Опыт проектирования учебного программного продукта с помощью инструментария PlantUML

А.В. Параничев,
преп. СПбКТ, pav-83@yandex.ru
СПбГУТ, г. Санкт-Петербург

Опыт использования различных инструментов проектирования привел к выводу о возможном использовании инструментария PlantUML в качестве основного при проектировании учебных программных продуктов. Рассматривается построение диаграмм при проектировании учебного программного продукта по стандарту UML 2 с помощью инструментария PlantUML. Нами предложены варианты представления как 9 видов диаграмм, официального представленных в PlantUML, так и 5 UML-диаграмм, формально не поддерживаемых в данном средстве проектирования.

Experience in using various design tools led to the conclusion of the possible use of the PlantUML tool as the main tool in the designing of educational software products. The paper presents the construction of diagrams in the design of the educational software product under the UML 2 standard using the PlantUML toolkit. We have proposed variants of representation in 9 forms of diagrams, officially presented in PlantUML, and 5 UML diagrams formally not supported in this design tool.

Введение

Проектирование программного обеспечения является одной из важнейших составляющих при обучении будущих IT-специалистов решению практических задач, таких как разработка специализированной САПР [1]. Вместе с тем, в настоящее время не выработано устоявшихся методик проектирования учебных программных продуктов; просматривается лишь общая нацеленность на полное или частичное соответствие проектируемых решений стандартам UML 1 и/или UML 2 [2–8].

Опыт программных разработок, показывает, что необходимость составления всех 14 диаграмм, содержащихся в стандарте UML 2, вряд ли может возникнуть при проектировании даже разнопланового программного продукта. Однако может оказаться полезным «попробовать» составить каждую из диаграмм как в учебных целях, так и при проектировании коммерческого продукта.

В таком случае необходимо:

- более полно раскрыть постановку задачи на создание программного продукта;
- разработать спецификацию на программный продукт, достаточную для перехода к созданию программного кода.

1. Выбор инструмента для построения диаграмм UML 2

Опыт проектирования UML диаграмм позволяет утверждать, что наибольшие результаты на этапе обучения проектированию программ достигаются использованием CASE-инструмента[1] PlantUML [9, 10] за счет возможностей создания диаграмм с помощью набора инструкций в виде скрипта. Здесь следует отметить некий игровой эффект, заинтриговывающий проектировщика тем, что в следующий момент времени может получиться более совершенная диаграмма. Оборотной стороной медали является то, что PlantUML постоянно меняется, в результате диаграммы, созданные в разное время, могут по-разному выглядеть.

Кроме того, официально PlantUML поддерживает 9 видов диаграмм из 14, представленных в стандарте UML 2. Следует отметить, что все виды диаграмм UML 2 не поддерживает ни один из существующих в настоящее время (2017 г.) CASE-инструментов. При этом, с помощью различных ухищрений, в PlantUML могут использоваться и те 5 видов диаграмм, которые официально не поддерживаются в данном CASE-инструменте; при составлении официально неподдерживаемых видов диаграмм UML 2, предлагается руководствоваться следующими положениями:

- *диаграмма пакетов данных (package diagram)*: используется ключевое слово «package» для обозначения пакетов данных; если пакет данных не содержит другие пакеты данных, то при его описании блок фигурных скобок не указывается; подстановка одного пакета данных в другой показывается с помощью соединительного блока, используемого при наследовании интерфейсов диаграммы классов (class diagram);
- *диаграмма профилей (profile diagram)*: см. диаграмму пакетов данных; при этом выполняется стереотипное обозначение взаимодействия с помощью угловых скобок, а внутри профилей – иконки из стандартного набора либо спрайты [10];
- *диаграмма составной структуры (composite structure diagram)*: используется диаграмма объектов, при этом указываются разделы «*provided interface*» и «*required interface*» (для лаконичной записи *предоставляемых* и *требуемых* для реализации интерфейсов предлагается использовать макроопределения и макроподстановки);
- *диаграмма обобщения взаимодействия (overview interaction diagram)*: для представления общей структуры используется синтаксис диаграммы деятельности (activity diagram), а для представления диаграммы последовательности (sequence diagram) или ссылочных данных (reference data) – используется блок диаграммы классов «*partition*», внутри которого применяется синтаксис диаграммы состояний (state diagram), но с обязательной нумерацией последовательности всех действий в подписях к ним; сами диаграммы также нумеруются, при этом указывается «*sd*» для диаграммы последовательности и «*ref*» для определения набора ссылочных данных;

[1] CASE-инструмент или CASE-средство (Computed-Aided Software Engineering tool) – «программный продукт для IT-проектировщиков, позволяющий обеспечить автоматизированную поддержку процессов жизненного цикла программного обеспечения» [6]

- *диаграмма коммуникации (communication diagram)*: см. диаграмму обобщения взаимодействия; при этом в обязательном порядке указываются имена «общающихся» объектов, а типы объектов – опционально.

Особенности выбора диаграмм, создаваемых в процессе проектирования учебного программного продукта, рассматриваются в следующем разделе.

2. Выбор групп диаграмм UML 2 при проектировании учебного программного продукта

Для того, чтобы проектировщик программного продукта мог сориентироваться, исходя постановки задачи и, если имеются, данных спецификации на программный продукт, нами предложена группировка диаграмм UML 2, представленная в табл. 1.

Таблица 1

Группы диаграмм UML 2 при проектировании учебного программного продукта

№	Часть/целое проектируемого программного решения	Назначение группы диаграмм UML 2 при проектировании программного продукта	Группа диаграмм UML 2 (в скобках указаны официальные названия диаграмм согласно стандарту [8])	Особенности построения диаграмм UML 2 (в скобках указываются номера диаграмм соответствующей группы)
1	Решение в целом	Представить работу программы в виде «коробочного» решения.	1.1. Прецедентов (Use case); 1.2. *Компонентов (Component)*; 1.3. *Развертывания (Deployment)*.	Выполняется описание развертывания приложения (1.3), включающего компоненты (1.2), определяемые прецедентами (1.1), возникающими при разработке и эксплуатации программы.
2	Работа части программного функционала	Составить блок-схемы работы основных функций программы.	2.1. Последовательности (Sequence); 2.2. Состояний (State); 2.3. Деятельности (Activity).	Определяется последовательность действий основных объектов (2.1), устанавливается взаимодействие основных состояний объектов (2.2) и возникающих при этом процессов (2.3).
3	Решение в целом	Показать взаимодействие функциональных модулей программы.	3.1. *Классов (Class)*; 3.2. Коммуникаций (Communication);	Выполняется описание разрабатываемых классов (3.1) и передача данных между ними (3.2).
4	Решение в целом	Показать взаимодействие структурных модулей программы.	4.1. *Профилей (Profile)*; 4.2. Обобщения взаимодействия (Overview interaction); 4.3. *Пакетов данных (Package)*;	Выполняется описание структурного состава приложения (4.3) по данным схемы взаимодействия программных модулей (4.2) для заданных профилей работы программы (4.1).
5	Работа части программного функционала	Составить «временную развертку» работы части программного продукта.	5.1. *Составной структуры (Composite structure)*; 5.2. *Объектов (Object)*; 5.3. Времени (Timing).	Осуществляется описание структур данных (5.1), определяются соотношения между различными данными по типу часть/целое (5.2), а затем и во времени (5.3).

Примечания:
1. Курсивом отмечены *структурные* (*structural*), а обычным шрифтом – поведенческие (behavioral) виды диаграмм.
2. Выбирать группы диаграмм предлагается, рассматривая их в порядке возрастания номера группы; внутри каждой группы подставлена и охарактеризована рекомендуемая последовательность создания диаграмм.

Выбор групп диаграмм (табл. 1), которые следует составить при проектировании учебного программного продукта, определяется видом работ (перечислены от простого к сложному):

- *практическая и/или лабораторная работа*: группа диаграмм задается преподавателем, обучаемый выполняет проектирование исходя из GUI-приложения;
- *курсовой проект*: необходимо представить все группы диаграмм по данным технического задания на разработку; группы диаграмм создаются в любой последовательности, по выбору обучаемого;
- *дипломный проект*: необходимо выбрать не менее двух групп диаграмм, включая группу № 3, в зависимости от постановки задачи и информации, определяющей спецификацию на проектируемый программный продукт.

В ходе проектирования каждого учебного программного продукта, обучаемый может не только разобраться с тем, как предлагаемые группы диаграмм помогают перейти к более продуктивному кодированию, но и «опровергнуть» представленные наборы диаграмм, предложив свой вариант. Таким образом, лишь предполагая полезность представленной в данной работе группировки диаграмм, обучаемый «провоцируется» на более глубокое осмысление всех диаграмм UML 2 (при этом, очевидно, выражаются его индивидуальные особенности и вкусы проектировщика).

3. Проектирование учебного программного продукта в PlantUML

Пример исходных данных при проектировании учебного программного продукта представлен в табл. 2.

Таблица 2

Пример задания на разработку для проектирования учебного программного продукта

Наименование	Содержание	Примечание
Тема	Разработка системы поддержки пользователей ПК	Курсовой проект
Задание	Разработать программу, позволяющую: – осуществлять поиск решений по различным	Предусмотреть, что: – поиск решений необходимо осуществлять

	проблемам; – создавать заявки на обслуживание; – обслуживать заявки.	по различным неисправностям ПК; – для заполнения заявок должен быть заготовлен шаблон.
Средства разработки	PlantUML, Visual Studio	Выполнить проектирование программного продукта в PlantUML до написания программного кода.

Как видно из табл. 2, задание на разработку программного продукта сформулировано достаточно абстрактно, следовательно, **группа диаграмм № 1** позволит конкретизировать исходные данные на разработку в процессе составления диаграмм *прецедентов*, *компонентов* и *развертывания* (рис. 1).

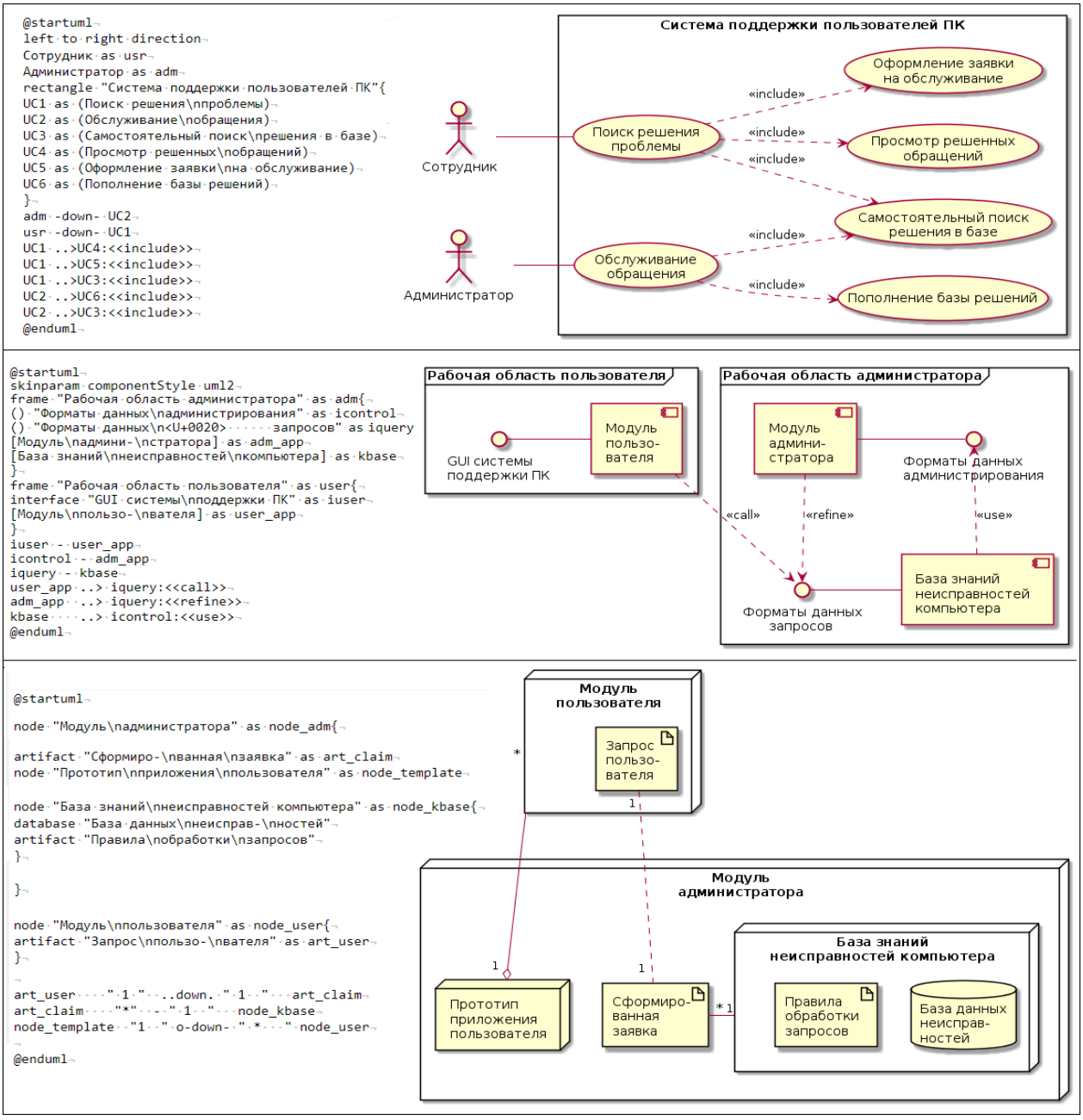

рис. Диаграммы прецедентов (вверху), компонентов (посередине) и развертывания (внизу) для учебного программного продукта по теме «Разработка системы поддержки пользователей ПК»

Из представленных диаграмм (рисунок) видно, что:

- на диаграмме *прецедентов* представлены две основные роли (actor): «*администратор*» и «сотрудник», при этом на последующих диаграммах предложено более общее определение роли сотрудника – «*пользователь*» (поскольку заявку может составить пользователь, формально не являющийся сотрудником); общий прецедент для обеих ролей – самостоятельный поиск в базе решений, – с учетом диаграммы компонентов, может быть уточнен как «*поиск по запросу в базе знаний неисправностей компьютера*»;
- на диаграмме *компонентов* показана необходимость разработки интерфейсов формируемых запросов, которые пользователь вызывает (*call*), а администратор уточняет (*refine*); в свою очередь, разрабатываемые интерфейсы для управления базой знаний использует (*use*) только администратор;
- на диаграмме *развертывания* представлены соотношения: между шаблоном пользовательского приложения и его независимыми развернутыми копиями («один-ко-многим»), между базой знаний неисправностей и сформированными заявками на обслуживание неисправности компьютера (соотношение «один-ко-многим»); при этом каждому запросу пользователя может соответствовать одна сформированная заявка (соотношение «один-к-одному»).

Дальнейшее проектирование учебного программного продукта «Разработка системы поддержки пользователей ПК» (табл. 2) выполнялось в следующем порядке (соответствующие группы диаграмм перечислены в табл. 1):

1. Ориентируясь на диаграммы группы № 1 (рис. 1), составлены **диаграммы группы № 2**: в итоге получены блок-схемы основных функций (в каждом случае сначала перечислялась общая последовательность действий, затем – основные состояния разрабатываемой системы, далее – логическая взаимосвязь действий при переходе из одного состояния в другое):
   - «обработка пользовательского запроса о неисправностях компьютера»;
   - «создание заявки на обслуживание неисправного компьютера»;
   - «автоматизированное обслуживание базы знаний неисправностей компьютера».
2. С учетом полученных диаграмм групп № 1 и № 2, показано взаимодействие классов на структурном и функциональном уровне (**диаграммы группы № 3**); в частности разрабатывались следующие классы:
   - «*Administrator*» – для реализации интерфейсов управления базой знаний неисправностей ПК;
   - «*User*» – для работы с GUI пользовательского приложения;
   - «*Claim*» – для обработки данных заявки на устранение неисправностей ПК;
   - «*KBase*» – для работы с базой знаний неисправностей ПК (включает подкласс «*KB_Database*»для работы с базой данных и подкласс «*KB_Rules*» для формирования правил работы с базой знаний);
3. Ориентируясь на диаграммы групп № 1 и 2, построены **диаграммы группы № 4**, при этом:
   - с помощью диаграммы *профилей* (с учетом диаграммы *прецедентов*), удалось показать, что в описании функций *администратора* следует разделить сопровождение программного и технического обеспечения проекта: соответственно, создаются профили «*проектировщик-разработчик*» и «*проектировщик-эксперт*» (понятие «*администратор*» предложено заменить термином «*проектировщик*», чтобы указать на разнонаправленность возникающих задач проектирования программного продукта);
   - с помощью диаграммы *обобщения взаимодействия* (с учетом диаграмм *последовательности*, *состояния, деятельности* и *компонентов*) построена общая блок-схема, на которой показана работа модулей с точки зрения внешнего (*external*) взаимодействия приложения и, одновременно с этим, показано взаимодействие компонентов внутри программы (*internal data*);
   - с помощью диаграммы *пакетов данных* (с учетом диаграммы *развертывания*) показан выбор базы данных MySQL в виде соответствующих дополнений для среды программирования Visual Studio, заданной в табл. 2.
4. Построение **диаграмм группы № 5** оказалось по большей части избыточным при проектировании представленного программного продукта: с помощью диаграмм *составной структуры* и *объектов* выполнено описание, соответственно, состава и взаимосвязей диаграммы классов; в свою очередь, на *временной* диаграмме (с учетом диаграммы профилей) показано взаимодействие *пользователей*, *проектировщика–разработчика* и *проектировщика–эксперта*; при этом функции тестирования были распределены между указанными тремя ролями: внутреннее тестирование проводилось силами *проектировщика-разработчика*, внешнее – *пользователями* и *проектировщиком-экспертом*.

Следует отметить, что, после построения и описания всех перечисленных выше диаграмм, удалось достаточно быстро выполнить кодирование, отладку и тестирование спроектированного программного продукта.

Заключение

Таким образом, ориентируясь на требования, предъявляемые к коммерческим программным решениям, нами предлагается осваивать на практике все 14 диаграмм UML 2, используя для этого CASE-инструмент PlantUML. Для упрощения такого освоения нами предложены 5 групп диаграмм, особенности их выбора и построения при проектировании учебного программного продукта.

Составление UML-диаграмм может быть использовано будущими IT-специалистами в процессе получения как высшего технического (уровень *инженера-программиста*), так и среднего специального образования (уровень *техника-программиста*). Очевидно, что уровень *программиста-техника* должен быть максимально облегчен: путем предварительного построения всех видов диаграмм «под копирку», а также путем рассмотрения примеров проектирования типового учебного программного продукта. Уровень *инженера-программиста* следует, напротив, усложнить, предлагая для обучаемых, претендующих на глубокое усвоение материала, такие дополнения, реализация которых неочевидна из типовых примеров; предлагать к реализации один или несколько паттернов проектирования по GoF-стандарту [11] с обоснованием выбора таких шаблонов для разных групп диаграмм.

Литература

1. Параничев А. В. Автоматизация компоновки сборочного чертежа на примере взрывозащищенного корпуса с кабельными вводами и клеммами // САПР и графика, 2015. № 10. С. 78-79. URL: http://sapr.ru/article/25036 (дата обращения: 01.12.2017).
2. Коломец Н. В. О методах и средствах проектирования программного обеспечения (обзор и примеры) // Ростовский научный журнал, 2017. № 4. С. 249-265.
3. Галиаскаров Э. Г., Агеев М. С., Умнов Д. М. Организация учебного проекта разработки программного обеспечения информационной системы // Объектные системы, 2013. № 7. С. 9-14.
4. Вичугова А. А., Вичугов В. Н., Цапко Г. П. Методы и средства UML как инструменты проектирования программного обеспечения // Вестник науки Сибири, 2013. № 2. С. 73-78.
5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3d Ed. Boston: Addison-Wesley, 2003. 178 p.
6. ISO/IEC/IEEE 24765:2010. System and software engineering – Vocabulary. Geneva: ISO, 2010. 410 p.
7. Information technology – Object Management Group. Unified Modeling Language (OMG UML). Version 1.4. 2001. 454 p.
8. Information technology – Object Management Group. Unified Modeling Language (OMG UML). Version 2.5. 2015. 794 p.
9. Tools using the PlantUML language. URL: http://plantuml.com/running (дата обращения: 01.12.2017)
10. Drawing UML with PlantUML. Language Reference Guide. 2017. 128 p.
11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software // Forework by Grady Booch. 37th printing. Boston: Adisson-Wesley, 2009. 417 p.