УДК 004.4

НЕКОТОРЫЕ АСПЕКТЫ, СВЯЗАННЫЕ С РАЗВИТИЕМ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

[1]Шкатова Г.И., [1,2]Берестнева О.Г.

[1] ФГБОУ ВПО «Национальный исследовательский Томский политехнический университет», Томск, Россия (634050, г. Томск, пр. Ленина, 30), e-mail: ogb@tpu.ru
[2]ГБОУ ВПО «Сибирский государственный медицинский университет» Томск, Россия (634050, г. Томск, Московский тракт, 2), e–mail: office@ssmu.ru

Как известно, языки программирования являются средством представления знаний для компьютерных систем. В статье представлен обзор языков программирования, концептуальные идеи которых оставили свой след в развитии программирования и используются в современных языках в настоящее время. Представлены основные характеристики и свойства языков программирования. Концептуальные идеи языков программирования, описанные способы реализации в них семантических структур, а также свойства, характеризующие язык, способствуют выбору наилучшего в плане решения той или иной задачи. Поскольку объектно-ориентированное программирование представляет одну из важных парадигм современного программирования, рассматривается один из аспектов, связанных с разработкой классов. В статье также рассматриваются аспекты, которые следует учитывать при разработке концептуальной модели функционирования пользовательского интерфейса.

Ключевые слова: объектно-ориентированное программирование, языки программирования, семантические структуры.

ADAPTATION SOME ASPECTS RELATED TO THE DEVELOPMENT OF PROGRAMMING LANGUAGES

[1]Shkatova G.I., [1,2]Berestneva O.G.

[1] National Research Tomsk Polytechnic University, Tomsk, Russia (634050, Tomsk, Lenin avenue, 30), e-mail: ogb@tpu.ru
[2] Siberian State Medical University, Tomsk, Russia (634050, Tomsk, Moscow highway,2),e–mail: office@ssmu.ru

As you know, programming languages are a means of knowledge representation for computer systems. The article presents an overview of programming languages, conceptual ideas which have left their mark on the development of programming and used in modern languages at the moment. The main characteristics and properties of programming languages. Conceptual ideas of programming languages, described how to implement them in the semantic structures, as well as properties that characterize the language contribute to the choice of the best in terms of solving a particular problem. As object-oriented programming is one of the most important paradigms of modern programming is considered one of the aspects related to the development of classes. The article also discusses the aspects that should be considered when developing a conceptual model of the functioning of the user interface.

Keywords: object-oriented programming, programming languages, semantic structures.

Языки программирования являются средством представления знаний для компьютерных систем. Они предлагают концептуальные средства представления и возможности моделирования, приспособленные к решению конкретных задач [3]. При этом концепции языков программирования складываются и развиваются в результате стремления разработчиков снизить «семантический разрыв» между языком описания работы вычислительного устройства и языком, на котором осуществляется постановка задачи. Развитие языков на эмпирическом уровне определяется развитием вычислительной техники. На теоретическом уровне изменения в представлениях о языках программирования определяется выбором формы управления вычислительными устройствами. Многообразие концепций языков, разработанных за период

в 60 лет, привело к многообразию парадигм программирования, сложившихся к настоящему времени [3].

Концептуальные идеи языков программирования, которые нашли свое отражение в современных языках программирования, сыграли важную роль в теории программирования. А знание способов реализации в них семантических структур является важным фактором при выборе языка программирования для решения поставленной задачи.

Характеристики и свойства языков программирования

Основными характеристиками, позволяющими сравнивать языки программирования и выбирать наилучшие для решения той или иной задачи, являются: мощность, уровень и концептуальная целостность.

Мощность языка характеризуется количеством и разнообразием задач, алгоритмы, решения которых можно записать, используя этот язык. Очевидно, самым мощным является машинный язык. Любую задачу, запрограммированную на каком-либо языке, можно запрограммировать и на машинном языке.

Уровень языка характеризуется сложностью решения задач с помощью этого языка. Чем проще записывается решение задач, чем более непосредственно реализуются сложные операции и понятия, чем меньше объем получаемых программ, тем выше уровень языка.

Концептуальная целостность языка, в свою очередь, характеризуется совокупностью понятий, служащих для описания этого языка, и включает три взаимосвязанных аспекта: экономию, ортогональность и единообразие понятий. Экономия понятий предполагает достижение максимальной мощности языка с помощью минимального числа понятий. Ортогональность понятий означает, что между понятиями не должно быть взаимного влияния. Так, если какое-либо понятие используется в различных контекстах, то правила использования должны быть одни и те же. Единообразие понятий требует согласованного, единого подхода к описанию и использованию всех понятий.

Обычно чем меньше мощность языка (то есть чем уже область его применения), тем выше его уровень. По этой причине наряду с универсальными языками разрабатываются и специализированные языки в некоторой конкретной области. Конечно, чем мощнее язык, тем труднее обеспечить концептуальную целостность; в то же время высокий уровень языка непосредственно связан с концептуальной целостностью.

Перечисленные характеристики языков программирования определяют наличие или отсутствие свойств: надежности, удобочитаемости, полноты, гибкости, простоты. Эти свойства позволяют наиболее детально сравнивать языки.

Надежность языка обеспечивает минимум ошибок при написании программ. Для этого язык должен быть таким, чтобы было трудно делать ошибки, не обнаруживаемые при

компиляции. Например, благодаря наличию в языке требования, чтобы все переменные были объявлены до использования, ошибки, связанные с неправильным написанием имен, выявляются при компиляции (то есть автоматически). Язык должен защищать программиста от него самого, сделав трудным или даже невозможным появление некоторых ошибок. Так, плохо, если можно сделать одно и то же более чем одним способом: лишняя возможность выбора может привести к ошибке. Примером более тонкой защиты программиста является трактовка отношения равенства: поскольку точное равенство двух чисел с плавающей точкой есть не что иное, как удачное совпадение, то разумно определить его как приблизительное равенство с некоторой точностью.

Удобочитаемость языка — это свойство, обеспечивающее легкость восприятия программ человеком. Удобочитаемость зависит от широкого спектра факторов, включающего, с одной стороны, выбор ключевых слов, а с другой — возможность модулеризации программы. Главное, чтобы нотация языка позволяла при чтении программы легко выделять основные понятия каждой конкретной части программы, не обращаясь к сопровождающим ее описаниям.

Высокая степень удобочитаемости оказывается полезной с различных точек зрения. Во-первых, уменьшается сложность документирования, если центральным элементом документации является сама программа. Во-вторых, удобочитаемость позволяет легче сопровождать программу: ясно, что существенные изменения в программе могут быть сделаны лишь тогда, когда ее работа понимается совершенно правильно.

Очевидно, что реализация требований удобочитаемости зависит от самого программиста, который должен постараться по возможности четче структурировать свою программу и так расположить ее текст, чтобы подчеркнуть эту структуру. Тем не менее важную роль играет и используемый программистом язык. На нижних уровнях программных конструкций язык должен обеспечить возможность четкой спецификации того, какие объекты данных подвергаются обработке и как они используются. Эта информация определяется выбором идентификаторов и спецификаций типов данных.

Программиста не нужно заставлять прибегать к искусственным построениям, вводя в язык такие ограничения, как максимальная длина идентификатора или определенные фиксированные типы данных. Алгоритмические структуры должны выражаться в терминах легко понимаемых структур управления. Ключевые слова не следует сводить к аббревиатурам, а символы операций должны быть выбраны так, чтобы они отображали их смысл. На более высоких уровнях программных конструкций язык должен обеспечивать возможность модулеризации. Общее поведение программы гораздо легче понять, когда она составлена из ряда

автономных единиц, каждая из которых должна быть понятна вне связи с остальными частями программы.

Полнота языка обеспечивает описание на языке решения задач определенной предметной области, а также с помощью средств языка, например средств отладки, процесса разработки программ.

Гибкость языка обеспечивает легкость выражения на языке необходимых для решения задач действий, предоставляет программисту достаточно возможностей для выражения всех операций в программе.

Простота языка обеспечивает легкость понимания семантики языковых конструкций и запоминания их синтаксиса. А это в свою очередь позволяет уменьшить затраты на обучение программиста и вероятность совершения ошибок, возникающих в результате неправильной интерпретации спецификаций языка. Например, язык должен быть таким, чтобы конструкции, означающие близкие по смыслу понятия, и выглядели одинаково, и, что еще более важно, конструкции, означающие различные понятия, должны выглядеть по-разному.

При сравнении и выборе языков программирования следует учитывать еще два свойства языка, хотя они и не влияют непосредственно на процесс разработки программ. Это мобильность и эффективность.

Мобильность языка обеспечивает независимость его аппаратных средств, позволяет переносить программное обеспечение с машины на машину с относительной легкостью.

Эффективность языка обеспечивает эффективную реализацию языка (включая эффективную реализацию компилятора и эффективные программы, генерируемые компилятором). В настоящее время в связи с все снижающейся стоимостью аппаратных средств и все возрастающей стоимостью программного обеспечения необходимость эффективности отходит на второй план по сравнению с надежностью.

Эффективность создания, тестирования и использования программы хорошо иллюстрируется языком АПЛ. При использовании этого языка для решения некоторого класса задач проектирование, кодирование, тестирование, модификация и выполнение программы отнимает у программиста минимальное количество времени и энергии. АПЛ может быть назван эффективным в полном смысле слова — он позволяет минимизировать суммарное время и энергию, затрачиваемые на решение задач на ЭВМ.

Описание свойства языков можно еще более детализировать, указав их зависимость от тех или иных частных требований к языкам. Совокупность этих требований уже довольно велика, что затрудняет их прямое использование для сравнения и выбора языков. Именно поэтому целесообразно остановиться на перечисленных выше трех характеристиках и семи

свойствах. Однако для адекватного оценивания характеристик и свойств необходимо знать их зависимость от многочисленных требований.

Некоторые аспекты объектно-ориентированного программирования

Если приемы процедурного программирования концентрируются на алгоритмах, то объектно-ориентированное программирование (ООП) концентрируется на сути задачи. Элементы программы разрабатываются в соответствии с объектами, присутствующими в описании задачи. Отсюда и терминология – «Объектно-ориентированное программирование». При этом первичными считаются объекты (данные), которые могут активно взаимодействовать друг с другом с помощью механизма передачи сообщений (называемого также и механизмом вызова методов). Функция программиста - определить объекты, взаимодействие которых после старта программы приведет к достижению необходимого конечного результата.

Общий подход к ООП включает в себя следующие концепции [6]: 1) наличие типов, определенных пользователем; 2) скрытие деталей реализации (инкапсуляция); 3) использование кода через наследование; разрешение интерпретации вызова функции во время выполнения программы (полиморфизм).

Известно, что реальные/физические объекты окружающего мира обладают тремя базовыми характеристиками:

1) набором свойств, характеризующих объект;
2) способностью объекта изменять свойства;
3) способностью реагировать на события как в окружающем мире, так и внутри самого себя.

Хотя термины «тип данных» (или просто тип) и «абстрактный тип данных» звучат похоже, они имеют различный смысл. В языках программирования тип данных (переменной) обозначает множество значений, которые может принимать эта переменная. Абстрактный тип данных (АТД) определяется как *математическая модель с совокупностью операторов, определенных в рамках этой модели*. Таким образом, классы могут служить простым примером АТД.

Разработку класса – абстрактного типа данных рекомендуется начинать с разработки информационной и математической моделей реального или физического объекта, как это показано в схеме, представленной на рис. 1.

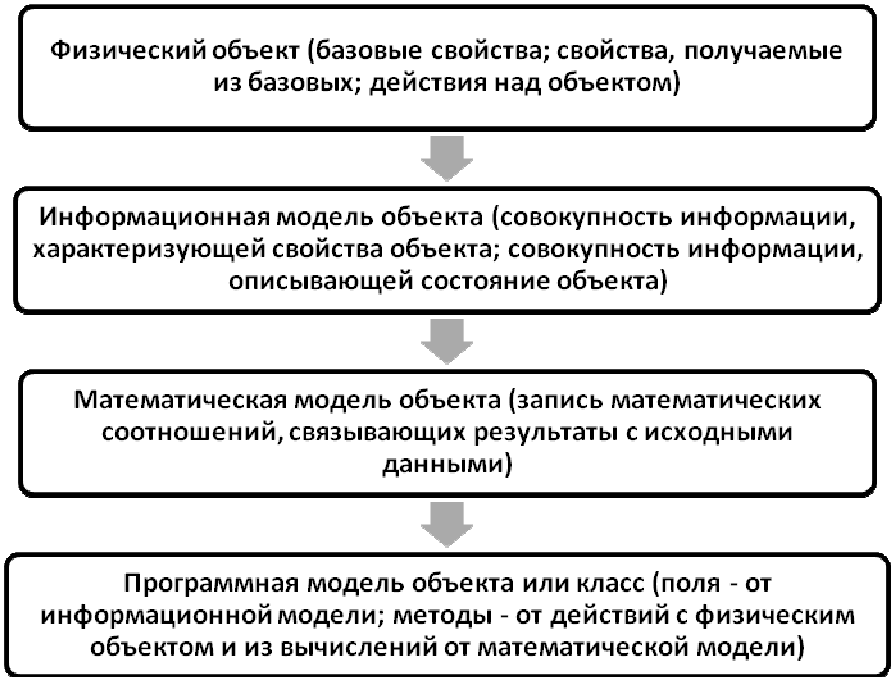

Рис. 1. Схема процесса создания программной модели (класса) в рамках ООП.

В соответствии с этой схемой на начальном этапе исследования задачи нужно выделить физические объекты. После чего нужно определить свойства, характеризующие каждый физический/реальный объект. Следует отметить, что из множества свойств объекта следует взять только те, которые необходимы для решения задачи. Во избежание избыточности следует выяснить: какие из свойств являются базовыми; какие свойства можно получить на основании базовых свойств, какие свойства будут меняться; как воздействовать на объект с целью изменения его свойств; в каких процессах будет задействован объект и как это отразится на его свойствах (действия над объектом).

Следующим шагом в исследовании задачи будет построение информационной модели объекта. Здесь требуется дать названия свойствам, установить диапазоны возможных значений свойств и определить их типы. Выяснить, что считать исходными данными и что считать результатами. Определить возможные места расположения данных. Если эти данные расположены или в результате решения будут располагаться на внешних носителях, например в виде файлов, нужно описать структуру, в которой они представлены или будут представляться на этом носителе.

Построение математической модели сводится к записи математических соотношений, связывающих результаты или новые свойства с исходными данными. Эти соотношения, как правило, представлены в аналитическом виде формулами.

Информационная и математическая модели служат основой для построения программной модели объекта или класса.

Построение класса начинается с разработки его интерфейса. Поля класса формируются из информационной модели. А в методы класса, помимо стандартных методов (конструкторов и деструктора), включаются методы, связанные с изменением свойств объекта, получением новых свойств как воздействием на объект, так и на основании соотношений, заявленных в математической модели.

Та последовательность действий, которая представлена на рис. 1, исходит из того, что в задаче уже выделены физические объекты. Именно физические объекты определяют интерфейс программы в целом, ее программную модель и в значительной степени влияют на логику решения задачи.

Выделение объектов само по себе не является простой задачей. Для выделения объектов в части анализа и исследования задачи значительное место отводится разработке концептуальной модели функционирования пользовательского интерфейса.

Аспекты, которые следует учитывать при разработке концептуальной модели функционирования пользовательского интерфейса

Важность этапа разработки концептуальной модели пользовательского интерфейса сложно переоценить также из-за возросших требований к качеству программного продукта как на отечественных, так и на западных рынках, т.к. в оценку программного продукта входит качество интерфейса, обеспечивающего взаимодействие пользователя с программой.

В. Головач [2] отмечает четыре основных критерия качества интерфейса, а именно:

1) скорость работы пользователей;
2) количество человеческих ошибок;
3) скорость обучения;
4) субъективное удовлетворение (подразумевается, что соответствие интерфейса задачам пользователя является неотъемлемым свойством интерфейса).

Скорость выполнения работы является важным критерием эффективности интерфейса. Длительность выполнения работы пользователем состоит из:

- длительности восприятия исходной информации;
- длительности интеллектуальной работы (пользователь думает, что он должен сделать);
- длительности физических действий пользователя;
- длительности реакции системы.

Как правило, длительность реакции системы является наименее значимым фактором.

Процесс разработки дизайнов интерфейсов в настоящее время является настолько важным составляющим элементом при создании программного продукта, что он стал профессией отдельных людей. Появились научные разработки в области проектирования пользовательского интерфейса, которые опираются на познавательные процессы человеческого

сознания, т.е. знания из областей когнитивной и инженерной психологии, а также эргономики.

Согласно Дональду Норману, на которого ссылается В. Головач, взаимодействие пользователя с системой (не только компьютерной) состоит из семи шагов:

1) формирование цели действий;
2) определение общей направленности действий;
3) определение конкретных действий;
4) выполнение действий;
5) восприятие нового состояния системы;
6) интерпретация состояния системы;
7) оценка результата.

Из этого списка становится видно, что процесс размышления занимает почти все время, в течение которого пользователь работает с компьютером, во всяком случае шесть из семи этапов полностью заняты умственной деятельностью. Соответственно, повышение скорости этих размышлений приводит к существенному улучшению скорости работы.

К сожалению, существенно повысить скорость собственно мышления пользователей невозможно. Тем не менее уменьшить влияние факторов, замедляющих процесс мышления, вполне возможно. Очевидно, что значение этапа проектирования интерфейса приложения трудно переоценить. На нем закладываются основные концепции системы, ее структуры, а также выделяются объекты для проектирования классов.

Список литературы

1. Богатырев Р. Природа и эволюция сценарных языков // МИР ПК – ДИСК. - 2005. - № 10.
2. Головач В. Дизайн пользовательского интерфейса. - 2002. – Режим доступа: www.uibook.ru.
3. Казакова А.Е. Методологические основания развития языков программирования : диссертация по ВАК 09.00.08. – М., 2008. – Режим доступа: dissertCat.com.
4. Неклюдова С.А., Балса А.Р. Парадигмы программирования как инструменты разработчика программных систем // Информационные технологии и системы : межвузовский сборник научных трудов. Выпуск 1 (12). – СПб., 2014.
5. Теслер Г.С. Новая кибернетика. – Киев : Логос, 2004. - Режим доступа: immsp.kiev.ua
6. Эккель Б. Философия C++. Введение в стандартный C++. - 2-е изд. – СПб. : Питер, 2004. – 572 с.

Рецензенты:

Иванкина Л.И., д.ф.н., профессор кафедры менеджмента, Институт социально-гуманитарных технологий Национального исследовательского Томского политехнического университета, г.Томск.

Романенко С.В., д.х.н., профессор, заведующий кафедрой экологии и безопасности жизнедеятельности, Институт природных ресурсов Национального исследовательского Томского политехнического университета, г.Томск.