ОБУЧЕНИЕ СТУДЕНТОВ ОСНОВАМ ОБЪЕКТНО-ОРИЕНТРОВАННОГО ПРОЕКТИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА UML И ПАКЕТА BLUEJ

Петров А.Н.
Московский государственный областной университет
Москва, Россия

Разработка объектно-ориентированных методов проектирования началась в 80-е годы XX века и была связана со значительным расширением сферы применения объектно-ориентированного подхода, который до этого времени использовался в академической сфере и в основном был связан с разработками графического интерфейса пользователя. Изменение подходов к разработке программного обеспечения и рассмотрение сложных взаимосвязей между классами приводило к необходимости использования визуальных языков моделирования.

Первый объектно-ориентированный метод проектирования Booch86 был предложен Гради Бучем (Grady Booch). Появление данного метода способствовало разработке множества различных методов проектирования. Как считал М. Фаулер (M. Fowler), основная причина их создания «состоит в том, что языки программирования не обеспечивают нужный уровень абстракции, способный облегчить процесс проектирования» [3, С. 27].

Перечислим некоторые из ранних методов объектно-ориентированного проектирования: Booch86, GOOD, HOOD, OOSD, JSD, OODLE. Они имели свою сферу применения и различные представления одних и тех же нотаций, что иногда приводило к неоднозначной их интерпретации и, как следствие, к ошибкам. Такое положение дел затрудняло разработку программного обеспечения. Таким образом, появилась потребность в визуальном языке проектирования, который, с одной стороны, имел широкую сферу применения, а, с другой стороны, его нотации, по возможности, были согласованы с другими методами проектирования. Эти требования во многом были реализованы в языке UML, в 1997 году объявленным стандартным языком объектно-ориентированного моделирования.

Рассмотрим некоторые особенности языка UML, которые имеют значение для обучения объектно-ориентированному программированию:

- представление взаимодействий и отношений между классами наглядно;
- абстрагирование от конкретной реализации классов и возможность сконцентрироваться в целом на классах и отношениях между ними;
- управление созданным программным кодом.

Одним из основных преимуществ объектно-ориентированного подхода является возможность повторного использования программного кода. Очень сложно создавать программный код, который действительно является повторно используемым, но затраты на его создание со временем окупятся, так как этот код можно будет применять в большом числе проектов. В повторно используемом коде будут сохраняться уже имеющиеся разработки. Хорошо отлаженный и проверенный программный код при повторном использовании может обеспечить большую надежность программного обеспечения.

«Объектно-ориентированное проектирование в значительной мере способствует решению проблемы повторного использования, возможности сопровождения и обеспечения способности к взаимодействию, поскольку при этом используется три основных приема: проектирование «снизу вверх», инкапсуляция и наследование» [2, С. 77].

Особенностью проектирования «снизу вверх» является то, что создание абстракций представляет собой трудоемкий процесс, на который оказывает влияние множество факторов. Этот процесс нельзя формализовать и составить к нему алгоритм действий. Как правило, в начале процесса поиска абстракций человек рассматривает конкретные объекты, которые имеют отношение к решаемой задаче. Изучение этих объектов, выявление в них как общего, так и различий, позволяет определить некоторые из необходимых классов и сформировать предварительное представление о структуре отношений между классами. Добавление новых классов для решения задачи может привести к изменению имеющихся классов и отношений между ними. На определенном этапе формирования структуры классов могут быть выявлены абстрактные классы. На практике не обязательно ограничиваться использованием только проектирования «снизу вверх», рекомендуется комбинировать его с проектированием «сверху вниз», когда могут быть сразу сформированы наиболее общие абстракции.

В программном коде довольно часто меняются структуры данных, при этом очень важно сохранить их целостность. Для достижения этого необходимо локализовать программный код, который обращается к структурам данных, это позволяет сделать принцип инкапсуляции, который заключается в том, чтобы сокрыть данные, предоставить доступ к данным через public-методы.

Принцип инкапсуляции имеет два преимущества:

1. исключение возможности «случайной» порчи данных;
2. упрощение процесса изменения структуры данных.

В реализации повторного использования программного кода принцип наследования играет особую роль. Он позволяет на основе уже существующих классов создавать новые классы, при

этом сохраняя члены базовых классов и добавляя к ним новые атрибуты и операции.

Для большинства студентов проектирование является новым видом деятельности, поэтому мы рекомендуем проводить специальную подготовку для обучения объектно-ориентированному проектированию.

В процессе обучения в вузе особое внимание желательно обратить на мотивацию необходимости реализации студентами объектно-ориентированного проектирования. От того, как будет проведено проектирование, скорее всего, будет зависеть выполнение всего проекта. Ошибки, которые могут быть допущены студентами при проектировании, придется исправлять в процессе написания программного кода. В связи с этим большую часть времени работы студентов над проектом рекомендуется направить на объектно-ориентированное проектирование.

М. Фаулер (M. Fowler) считает, что не обязательно создавать диаграммы классов для всего программного кода [3]. Во время обсуждения диаграммы классов полезно создавать наброски, эскизы наиболее сложных отношений и взаимодействий между классами. Это справедливо при конструировании сложных программных комплексов, а при разработке простых учебных проектов важно отображать все взаимосвязи, так как количество взаимодействующих классов при этом небольшое.

Проектирование на начальных этапах создания программного кода может способствовать серьезному отношению студентов к планированию работы и пониманию того, что нельзя создавать программу, пока нет четкого представления об ее структуре.

Мы считаем, что при обучении студентов проектированию следует выбрать задания из известных им областей знаний, чтобы сконцентрироваться на объектно-ориентированном проектировании, а не собирать и изучать информацию по какой-либо теме, не имеющей отношения к занятию. При этом можно использовать сюжетные примеры, задания и лабораторные работы, которые могут способствовать развитию устойчивой познавательной мотивации студентов. Сюжетные задания строятся на основе реальных объектов, процессов, ситуаций и т.д. Прототипом сюжетных задач для объектно-ориентированного проектирования может быть игра в шахматы, моделирование дорожного движения на перекрестке, Интернет-магазин (для студентов-экономистов) и т.п.

Многие интересные проекты с точки зрения обучения объектно-ориентированному проектированию могут представлять трудность для студентов в плане программирования. Для использования этих проектов в обучении желательно познакомить студентов с примером программного кода, который вызывает у них наибольшее затруднение, предоставить подробные комментарии и объяснения.

Нами было замечено, что построение графического интерфейса пользователя полезно изучать на основе готового программного кода, на базе которого студенты будут создавать диаграммы UML. Это позволяет установить обратную совместимость UML – программный код. Желательно рассматривать построение графического интерфейса пользователя как дополнение к другому проекту, а не как самостоятельный проект.

В процессе обучения объектно-ориентированному проектированию рекомендуется объяснить студентам, что документирование – это трудоемкий процесс, который имеет большое значение для понимания созданной диаграммы классов, внесения в нее изменений, исправления ошибок и проверки ее корректности. Для того чтобы документация была качественной и полной, желательно, с самого начала создания проекта документировать классы и отношения между ними.

Комментарии в языке UML можно добавлять внутри элементов диаграммы UML. Для этого перед комментарием ставиться двойной дефис. Если комментарий имеет отношение в целом к элементу диаграммы UML или к нескольким элементам сразу, тогда используется элемент «примечание» диаграммы UML.

К дидактическим рекомендациям можно отнести то, что в процессе обсуждения проекта студентам желательно вести протокол принятых решений, которые удобно записывать в цепочку и представлять в виде «древа» принятых решений. Если в результате обсуждения будет изменяться какое-либо решение, то тогда ими устанавливается отдельная «ветвь». Этот дидактический прием позволит студентам не забывать об интересных идеях, развивать их в дальнейшем, а по завершении обсуждения проекта, в него будут занесены основные и наиболее значимые находки, которые могут пригодиться студентам для других проектов.

Для обучения основам объектно-ориентированного программирования был специально создан пакет BlueJ [4], являющийся интегрированной средой разработки с простым интерфейсом, включающий редактор, компилятор, отладчик, автоматическое генерирование документации с использованием javadoc и д.р.

BlueJ позволяет наглядно представить модель создаваемой программы, на основе наиболее часто используемых элементов диаграммы классов языка UML. В BlueJ создание классов и работа с объектами реализована с использованием диалогов, назначение которых не в ускорении процесса разработки, а в применении понятного для студентов интерфейса. Из нотаций отношений между классификаторами языка UML включены наиболее часто используемые отношения

обобщения и зависимости. Установление отношения обобщения на диаграмме классов UML в пакете BlueJ сопровождается соответствующими изменениями в программном коде. Если создается экземпляр класса, то на диаграмме классов UML устанавливается отношение зависимости между соответствующими классами.

Пакет BlueJ выступает в роли промежуточного звена между обучением основам объектно-ориентированного программирования и более глубоким изучением проектирования и языка UML. Навыки, получаемые студентами при обучении объектно-ориентированному программированию с использованием пакета BlueJ, могут способствовать дальнейшему освоению объектно-ориентированного программирования и проектирования на языке UML.

Перечислим основные дидактические задачи, которые студенты смогут решать с помощью BlueJ:

1. освоение объектно-ориентированного стиля мышления;
2. обучение созданию объектно-ориентированного программного кода на языке Java;
3. ознакомление на практике с некоторыми элементами языка UML.

Таким образом, объектно-ориентированное проектирование с использованием пакета BlueJ позволит сделать обучение студентов объектно-ориентированному программированию более доступным и понятным, а дидактические рекомендации создадут условия для качественного обучения в вузе основам объектно-ориентированного программирования и проектирования.

СПИСОК ЛИТЕРАТУРЫ:

1. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. - М.: Конкорд, 1992. – 519 с., ил.
2. Грэхем, Иан. Объектно-ориентированные методы. Принципы и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 880 с.: ил. – Парал. тит. англ.
3. Фаулер М. UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ-плюс, 2004. – 192 с., ил.
4. Проект BlueJ: http://www.bluej.org.

ФОРМИРОВАНИЕ ОПЫТА КРЕАТИВНОЙ ДЕЯТЕЛЬНОСТИ СТУДЕНТОВ

Петрова В.Н.
Университет РАО
Москва, Россия

Под креативностью многие авторы понимают готовность к творчеству, так ли это на самом деле? Не секрет, что большое количество различных приемов и способов нестандартного мышления позволяют человеку более комфортно чувствовать себя в ситуациях неопределенности и неоднозначности, но они далеки от творческого процесса, если нет понимания природы творчества. В психолого-педагогической литературе представлены различные трактовки понятий «творчество», «креативность», зачастую происходит их смешение.

Наша позиция отличается от общепринятых подходов, основная концептуальная идея сводится к тому, что мышление и творчество – это разные области:

- творчество не использует мышление, а мышление не использует творчество, во время творчества не может быть мышления, во время мышления не может быть творчества;
- мышление работает с продуктами творчества и мышления, а творчество с продуктами мышления и творческим продуктом;
- восприятие творчества у людей разное, но суть творчества одна;
- процесс творчества не нуждается в развитии, так как он постоянен в плане своей конструкции и ограничивается лишь нашим опытом творчества;
- креативность – одно из следствий формирования опыта творчества, она основывается на обучении творчеству (творчество первично).

Творчеству может обучаться любой человек, желающий этого. В наше время растет спрос на творческую личность, на творческий продукт, а четкого понимания, что такое творчество и креативность, озарение и интуиция - нет.

Мы считаем, что озарение позволяет рассматривать предмет исследования в той точки зрения, которая ранее не рассматривалась, зачастую данный феномен происходит случайно. Достаточно вспомнить Архимеда, который открыл закон только на основе того, что он увидел и сопоставил с данным ему заданием. Озарение не относиться к обработке материала, а является случайным обращением внимания на некоторый объект с той стороны, с которой он не рассматривался в данной ситуации. Наше определение термина «интуиция» совпадает с нашим определением креативности, поэтому необходимо внести уточнение. Креативность человеку полностью подчиняется, а интуиция – нет, ее нельзя запустить волевым усилием, она идет спонтанно и некорректно говорить о целенаправленности. Под креативностью и интуицией мы понимаем обработку невербальной информации с участием памяти внимания, но в креативности эта обработка идет целенаправленно, а в интуиции – нет.

Для формирования опыта креативной деятельности должна осуществляться целенаправленная работа: проектирование, планирование, специальный отбор креативных элементов, появляющихся в творческом процессе. Это, конечно идеальный вариант создания опыта креативной деятельности. На практике креативность сводится