Эволюция технологий программирования

Кудрина Е.В.[1], Огнева М.В.[2]
kudrinaev@mail.ru[1], ognevamv@gmail.com[2]
Саратовский государственный университет имени Н.Г. Чернышевского

С момента своего появления языки программирования динамически развиваются, появляются новые языки, предоставляя программистам все более мощные возможности разработки качественного и надежного программного обеспечения. Параллельно с развитием языков программирования идет и эволюционирование технологий разработки программного обеспечения. В данной статье будет представлена эволюция технологий программирования в контексте развития языков программирования от низкоуровневых до современных высокоуровневых языков.

Ключевые слова: программирование, языки программирования, технологии программирования.

С момента своего появления языки программирования динамически развиваются, появляются новые языки, предоставляя программистам все более мощные возможности разработки качественного и надежного программного обеспечения. Параллельно с развитием языков программирования идет и эволюционирование технологий разработки программного обеспечения.

В данной статье мы рассмотрим эволюцию технологий программирования в контексте развития языков программирования от низкоуровневых до современных высокоуровневых языков.

На каком бы языке программирования не были написаны программы, они выполняются компьютером, поэтому в окончательном виде любая программа представляет собой набор инструкций процессора. Однако пишут программы люди, поэтому практически все языки программирования предоставляют возможность более удобной для человека записи данного набора инструкций, что облегчает написание, отладку и последующую модификацию программ.

Первые программы писались на машинном языке и представляли собой двоичную или шестнадцатеричную запись команд. Потом стали использовать так называемые мнемокоды (вместо двоичной последовательности для каждой команды стало использоваться специальное обозначение). Для этой цели был изобретен язык Assembler [1], который позволил писать более длинные программы. Assembler относится к языкам низкого уровня, но не потому, что программы, разработанные на данном языке «низкого» качества, а потому, что для написания самой простой программы программисту требовалось знать команды конкретного типа процессора и напрямую обращаться к данным, размещенным в его регистрах. Для перевода в машинный код программы, написанной на языке Assembler, стал использоваться транслятор. Появления языка Assembler и транслятора к нему существенно облегчило процесс разработки программ, однако со временем проявился существенный недостаток – отсутствие переносимости программ, написанных на языке Assembler, на компьютеры с другой архитектурой и системой команд.

Развитие вычислительной техники вело к быстрой смене типов и моделей процессоров, поэтому стало необходимо обеспечивать не только удобство в разработки программ, но и аппаратную переносимость программ. Это привело к появлению языков программирования высокого уровня, первым из которых был Fortran [2]. Его существенным отличием стало то, что программа разрабатывалась на языке, «схожем» с естественным языком, в котором появились смысловые конструкции, описывающие структуры данных различной сложности и операции над ними. Также для каждой архитектуры ЭВМ стали разрабатываться платформенно-уникальные трансляторы. Теперь, чтобы перенести программу с одной аппаратной платформы на другую, следовало «перетранслировать» исходный код программы с помощью соответствующей версии транслятора.

Появление высокоуровневых языков программирования и трансляторов к ним не только решило проблему переносимости программ, но и дало толчок к развитию технологии структурного программирования.

Первым шагом в развитии структурного программирования стало использование подпрограмм. Использование подпрограмм позволяет после их разработки и отладки отвлечься от деталей реализации. Для вызова подпрограммы требуется знать только ее интерфейс, который, если не используются глобальные переменные, полностью определяется заголовком подпрограммы.

Вторым шагом стала возможность создания собственных типов данных, позволяющих структурировать и группировать информацию, представлять ее в более естественном виде. Естественно, что для работы с собственными типами данных разрабатываются специальные подпрограммы.

Объединение в модули описаний собственных типов данных и подпрограмм для работы с ними стало следующим шагом в развитии структурного программирования. Создаваемые модули стали помещаться в библиотеку подпрограмм. Таким образом, во все языках программирования, в том числе и в языке C++[3,4], появились обширные библиотеки стандартных подпрограмм. Следует отметить, что каждый модуль стандартной библиотеки реализует некоторую функциональность (возможность) языка, которая может использоваться другими программистами. Если в языке не хватает той или иной функциональности, создается очередной модуль и помещается в библиотеку подпрограмм.

Использование технологий структурного программирования и готовых библиотек позволило успешно создавать достаточно крупные проекты, а также уменьшить время разработки проектов и облегчить возможность их модификации. Однако сложность программного обеспечения продолжала возрастать, и требовались все более сложные средства ее преодоления. Идеи структурного программирования получили свое дальнейшее развитие в объектно-ориентированном программировании – технологии, позволяющей достичь простоты структуры и управляемости очень крупных программных систем.

Технология объектно-ориентированного программирования использует идею объединения данных и подпрограмм (функций, методов) для их обработки в специальные конструкции, получившие название объектов. Программные объекты отражают строение объектов реального мира, поэтому в качестве таких объектов могут выступать люди (сотрудники, студенты, читатели библиотеки, вкладчики банков), хранилища данных (словари, списки товаров, каталоги книг), типы данных (комплексные числа, точки на плоскости, время), структуры данных (массивы, списки, деревья, графы) и многое другое. Программа, разработанная на основе технологии объектно-ориентированного программирования, представляет собой совокупность объектов, которые взаимодействуют между собой посредством вызова функций/методов друг друга. Технология объектно-ориентированного программирования позволяет упростить процесс написания программ за счет использования таких важных принципов как инкапсуляция, наследование и полиморфизм.

Объектно-ориентированное программирование основывается на понятии «класс». С точки зрения компилятора класс является типом данных, определяемым пользователем. Содержательно класс объединяет в себе данные и функции/методы для их обработки, позволяя скрывать ту часть информации, которую не нужно видеть пользователю. В этом случае говорят, что данные и функции/методы инкапсулированы. Инкапсуляция повышает степень абстракции программы – для использования в программе какого-то класса не требуется знаний о его реализации, достаточно знать только его интерфейс (описание). Это позволяет изменить реализацию класса, не затрагивая саму программу, при условии, что интерфейс класса останется прежним.

Конкретные величины типа данных «класс» называются экземплярами класса или объектами. Класс содержит члены-данные (поля) и члены-функции (методы класса). Например, мы можем определить класс «сотрудник», в котором описать такие поля как «фамилия», «имя», «отчество», «дата рождения», «оклад», «стаж» и многое другое, а также такие функции/методы как «показать данные о сотруднике», «рассчитать зарплату», «рассчитать отпускные», «начислить премию» и т.д. Объекты класса «сотрудник» – это реальные люди, например, Иванов Иван Иванович, Алексеева Татьяна Сергеевна, Данилов Павел Александрович. У всех объектов значения полей данных будут различными, а вот функции/методы по их обработке – одинаковыми, и это естественно, так как существует единое правило (алгоритм) расчета зарплаты, отпускных и премиальных. Если изменится, например, правило расчета зарплаты, то достаточно будет изменить соответствующую функцию/метод в классе «сотрудник», и это изменение отразится на всех объектах.

В повседневной жизни мы часто сталкиваемся с разбиением классов на подклассы. Например, класс «наземный транспорт» содержит такие подклассы, как «легковые автомобили», «грузовые автомобили», «общественный транспорт» и т.д. Каждый подкласс обладает свойствами того класса, из которого он выделен. Кроме этого, каждый подкласс обладает и

собственными свойствами. Так, например, и легковые, и грузовые автомобили обладают колесами и мотором и способны передвигаться по земле. Это – свойства, присущие классу «наземный транспорт». В то же время подкласс «легковые автомобили» содержит небольшое количество посадочных мест для перевозки пассажиров, а подкласс «грузовые автомобили» – кузов для перевозки грузов.

Деление классов на подклассы лежит в основе принципа наследования. Наследование в объектно-ориентированном программировании – это возможность создания иерархии классов, когда потомки (производные классы) наследуют все свойства своих предков (базовых классов), могут их изменять и добавлять новые. При этом свойства повторно не описываются, что сокращает объем программы. Иерархия классов представляется в виде древовидной структуры, в которой более общие классы располагаются ближе к корню, а более специализированные – ближе к листьям.

Еще один важный принцип, заложенный в основу объектно-ориентированного программирования, – это полиморфизм. Полиморфизм – это возможность использовать в рамках одного класса или различных классов одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы. Частным случаем полиморфизма является перегрузка функций/методов.

Перечислим основные преимущества объектно-ориентированного программирования [3-7]:

– *Логичность*. Человек мыслит не терминами команд, переменных и операций, а терминами объектов, их характеристик и возможностей. С этой точки зрения объектно-ориентированное программирование идеально соответствует образу человеческого мышления.

– *Повторное использование кода*. Программный код, написанный в концепциях объектно-ориентированного программирования, достаточно хорошо подходит для повторного использования (как непосредственного, так и с некоторыми переработками).

– *Высокая скорость разработки*. Данный показатель обусловливается как упрощением процесса написания кода, так и возможностью повторного использования ранее написанного кода. В идеале, при наличии соответствующих библиотек, приложение не пишется, а собирается из готовых кусков с написанием незначительных по объему «связок».

Вместе с тем у объектно-ориентированного программирования есть и недостатки [3-7]:

– *Бóльшие, по сравнению с остальными технологиями программирования, временны́е затраты на проектирование*. Это тормозит начальную фазу разработки, но окупается на последующих этапах разработки программного обеспечения.

– *Снижение скорости работы приложения*. Поддержка технологии объектно-ориентированного программирования требует от среды исполнения некоторых затрат, как по времени работы, так и по объему занимаемой оперативной памяти.

Одним из языков программирования, целиком функционирующим на принципах объектно-ориентированного программирования является язык C#, разработанный в 2000 году компания Microsoft. Появление нового языка программирования стала частью более значительного события – разработка платформы .NET (.NET Framework). Платформа .NET по сути представляла собой новую модель создания приложений, которая включает в себя следующие возможности [5-8]:

– использование библиотеки базовых классов, предлагающих целостную объектно-ориентированную модель программирования для всех языков программирования, поддерживающих .NET;

– полное и абсолютное межъязыковое взаимодействие, позволяющее разрабатывать фрагменты одного и того же проекта на различных языках программирования;

– общая среда выполнения приложений .NET, независимо от того, на каких языках программирования для данной платформы они были созданы; при этом среда берет на себя контроль за безопасностью выполнения приложений и управление ресурсами;

– упрощенный процесс развертывания приложения, в результате чего установка приложения может свестись к простому копированию файлов приложения в определенный каталог.

Основными языками программирования, изначально предназначенными для платформы .NET, являлись C#, VB.NET, Managed C++ и JScript .NET. Для данных языков Microsoft предлагает собственные компиляторы, переводящие программу в специальный код, называемый IL-кодом (Intermediate Language – промежуточный язык), который выполняется средой CLR (Common Language Runtime – единая среда выполнения программ, входящая в состав платформы .NET). Другими словами, компиляция со всех языков программирования .NET, включая C#, происходит в IL-код, который не является специфическим ни для какой операционной системы и ни для какого языка программирования. Он может быть выполнен в любой операционной системе JIT-компилятором (Just In Time compiler – компилирование точно к нужному моменту) при условии, что в операционную систему встроена среда CLR. Важно, что в процессе выполнения программы среда CLR берет на себя всю низкоуровневую работу, например, автоматическое управление памятью.

Популярность платформы .Net привела к появлению новых языков программирования таких как A#, F#, Cobra, J#. Вместе с Microsoft еще несколько компаний и академических организаций стали создали свои собственные компиляторы для уже существующих языков, генерирующие IL-код, работающий в CLR. На сегодняшний момент известны компиляторы для

языков Pascal, Cobol, Lisp, Perl, Prolog и т.д. Это означает, что можно написать программу, например, на языке Pascal, а затем, воспользовавшись соответствующим компилятором, создать приложение, который будет работать под управлением среды CLR.

Эволюция технологий программирования не остановилась на объектно-ориентированном программировании. С развитием Windows-технологий связано появление компонентного подхода в программировании [9], с развитием Internet-технологий – появление технологий Web-программирования [10], с развитием мобильных устройств и гаджетов – появление технологий разработки мобильных приложений [11]. Одним из современных достижений в области программирования стало развитие технологий параллельного программирования [12]. Однако неоспоримым остается то, что именно объектно-ориентированное программирование дало толчок к появлению этих новых технологий программирования.

В заключение следует отметить, что включение в образовательные программы подготовки студентов IT-направлений дисциплин, связанных изучением современных языков и технологий программирования, таких как «Машинно-зависимые языки программирования», «Объектно-ориентированное программирование», «Разработка приложений для ADO.Net», «Технологии программирования», «Разработка web-приложений», «Разработка мобильных приложений», «Параллельное и распределенное программирование», позволит сформировать готовность выпускников к осуществлению профессиональной деятельности в области разработки программного обеспечения.

Список литературы

[1] *Титовский С.Н.* Языки программирования. Ассемблер: учебное пособие/ С.Н. Титовский, Н.В. Титовская. – Красноярск: ИПК СФУ, 2008 – 125 с.

[2] *Горелик А.М.* Эволюция языка программирования Фортран (1957-2007) и перспективы его развития// Вычислительные методы программирования: новые вычислительные технологии. – 2008. – Т.9 – № 2. – С. 53-71.

[3] *Огнева М.В.* Структуры данных и алгоритмы: программирование на языке С++: Учеб. пособие в 2 ч. Часть 1/ М.В. Огнева, Е.В.Кудрина – Саратов: Издательство «Наука», 2013 – 100 с.

[4] *Огнева М.В.* Программирование на языке С++: практический курс: учебное пособие для вузов / М. В. Огнева, Е. В. Кудрина. – М. : Издательство Юрайт, 2017. – 335 с.

[5] *Кудрина Е.В.* Программирование в среде VISUAL STUDIO.NET: разработка приложений на языке С#: учебное пособие/ Е.В. Кудрина, М.В. Огнева. – Саратов: ООО Издательство «Кубик», 2010. – 546 с.

[6] *Агуров П.В.* Модуль 1. Основы программирования на язык С# [Электронный ресурс]/ П.В. Агуров, А.В. Кузнецов, Е.В. Кудрина, М.В. Огнева //Портал проекта «Твой курс: IT для молодежи». – Режим доступа: https://www.it4youth.ru/shared/attach_get.php?id=65669&key=f3d94b2baffc7957941cb4a20e7af724

[7] *Агуров П.В.* Модуль 2. Основы объектно-ориентированного программирования на язык С# [Электронный ресурс]/ П.В. Агуров, А.В. Кузнецов, Е.В. Кудрина, М.В. Огнева //Портал проекта «Твой курс: IT для молодежи». – Режим доступа: https://www.it4youth.ru/shared/attach_get.php?id=65671&key=bad7314623005015aee9bee9a35cffce

[8] *Рихтер Дж.* CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. – СПб.: Питер, 2013. – 896 с.
[9] *Кулямин В.В.* Технологии программирования. Компонентный подход: учебное пособие. – М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 463 с.
[10] *Глибовец Н.Н.* Эволюция принципов и средств веб-программирования/ Н.Н. Глибовец, С.С. Гороховский, И.В. Коваль, А.Н. Корень// Управляющие системы и машины. – 2012. – № 1. – С. 49-54.
[11] *Соколова В.В.* Разработка мобильных приложений: учебное пособие. – Томск: Томский политехнический университет, 2014. – 176 с.
[12] *Гергель В.П.* Современные языки и технологии параллельного программирования: учебник для студентов высших учебных заведений. – М.: Изд-во Московского ун-та, 2012. – 408 с.