI-2009-eierbv.pdf

10. Ермаков, И.Е. Структурирование «промышленного» цикла. // OberonCore. Web: http://oberoncore.ru/wiki/структурирование_промышленного_цикла

11. Арнольд, В.А. О преподавании математики. // Успехи математических наук. — 1998. — т. 53, вып. 1. Web: http://www.mccme.ru/edu/index.php?ikey=viarn_oprepmat

12. Арнольд, В.А. Математическая дуэль вокруг Бурбаки. // Вестник Российской академии наук. — 2002. — т. 72, N 3. — С. 245-250.

13. Новиков, С.П. Математика на пороге XXI века (Историко-математические исследования). // Сайт РГГУ. Web: http://www.rsuh.ru/article.html?id=50768

14. Ткачёв, Ф.В. Компонентный Паскаль/Оберон как технологическая основа единой системы вводных курсов информатики. // Информатика-21. Web: http://www.inr.ac.ru/~info21/pdf/tomsk2007fvt.pdf

15. Обсуждение «Математика — информатика — языки/литература». // Информатика-21. Web: http://forum.oberoncore.ru/viewtopic.php?f=7&t=741

16. Ткачёв, Ф.В. О дисциплине программирования. Почему в Обероне/Компонентном Паскале ограничена «свобода творчества» программиста. // Информатика-21. Web: http://www.inr.ac.ru/~info21/blackbox/disciplina/welcome.html

17. Вейценбаум, Дж. Возможности вычислительных машин и человеческий разум (от суждений к вычислениям). / Дж. Вейценбаум. — Пер. с англ. — М. Радио и связь. — 1982. Web: http://oops.tepkom.ru/~msk/Weiz/Weizenbaum.html

18. Паронджанов, В.Д. Как улучшить работу ума. Алгоритмы без программистов — это очень просто! / В.Д. Паронджанов. — М.: Дело. — 2001. — 360 с.

19. Визуальный язык «Дракон». // OberonCore. Web: http://oberoncore.ru/wiki/drakon/start

20. Брукс, Ф. Мифический человеко-месяц или Как создаются программные системы. / Ф. Брукс. — Пер. с англ. — М. — 1975.

21. Фокс, Дж. Программное обеспечение и его разработка. / Дж. Фокс. — Пер. с англ. — М.: Мир. — 1985. — 368 с.

22. Соммервилл, И. Инженерия программного обеспечения. 6-е издание. / И. Соммервилл. — Пер. с англ. — М.: Вильямс. — 2002. — 642 с.

23. Басс, Л. Архитектура программного обеспечения на практике. 2-е издание. / Л. Басс, П. Клементс, Р. Кацман. — Пер. с англ. — М.: Питер. — 2006. — 575 с.

24. Липаев, В.В. Программная инженерия. Методологические основы (лекции). / В.В. Липаев. — М.: ТЕИС. — 2006. — 608 с.

25. Терехов, А.Н. Технология программирования. / А.Н. Терехов. — М.: Бином. — 2007. — 152 с.

26. Жоголев, Е.А. Технология программирования. / Е.А. Жоголев. — М.: Научный мир. — 2004. — 216 с.

27. Янг, С. Алгоритмические языки реального времени. / С. Янг. — Пер. с англ. — М.: Мир. — 1986. — 400 с. Web: http://317.metasystems.ru/lib/exe/fetch.php/knowledge:young.pdf

28. Русский ресурс по языку Ada. Web: http://ada-ru.org

29. Рыбин, С.И. Ада в России: обзор. Web: http://ada-ru.org/wiki/ada83_in_rus

30. Рыбин, С.; Фофанов, В. Язык Ада — 20 лет спустя. Web: http://ada-ru.org/files/ada_20_years.pdf.gz

31. Перминов, К.; Перминов, О. Ада — язык разработки больших программных комплексов реального времени. // Открытые системы. — 1996. — Вып. 6. Web: http://www.osp.ru/os/1996/06/179020/

32. Колташёв, А.А. Модула-2 в российском космосе. // Информатика-21. Web: http://www.inr.ac.ru/~info21/texts/aakmodula2.htm

33. Колташёв, А.А. Эффективная технология управления циклом жизни бортового программного обеспечения спутников связи и навигации. // Авиакосмическое приборостроение. — 2006. — N 12. — С. 20-25. Web: http://www.inr.ac.ru/~info21/pdf/AAKpaper2006.pdf

34. Бар, Р. Язык Ада в проектировании систем. / Р. Бар. — Пер. с англ. — М.: Мир. — 1988. — 320 с. Web: http://317.metasystems.ru/books/System-Design-With-Ada.djvu

35. Пентковский, В.М. Автокод эльбрус. Принципы построения языка и руководство к пользованию. / В.М. Пентковский. — М.: Наука. — 1982. — 352 с. Web: http://store.oberoncore.ru/lib/book/pvm1982r.djvu

36. Пентковский, В.М. Язык программирования Эль-76. Принципы построения языка и руководство к пользованию. — 2-е изд. испр. и доп. / В.М. Пентковский. — М.: Наука. — 1989. — 367 с. Web: http://store.oberoncore.ru/lib/book/pvm1989r.djvu

37. Сафонов, В.О. Языки и методы программирования в системе «Эльбрус» / В.О. Сафонов. — Под ред. С.С. Лаврова. — М.: Наука. — 1989. — 392 с. Web: http://store.oberoncore.ru/lib/book/svo1989r.djvu

38. Временный научно-технический коллектив «СТАРТ». Web: http://start.iis.nsk.su/

39. «Кронос»: история одного проекта. Web: http://www.kronos.ru/

40. Никлаус Вирт (Niklaus Wirth). // Информатика-21. Web: http://www.inr.ac.ru/~info21/wirth/wirth.htm

41. Что такое Оберон (Oberon). // Информатика-21. Web: http://www.inr.ac.ru/~info21/info/qtooberon.htm

42. Энциклопедия OberonCore. // OberonCore. Web: http://oberoncore.ru/wiki/start

43. The Go Programming Language. Web: http://golang.org/

44. Go и Оберон. // Информатика-21. Web: http://www.inr.ac.ru/~info21/go.htm

45. Software Engineering Body of Knowledge (SWEBOK). Web: http://swebok.org/

46. Основы программной инженерии (по SWEBOK). Web: http://swebok.sorlik.ru/index.html

47. Software Enginnering Method and Theory. Web: http://www.semat.org

48. Рыбин, С.И. Современное ИТ: индустрия и образование. Обзор языка Ada. // OberonCore. — 2006. Web: http://store.oberoncore.ru/lib/paper/rybin.pdf

49. ЗАО «Ланит-Терком». Web: http://www.lanit.ru/
50. Кафедра системного программирования Математико-механического факультета СПбГУ. Web: http://se.math.spbu.ru/
51. Периодическое издание «Системное программирование». Web: http://www.sysprog.info/
52. Факультет информационных технологий и программирования СПбГУ ИТМО. Web: http://fitp.ifmo.ru/
53. Сайт по автоматному программированию А.А. Шалыто. Web: http://is.ifmo.ru/
54. Одинцов, И.О. Профессиональное программирование. Системный подход. — 2-е изд. / И.О. Одинцов. — СПб.: БХВ-Петербург. — 2004. —624 с.
55. ООО «Метасистемы». Web: http://metasystems.ru/
56. Технологический институт ОрёлГТУ. Web: http://ostu.ru/inst/ti/main/
57. Факультет СПО ТИ ОрёлГТУ. Web: http://ostu.ru/faculties/fspo_fac/main/
58. Сайт специальных дисциплин программистских специальностей ФСПО ТИ ОрёлГТУ. Web: http://317.metasystems.ru/
59. Физико-математический факультет Орловского государственного университета. Web: http://phys-math.ru/
60. Пратт, Т.; Зелковиц, М. Языки программирования: разработка и реализация. 4-е издание. / Т. Пратт, М. Зелковиц. — Под общей редакцией А. Матросова. — СПб.: Питер. — 2002. —688 с.
61. Прикладная логика. Учебное пособие. / Н.Н. Непейвода. — Новосибирск: НГУ. — 2000. — 529 с.
62. Непейвода, Н.Н. Математик и прикладник: о взаимо(не)понимании. // Вестник Удмуртского университета. — Математика. — 2007. — N1. — С. 251-268.
63. Управление, информация, интеллект. / Под редакцией А.И Берга, Б.В. Бирюкова, Е.С. Геллера, Г.Н. Поварова. — М.: Мысль. — 1976. - 383 с.
64. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. — СПб.: Питер. — 2001. — 368 с.
65. Пфистер, К. Компонентное ПО. — Пер. с англ. И.Е. Ермакова. // OberonCore — 2005. — Web: http://oberoncore.ru/blackbox/articles
66. Szyperski, C. Insight EthOS — On Object Orientation In Operating Systems. — ETH Diss. No 9884. — Zurich, Switzerland. — 1992. — ISBN 3-7281-1948-2. Web: http://research.microsoft.com/en-us/um/people/cszypers/books/insight-ethos.htm
67. Szyperski, C. Component Software. Beyond Object-Oriented Programming. — Addison Wesley Longman. — 1998.
68. Губанов, С.Ю. Секреты модульных систем. // OberonCore. — 2006. Web: http://oberoncore.ru/programming/articles
69. Ермаков, И.Е. Оберон-технологии: что это такое? // OberonCore. — 2006. Web: http://oberoncore.ru/programming/oberon-technology
70. Ермаков, И.Е. Некоторые идеи архитектуры Оберон-систем. // OberonCore. — 2007. Web: http://oberoncore.ru/programming/oberon-technology
71. BlackBox Component Builder. // OberonCore. Web: http://oberoncore.ru/wiki/blackbox
72. О компании Oberon Microsystems. // Информатика-21. Web: http://www.inr.ac.ru/~info21/info/oberon_microsystems.htm
73. Бэкон, Д.; Харрис, Т. Операционные системы. Параллельные и распределённые системы. / Д. Бэкон, Т. Харрис. — Пер. с англ. — СПб.: Питер-БХВ. — 2004. — 800 с.
74. Язык Рефлекс — диалект языка Си для программирования управляющих алгоритмов промышленной автоматизации. Web: http://reflex-language.narod.ru/
75. Система ГРАФКОНТ/ГЕОЗ автоматизации создания бортовых управляющих алгоритмов и программ реального времени для космических аппаратов. Web: http://grafkont.boom.ru/
76. Калентьев, А.А.; Тюгашев, А.А. ИПИ/CALS технологии в жизненном цикле комплексных программ управления. / А.А. Калентьев, А.А. Тюгашев. — Самара: Изд-во Самарского научного центра РАН. — 2006. — 285 с.
77. Тюгашев, А.А. Графические языки программирования и их применение в системах управления реального времени. / А.А. Тюгашев. — Самара: Изд-во Самарского научного центра РАН. — 2009.
78. Карпов, Ю.Г. MODEL CHECKING. Верификация параллельных и распределённых систем. / Ю.Г. Карпов. — СПб.: БХВ-Петербург. — 2010. — 560 с.
79. Маторин, С.И. Анализ и моделирование бизнес-систем: системологическая объектно-ориентированная технология. / С.И. Маторин. — Харьков: ХНУРЭ. — 2002. — 322 с.
80. Мельников, Г.П. Азбука математической логики. / Г.П. Мельников. — М.: Знание. — 1967. — 105 с. Web: http://317.metasystems.ru/lib/exe/fetch.php/knowledge:melnikov_azbuka.djvu
81. Мельников, Г.П. Системология и языковые аспекты кибернетики. / Г.П. Мельников. — М.: Советское Радио. — 1978. Web: http://www.philol.msu.ru/~lex/melnikov/meln_r/titl.htm
82. Принцип Калашникова. // Информатика-21. Web: http://www.inr.ac.ru/~info21/princypKalashnikova.htm
83. Кауфман, В.Ш. Языки программирования. Концепции и принципы. Методический материал (на правах рукописи) / В.Ш. Кауфман. — М.: МГУ, ВМиК. — 1986.
84. Ермаков, И.Е. Опыт применения Оберона при интеграции исследований, образования и производства. // Сборник докладов секции «Информатика образования» VII межд. конф-и памяти акад. А.П. Ершова «Перспективы систем информатики». — Новосибирск. — 2009. Web: http://metasystems.ru/download/science/I21-a-056-Novosib-PSI-2009-eie.pdf
85. Ермаков, И.Е. Возможности Оберон-технологий для сложных задач системного программирования. // Доклады конференции «Свободный полёт — 2009». — Уфа, УГАТУ. — 2009. Web: http://metasystems.ru/download/science/a-039-ufa-2009-eie.pdf
86. Ермаков, И.Е.; Рюмшин, Б.В. О грамотной алгоритмизации (Опыт построения систематического базового курса программирования). // Презентация для секции «Информатика образования» VII межд. конф-и памяти акад. А.П. Ершова «Перспективы систем информатики» — Новосибирск. — 2009. Web: http://metasystems.ru/download/edu/I21-a-060-Novosib-PSI-2009-eie-P.pdf
87. Ермаков, И.Е. Проблема обучения программированию и пути её решения. // Сборник трудов Всероссийской конференции «Проблемы информатизации образования» (ТУЛАИНФОРМ-2008). — Тула: ТулГУ. — 2008. — с. 6-10. Web:

http://metasystems.ru/download/edu/I21-a-051a-Tulainform-2008-eie.pdf (статья) / http://metasystems.ru/download/edu/I21-a-052a-Tulainform-2008-eie-P.pdf (презентация).