ВЫСШАЯ ШКОЛА ЭКОНОМИКИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

С. В. Зыков

ПРОГРАММИРОВАНИЕ

ФУНКЦИОНАЛЬНЫЙ ПОДХОД

УЧЕБНИК И ПРАКТИКУМ
ДЛЯ АКАДЕМИЧЕСКОГО БАКАЛАВРИАТА

Рекомендовано Учебно-методическим отделом высшего образования в качестве учебника и практикума для студентов высших учебных заведений, обучающихся по инженерно-техническим направлениям и специальностям

Книга доступна в электронной библиотечной системе biblio-online.ru

Москва ▪ Юрайт ▪ 2016

УДК 004.42(075.8)
ББК 32.97-018я73
З96

***Автор*:**

Зыков Сергей Викторович — кандидат технических наук, доцент Департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики», доцент кафедры экономики инноваций и управления проектами Инженерно-экономического института Московского авиационного института, доцент кафедры кибернетики факультета кибернетики и информационной безопасности Национального исследовательского ядерного университета «МИФИ», доцент факультета инноваций и высоких технологий Московского физико-технического института.

***Рецензенты*:**

Вольфенгаген В. Э. — доктор технических наук, профессор кафедры кибернетики факультета кибернетики и информационной безопасности Национального исследовательского ядерного университета «МИФИ»;

Александров Д. В. — доктор технических наук, профессор Департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики».

Зыков, С. В.

З96 Программирование. Функциональный подход : учебник и практикум для академического бакалавриата / С. В. Зыков. — М. : Издательство Юрайт, 2016. — 164 с. — Серия : Бакалавр. Академический курс.

ISBN 978-5-9916-8217-6

Книга базируется на творческом синтезе избранных формальных теорий (лямбда-исчисление, комбинаторная логика, теория категорий и др.) и уникальной технологической платформы Microsoft .NET, обеспечивающей практически прозрачную интеграцию кода на языках программирования различных типов. Целью издания является формирование адекватного мировоззрения на современное программирование. Книга посвящена основам функционального подхода к программированию на основе языка F#.

Содержание учебника соответствует актуальным требованиям Федерального государственного образовательного стандарта высшего образования.

Книга будет полезна как для опытных программистов, так и для студентов, аспирантов и исследователей, специализирующихся в области компьютерных наук и информационных технологий.

УДК 004.42(075.8)
ББК 32.97-018я73

ISBN 978-5-9916-8217-6

Оглавление

Предисловие

Нахождение глубинной простоты в запутанном клубке сущностей — это и есть творчество в программировании.

Х. Д. Миллс
(*Harlan D. Mills, 1919–1996, Professor of Computer Science, IBM Research Fellow*)

Современное программирование отличается многоаспектностью и все возрастающей сложностью. За последние десятилетия накоплены миллионы строк системных и прикладных библиотек, разработаны тысячи языков, создан целый ряд платформ для профессиональной разработки кода... При этом сложность разрабатываемых программных систем зачастую увеличивается лавинообразно. Возникает вопрос: как сориентироваться в этом многообразии, как выработать подход к программированию, который позволит быстрее и эффективнее адаптироваться к новым языкам, платформам, технологиям? Найти ответ, пусть и отчасти, призвана эта книга.

Еще в 1960-х гг., на заре программной индустрии, было опытным путем принято решение о решении крупных и сложных задач посредством применения принципа «Разделяй и властвуй», или дробления таких задач на относительно мелкие и независимые подзадачи. При этом требовалось довольно точное понимание формата входных и выходных данных для «стыковки» каждой из полученных подзадач, в то время как уточнение их внутренней структуры происходило по мере разработки программной системы. В ходе продвижения от общей концепции системы к ее прототипу и готовому продукту каждая из подзадач могла быть, в свою очередь, разделена на части. Процесс подобного деления, или декомпозиции, происходил до тех пор, пока структура и функции каждой подзадачи, а также ее взаимосвязи с другими задачами, не определялись окончательно.

Первые языки программирования, которые появились в 1940-х гг., содержали явную пошаговую последовательность инструкций, которые должен был выполнить компьютер. Такие языки принято называть императивными («повелительными»). С появлением новых языков программирования возник так называемый декларативный («описательный») подход к программированию. При этом подходе программа представляет собой описание решения задачи на уровне правил или способов преобразования данных.

В то же время, очевидно, что любая сложная программная система состоит из большого количества компонентов, которые, по меньшей мере,

определяют: 1) внутреннюю логику работы и 2) взаимодействие с пользователем, аппаратной и программной платформой. Требования к этим двум видам компонентов, бизнес-логике и интерфейсу, вообще говоря, различны. В первом случае зачастую важнее оказываются компактность, непротиворечивость, полнота, а во втором — скорость взаимодействия и время ответа системы.

Оказывается, что в ряде случаев разработку бизнес-логики и интерфейса предпочтительно вести с использованием разных подходов к программированию, применяя в первом случае декларативный, а во втором — императивный подход. В действительности, эволюция языков программирования происходит существенно более сложным путем. В частности, имеет место тенденция к взаимопроникновению языков программирования, когда изначально императивные языки приобретают декларативные черты и наоборот. С другой стороны, появляется семейство так называемых предметно-ориентированных языков программирования, ориентированных на специфические программные системы (скажем, мобильные, встроенные приложения и т.д.). Перечисленные особенности, впрочем, лишь подчеркивают необходимость тщательного исследования каждого нового компонента на предмет выявления языка программирования, наиболее подходящего для его разработки.

Очевидно, что в таких условиях изучение программирования имеет смысл преимущественно в контексте разработки компонентных систем на базе разнородных языков программирования. При этом для разных языков и подходов весьма желательно сохранять общую платформу компонентной разработки во избежание непроизводительных потерь времени, вызванных необходимостью изучения особенностей новых платформ. Указанным целям в наибольшей степени отвечает платформа .NET, разработанная корпорацией Microsoft более 15 лет назад и объединяющая десятки языков и множество подходов к программированию.

Именно платформа .NET и положена в основу настоящего учебника. При этом изучение программирования начинается с декларативного подхода и функционального стиля, который зачастую позволяет в компактной форме, близкой к привычным математическим формулам, описать «сердце» программ — логику их поведения. В качестве целевого языка на этом этапе обучения выбран язык F#, позволяющий разрабатывать не только учебные программы, но и крупномасштабные приложения. Язык F# поможет глубже познакомиться с функциональным подходом к программированию, логикой работы программ, а также вычислительной средой на основе абстрактной машины, подобной виртуальной машине для среды разработки .NET.

Разработанные на языке F# компоненты предназначены для построения разнородных компонентных приложений на базе той же платформы .NET в ходе второй части курса. Она познакомит с языком C# — основным и «родным» языком платформы .NET, более удобным для разработки интерфейсов, чем F# — и объектно-ориентированным подходом к программированию, который ближе к императивному стилю разработки. Такой

порядок изучения позволит прийти к более глубокому пониманию природы объектов, поскольку функцию можно также считать моделью объекта. Кроме того, совместное изучение различных языков и стилей программирования поможет лучше сформировать профессиональный взгляд на современные методы и технологии разработки программных систем, зачастую объединяющих большое количество разнородных компонент на общей платформе.

Курс завершается изучением возможностей построения на платформе .NET небольших программных систем, состоящих из компонент на языках F# и C#. При этом представляется возможность проявить себя как в индивидуальной, так и в командной разработке.

В результате изучения дисциплины студент должен:

знать

- основы архитектуры электронно-вычислительных машин (ЭВМ) и программных систем;
- способы реализации программных систем, в том числе распределенных;
- как использовать операционные системы, сетевые технологии, средства разработки программного обеспечения;

уметь

- использовать различные подходы к разработке программного обеспечения и применять их на практике;
- самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения;
- применять основы программирования к проектированию, конструированию и тестированию программных продуктов;

владеть

- навыками чтения, понимания и выделения главной идеи прочитанного исходного кода (с минимальной зависимостью от языка реализации);
- методами и средствами обработки информации посредством современных компьютерных технологий, в том числе в глобальных компьютерных сетях;
- способами самостоятельного решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте.

***Желаем успехов*!**

Глава 1
МЕСТО ФУНКЦИОНАЛЬНОГО ПОДХОДА И MICROSOFT .NET В СЕМЕЙСТВЕ ЯЗЫКОВ И ПОДХОДОВ К ПРОГРАММИРОВАНИЮ

1.1. Классификация языков программирования

В данном параграфе исследуются вопросы истории и эволюции языков и подходов к программированию, анализируются их достоинства и недостатки, строятся классификации языков и подходов к программированию.

Первые языки программирования возникли относительно недавно: различные исследователи указывают 1920-е, 1930-е и даже 1940-е гг. Нашей задачей является не установление самого раннего языка, а поиск закономерностей в их развитии.

Как и следовало ожидать, первые языки программирования, как и первые ЭВМ, были довольно примитивны и ориентированы на численные расчеты. Это были и чисто теоретические научные расчеты (прежде всего, математические и физические), и прикладные задачи, в частности в области военного дела. Программы, написанные на ранних языках программирования, представляли собой линейные последовательности элементарных операций с регистрами, в которых хранились данные.

Нужно отметить, что ранние языки программирования были оптимизированы под ту аппаратную архитектуру конкретного компьютера, для которого они предназначались, и, хотя они обеспечивали высокую эффективность вычислений, до стандартизации было еще далеко. Программа, которая была вполне работоспособной на одной вычислительной машине, зачастую не могла быть выполнена на другой.

Таким образом, ранние языки программирования существенно зависели от того, что принято называть средой вычислений, и приблизительно соответствовали современным машинным кодам или языкам ассемблера.

1950-е гг. ознаменовались появлением языков программирования так называемого «высокого уровня», по сравнению с ранее рассмотренными предшественниками, соответственно именуемыми низкоуровневыми языками. При этом различие состоит в повышении эффективности труда разработчиков за счет абстрагирования или отвлечения от конкретных деталей аппаратного обеспечения. Одна инструкция (оператор) языка высокого уровня соответствовала последовательности из нескольких низкоуровне-

вых инструкций, или команд. Исходя из того, что программа, по сути, представляла собой набор директив, обращенных к компьютеру, такой подход к программированию получил название императивного.

Еще одной особенностью языков высокого уровня была возможность повторного использования ранее написанных программных блоков, выполняющих те или иные действия, посредством их идентификации и последующего обращения к ним, например, по имени. Такие блоки получили название функций или процедур, и программирование приобрело более упорядоченный характер. Кроме того, с появлением языков высокого уровня зависимость реализации от аппаратного обеспечения существенно уменьшилась. Платой за это стало появление специализированных программ, преобразующих инструкции возникших языков в коды той или иной машины, или трансляторов, а также некоторая потеря в скорости вычислений, которая, впрочем, компенсировалась существенным выигрышем в скорости разработки приложений и унификацией программного кода.

Стоит отметить, что операторы и ключевые слова новых языков программирования были более осмысленны, чем безликие цифровые последовательности кодов, что также обеспечивало повышение производительности труда программистов.

Естественно, для обучения новым языкам программирования требовались значительные затраты времени и средств, а эффективность реализации на прежних аппаратных возможностях снижалась. Однако трудности эти носили временный характер, и, как показала практика программирования, многие из первых языков высокого уровня оказались настолько удачно реализованными, что активно используются и сегодня. Одним из таких примеров является язык Fortran, реализующий вычислительные алгоритмы. Другой пример — язык APL, трансформировавшийся в BPL, а затем в C. Основные конструкции последнего остаются неизменными вот уже несколько десятилетий и присутствуют в языке C#, который нам предстоит изучить.

Примеры других языков программирования: ALGOL, COBOL, Pascal, Basic.

В 1960-х гг. возникает новый подход к программированию, который до сих пор успешно конкурирует с императивным, — декларативный подход. Суть его состоит в том, что программа представляет собой не набор команд, а описание действий, которые необходимо осуществить. Этот подход, как мы увидим впоследствии, существенно проще и прозрачнее формализуем математическими средствами. Отсюда следует тот факт, что программы проще проверять на наличие ошибок (тестировать), а также на соответствие заданной технической спецификации (верифицировать).

Высокая степень абстракции также является преимуществом данного подхода. Фактически программист оперирует не набором инструкций, а абстрактными понятиями, которые могут быть достаточно обобщенными.

На начальном этапе развития декларативным языкам программирования было сложно конкурировать с императивными в силу объективных трудностей при создании эффективной реализации трансляторов. Про-

граммы работали медленнее, однако они могли решать более абстрактные задачи с меньшими трудозатратами.

Одним из путей развития декларативного стиля программирования стал функциональный подход, возникший с появлением и развитием языка LISP. Отличительной особенностью данного подхода является то обстоятельство, что любая программа, написанная на таком языке, может интерпретироваться как функция с одним или несколькими аргументами. Такой подход дает возможность прозрачного моделирования текста программ математическими средствами, а значит, весьма интересен с теоретической точки зрения.

Сложные программы при таком подходе строятся посредством агрегирования функций. При этом текст программы представляет собой функцию, некоторые аргументы которой можно также рассматривать как функции. Таким образом, повторное использование кода сводится к вызову ранее описанной функции, структура которой, в отличие от процедуры императивного языка, прозрачна математически. Более того, типы отдельных функций, используемых в функциональных языках, могут быть переменными. Таким образом, обеспечивается возможность обработки разнородных данных (например, упорядочение элементов списка по возрастанию для целых чисел, отдельных символов и строк), или полиморфизм.

Еще одним важным преимуществом реализации языков функционального программирования является автоматизированное динамическое распределение памяти компьютера для хранения данных. При этом программист избавляется от рутинной обязанности контролировать данные, а при необходимости может запустить функцию «сборки мусора» — очистки памяти от тех данных, которые больше не потребуются программе (обычно этот процесс периодически инициируется компьютером).

Таким образом, при создании программ на функциональных языках программист сосредотачивается на области исследований (предметной области) и в меньшей степени заботится о рутинных операциях (обеспечении правильного с точки зрения компьютера представления данных, «сборке мусора» и т.д.).

Поскольку функция является естественным формализмом для языков функционального программирования, реализация различных аспектов программирования, связанных с функциями, существенно упрощается. В частности, интуитивно прозрачным становится написание рекурсивных функций, т.е. функций, вызывающих самих себя в качестве аргумента. Кроме того, естественной становится и реализация обработки рекурсивных структур данных (например, списков — базовых элементов, скажем, для семейства языков LISP, деревьев и др.).

Благодаря реализации механизма сопоставления с образцом такие языки, как ML и Haskell, весьма неплохо применимы для символьной обработки.

Примерами функциональных языков программирования помимо упомянутых LISP, ML, Haskell являются также SML, F#.

В 1970-х гг. возникла еще одна ветвь языков декларативного программирования, связанная с проектами в области искусственного интеллекта,

а именно *языки логического программирования*. Согласно логическому подходу к программированию, программа представляет собой совокупность правил или логических высказываний. Кроме того, в программе допустимы логические причинно-следственные связи, в частности на основе операции импликации. Таким образом, языки логического программирования базируются на классической логике и применимы для систем логического вывода, в частности для так называемых экспертных систем. На языках логического программирования естественно формализуется логика поведения, и они применимы для описаний правил принятия решений, например в системах, ориентированных на поддержку бизнеса.

Важным преимуществом подхода является достаточно высокий уровень машинной независимости, а также возможность откатов — возвращения к предыдущей подцели при отрицательном результате анализа одного из вариантов в процессе поиска решения (скажем, очередного хода при игре в шахматы), что избавляет от необходимости поиска решения полным перебором вариантов и увеличивает эффективность реализации.

Одним из недостатков логического подхода в концептуальном плане является специфичность класса решаемых задач. Другой недостаток практического характера состоит в сложности эффективной реализации для принятия решений в реальном времени, скажем, для систем жизнеобеспечения.

Нелинейность структуры программы является общей особенностью декларативного подхода и, строго говоря, оригинальной характеристикой, а не объективным недостатком.

В качестве примеров языков логического программирования можно привести Prolog (название возникло от слов PROgramming in LOGic) и Mercury.

Важным шагом на пути к совершенствованию языков программирования стало появление объектно-ориентированного подхода к программированию и соответствующего класса языков. В рамках данного подхода программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов). Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модели сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области. Перемещаясь по иерархии классов от более общих понятий предметной области к более конкретным (или от более сложных — к более простым) и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый им реальный мир.

Использование ранее разработанных (возможно, другими коллективами программистов) библиотек объектов и методов позволяет значительно сэкономить трудозатраты при производстве программного обеспечения, особенно типичного.

Объекты, классы и методы могут быть полиморфными, что делает реализованное программное обеспечение более гибким и универсальным.

Сложность адекватной (непротиворечивой и полной) формализации объектной теории порождает трудности тестирования и верификации созданного программного обеспечения. Пожалуй, это обстоятельство является одним из самых существенных недостатков объектно-ориентированного подхода к программированию.

Наиболее известным примером объектно-ориентированного языка программирования является язык C++, развившийся из императивного языка C. Его прямым потомком и логическим продолжением является язык C#.

Развитием событийно управляемой концепции объектно-ориентированного подхода стало появление в 1990-х гг. целого класса языков программирования, которые получили название *языков сценариев*, или *скриптов*. В рамках данного подхода программа представляет собой совокупность возможных сценариев обработки данных, выбор которых инициируется наступлением того или иного события (щелчок по кнопке мыши, попадание курсора в ту или иную позицию, изменение атрибутов того или иного объекта, переполнение буфера памяти и т.д.). События могут инициироваться как операционной системой (в частности, Windows), так и пользователем.

Основные достоинства языков данного класса унаследованы от объектно-ориентированных языков программирования. Это интуитивная ясность описаний, близость к предметной области, высокая степень абстракции, хорошая переносимость. Широкие возможности повторного использования кода также унаследованы сценарными языками от объектно-ориентированных предков.

Существенной положительной чертой языков сценариев является их совместимость с передовыми инструментальными средствами автоматизированного проектирования и быстрой реализации программного обеспечения, или так называемыми CASE- (Computer-Aided Software Engineering) и RAD- (Rapid Application Development) средствами.

Одним из наиболее передовых инструментальных комплексов для быстрой разработки приложений является Microsoft Visual Studio .NET, изучение и использование возможностей которого изучается в данной книге.

Естественно, что вместе с достоинствами объектно-ориентированного подхода языки сценариев унаследовали и ряд недостатков. К последним прежде всего относятся сложность тестирования и верификации программ и возможности возникновения в ходе эксплуатации множественных побочных эффектов, проявляющихся за счет сложной природы взаимодействия объектов и среды, представленной интерфейсами с подчас многочисленными одновременно работающими пользователями программного обеспечения, операционной системой и внешними источниками данных.

Примерами сценарных языков программирования являются JavaScript и VBScript.

Еще одним весьма важным классом языков программирования являются языки поддержки параллельных вычислений. Программы, написанные на этих языках, представляют собой совокупность описаний процессов, которые могут выполняться как в действительности одновременно, так и в псевдопараллельном режиме. В последнем случае устройство, обрабатывающее процессы, функционирует в режиме разделения времени, выделяя время на обработку данных, поступающих от процессов, по мере необходимости (а иногда с учетом последовательности или приоритетности выполнения операций).

Языки параллельных вычислений позволяют достичь заметного выигрыша в эффективности при обработке больших массивов информации, поступающих, например, от одновременно работающих пользователей, либо имеющих высокую интенсивность (как, например, видеоинформация или звуковые данные высокого качества).

Другой весьма значимой областью применения языков параллельных вычислений являются системы реального времени, в которых пользователю необходимо получить ответ от системы непосредственно после запроса. Системы такого рода отвечают за жизнеобеспечение и принятие ответственных решений.

Обратной стороной достоинств рассматриваемого класса языков программирования является высокая стоимость разработки программного обеспечения, а следовательно, разработка относительно небольших программ широкого (например, бытового назначения) зачастую нерентабельна.

Примерами языков программирования с поддержкой параллельных вычислений могут служить Ada, Modula-2 и Oz.

Итак, мы рассмотрели историю и эволюцию языков программирования и основные подходы к разработке программных систем. Сделана попытка классификации языков и подходов к программированию, а также проведен анализ достоинств и недостатков, присущих тем или иным подходам и языкам. Заметим, что приведенную классификацию не следует считать единственно верной и абсолютной, поскольку языки программирования постоянно развиваются и совершенствуются, и недавние недостатки устраняются с появлением необходимых инструментальных средств или теоретических обоснований.

Подводя итоги, кратко перечислим рассмотренные подходы к программированию:

- ранние неструктурные подходы;
- структурный или модульный подход (задача разбивается на подзадачи, затем на алгоритмы, составляются их структурные схемы и происходит реализация);
- функциональный подход;
- логический;
- объектно-ориентированный;
- смешанный (некоторые подходы возможно комбинировать);

• компонентно-ориентированный (программный проект рассматривается как множество компонент, такой подход принят, в частности, в .NET);
• чисто объектный подход (идеальный с математической точки зрения вариант, который пока не реализован практически).

Контрольные вопросы

Вопрос 1.

Вариант 1: какие из перечисленных языков программирования основаны на функциональном подходе?

а) SML и ProLog;
б) LISP и ProLog;
в) SML и LISP (+).

Вариант 2: что отличает императивные языки программирования от декларативных?

а) степень зависимости от среды реализации;
б) стиль программирования (+);
в) структура программы (+).

Вариант 3: в чем состоит особенность языков объектно-ориентированного программирования?

а) этот класс языков основан на сценариях;
б) этот класс языков концептуально близок к любой предметной области (+);
в) этот класс языков является наиболее машинно-независимым.

Вопрос 2.

Вариант 1: какие из перечисленных языков программирования основаны на объектно-ориентированном подходе?

а) C# и SML;
б) C# и C++ (+);
в) C# и ProLog.

Вариант 2: что отличает ранние языки программирования от поздних?

а) степень зависимости от среды реализации (+);
б) программы на ранних языках более наглядны;
в) существенных различий нет.

Вариант 3: в чем состоит особенность языков логического программирования?

а) этот класс языков основан на функциях;
б) моделирует правила (+);
в) является наиболее машинно-независимым.

Вопрос 3.

Вариант 1: какие из перечисленных языков программирования основаны на структурном подходе?

а) C# и ProLog;
б) C# и Fortran (+);
в) C# и SML.

Вариант 2: что отличает объектно-ориентированный подход к программированию от компонентно-ориентированного?

а) степень зависимости от среды реализации;
б) стиль программирования;
в) структура программы (+).

Вариант 3: в чем состоит особенность языков функционального программирования?
а) этот класс языков основан на сценариях;
б) концептуально близок к любой предметной области;
в) легко формализуем математически (+).

1.2. Концепция и возможности подхода .NET

В данном параграфе рассмотрены вопросы, относящиеся к идеологии, технологии и обзору практических возможностей создания программных систем на основе наиболее современного подхода проектирования и реализации программного обеспечения, известного под названием Microsoft .NET.

В отличие от всех предшествующих подходов, компания Microsoft предлагает наиболее развитое и комплексное решение для проектирования и реализации программного обеспечения. Рассмотрим, в частности, такие аспекты .NET, как:

- идеология;
- вычислительная модель;
- технологическая платформа;
- инструментальное решение;
- безопасность;
- интеграция приложений;
- поддержка веб-сервисов.

Прежде всего, необходимо ответить на важный вопрос: что такое .NET? Несмотря на широкое освещение в прессе, ответить однозначно непросто, прежде всего по той причине, что ответ представляется многоаспектным.

Итак, можно сказать, что .NET — это подход к проектированию и реализации программного обеспечения, включающий по меньшей мере четыре следующих компонента:

1) идеология проектирования и реализации программного обеспечения;

2) модель эффективной поддержки жизненного цикла прикладных систем;

3) унифицированная, интегрированная технологическая платформа для программирования;

4) современный, удобный в использовании, безопасный инструментарий для создания, размещения и поддержки программного обеспечения.

Остановимся подробнее на каждом из этих существенных аспектов. Прежде всего, постараемся сформировать понимание идеологии подхода Microsoft .NET.

Самой корпорацией-разработчиком сформулированы приблизительно следующие важнейшие аспекты видения (vision) идеологии .NET:

1) легкость развертывания приложений в глобальной среде Интернет;

2) экономичная разработка программного обеспечения;

3) «бесшовная», гибкая интеграция программных продуктов и аппаратных ресурсов;

4) предоставление программного обеспечения как сервиса;

5) новый уровень безопасности и удобства использования.

Действительно, все аспекты видения .NET удалось реализовать на качественно новом уровне, обеспечив существенное продвижение вперед в направлении гибкости интеграции с программно-аппаратными ресурсами, безопасности и удобстве использования кода, а также снижении затрат на производство программного обеспечения.

Рассмотрим подробнее, как идеология .NET претворяется в практические вопросы проектирования программного обеспечения.

Корпорацией Microsoft предложен новаторский компонентно-ориентированный подход к проектированию, который является развитием объектно-ориентированного направления. Согласно этому подходу интеграция объектов (возможно, гетерогенной природы) производится на основе интерфейсов, представляющих эти объекты (или фрагменты программ) как независимые компоненты. Такой подход существенно облегчает написание и взаимодействие программных «молекул»-компонент в гетерогенной среде проектирования и реализации. Стандартизируется хранение и повторное использование компонент программного проекта в условиях распределенной сетевой среды вычислений, где различные компьютеры и пользователи обмениваются информацией, например, взаимодействуя в рамках исследовательского или бизнес-проекта.

Существенным преимуществом является и возможность практической реализации принципа «всякая сущность является объектом гетерогенной программной среды». Во многом это стало возможным благодаря усовершенствованной, обобщенной системе типизации Common Type System, или CTS, которая подробнее рассмотрена ниже.

Строгая иерархичность организации пространств для типов, классов и имен сущностей программы позволяет стандартизировать и унифицировать реализацию. Новый подход к интеграции компонент приложений в среде вычислений Интернет (или так называемые веб-сервисы) дает возможность ускоренного создания приложений для глобальной аудитории пользователей. Универсальный интерфейс .NET Framework обеспечивает интегрированное проектирование и реализацию компонент приложений, разработанных согласно различным подходам к программированию.

Говоря о .NET как о технологической платформе, нельзя не отметить тот факт, что она обеспечивает одновременную поддержку проектирования и реализации программного обеспечения с использованием различных языков программирования. При этом поддерживаются десятки языков программирования, начиная от самых первых (в частности, COBOL и FORTRAN) и заканчивая самыми современными (например, C#, F# и Visual Basic). Ранние языки программирования до сих пор активно используются, в частности, для обеспечения совместимости с ранее созданными приложениями, критичными для бизнеса (скажем, COBOL весьма широко использовался для создания прикладных программ, поддерживающих финансовую деятельность).

Применение технологии веб-сервисов — это не просто дань моде, а реальная (и, пожалуй, наиболее приемлемая практически возможность) обеспечения масштабируемости и интероперабельности приложений. Под масштабируемостью понимают возможность плавного роста времени ответа программной системы на запрос с ростом числа одновременно работающих пользователей; в случае веб-сервисов масштабируемость реализуется посредством распределения вычислительных ресурсов между сервером, на котором выполняется прикладная программа (или хранятся данные) и компьютером пользователя.

Под интероперабельностью понимается возможность интегрированной обработки гетерогенных данных, поступающих от разнородных прикладных программ. Именно благодаря интероперабельности возможна унификация взаимодействия пользователей через приложение с операционной системой на основе специализированного интерфейса прикладных программ, или API-интерфейса (Application Programming Interface).

Важно отметить и то обстоятельство, что технология .NET не только востребована мировой общественностью, но и официально признана, что отражено в соответствующих стандартах ECMA (European Computer Manufacturers Association).

Далее рассмотрим инструментальные возможности .NET как средства проектирования и реализации программного обеспечения, т.е. собственно программирования в широком смысле этого слова.

Прежде всего, необходимо отметить поддержку многоязыковой среды разработки приложений CLR (Common Language Runtime). Эта возможность появилась благодаря универсальному межъязыковому интерфейсу Common Language Infrastructure, или CLI, который поддерживает разработку программных компонент на различных языках программирования. При этом несомненным преимуществом для программистов является то обстоятельство, что они могут разрабатывать (или дорабатывать) программное обеспечение на наиболее подходящем языке программирования. Здесь следует учитывать характер задачи (скажем, рекурсия или символьная обработка прозрачнее и с меньшими трудозатратами реализуема на языке функционального программирования, а формализация структуры предметной области — на объектно-ориентированном языке). Кроме того, необходимо принимать во внимание опыт работы программистов в команде разработчиков и тот язык программирования, на котором изначально создавалась система.

Отметим еще два существенных обстоятельства. Во-первых, основные сервисные возможности для разработчиков, которые предоставляет среда .NET, (отладка, анализ кода и т.д.) не зависят от конкретного языка программирования, и, следовательно, программистам нет необходимости заново постигать особенности среды разработки, если необходимо перейти с одного языка на другой. Во-вторых, несмотря на то, что еще не все языки программирования поддерживаются .NET, существует возможность самостоятельной разработки транслятора для любого языка программирования, причем его реализация не вызывает трудностей даже у программистов,

практически не имеющих профессиональной подготовки в области разработки компиляторов.

Важнейшим элементом любой идеологии, технологии и инструментального средства программирования в настоящее время является безопасность. Данное утверждение неоспоримо, если принять во внимание тот факт, что многие важнейшие системы жизнеобеспечения и оборонной отрасли управляются автоматизированным образом, т.е. с помощью компьютеров. В связи с этим .NET как инструментальное средство призвано обеспечивать уровень безопасности, отвечающий современным требованиям, для чего в .NET реализована, в частности, такая мера безопасности, как автоматизированное управление жизненным циклом программного обеспечения. Для программиста это проявляется, например, в автоматической реализации процедуры «сборки мусора», а также в запрете на использование указателей на области памяти с неопределенным значением («висячих» ссылок) и самоссылающихся указателей (циклических ссылок).

Более существенным ограничением безопасности является автоматизация обеспечения синтаксической коррекции кода. Это достигается посредством безопасных вызовов функций и процедур, контроля выхода за границы заявленного программистом размера статически распределяемых областей памяти, а также запрета использования переменных без задания им значения по умолчанию (инициализации).

Еще одним важным аспектом комплексного обеспечения безопасности в .NET является обязательная проверка промежуточного кода (IL — Intermediate Language) на корректность типизации, которая осуществляется в рамках реализованной стратегии расширенного контроля соответствия типов.

Существенное нововведение добавлено и к правам доступа пользователей к ресурсам. В частности, для включения компонента в проект необходимо проверить источник кода, заверенный автором цифровой подписью, и убедиться в подлинности отправителя.

Гибкое и надежное ограничение доступа пользователей к ресурсам осуществляется также благодаря широкому спектру динамически корректируемых в соответствии с профилями пользователя политик доступа.

Немаловажным пунктом для обеспечения безопасности являются и криптографические методы, которые необходимы для шифрования конфиденциальной или коммерческой информации, передаваемой, например, по интернет-каналам.

Рассмотрим поддержку жизненного цикла программного обеспечения в рамках подхода .NET. Для установки на компьютеры пользователей ранее созданного прикладного программного обеспечения создаются инсталляционные комплекты в форме так называемых сборок. *Сборкой* называется множество модулей, необходимых для осуществления инсталляции программного обеспечения. Она характеризуется уникальностью, которая обеспечивается идентификатором версии сборки и цифровой подписью автора. Сборка является самодостаточной единицей для установки программного обеспечения и не требует никаких дополнений. Возможно