УДК: 004.031
DOI: 10.30987/article_5cf2d1f855d980.40342469

В.Л. Аршинский, Л.В. Аршинский, С.В. Бахвалов

ПРИНЦИПЫ НЕОБХОДИМОСТИ И ДОСТАТОЧНОСТИ В СИСТЕМАТИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статья содержит описание подхода к систематизации программного обеспечения (ПО) на основе принципа необходимости. Традиционный подход неявно опирается на принцип достаточности: что может делать программа. Принцип необходимости кладет в основу необходимость программы для решения какой-либо задачи, а также взаимную необходимость ПО в информационно-вычислительной системе. Вводятся три типа необходимости: экзистенциальная, функциональная и целевая со своей внутренней иерархией, что позволяет разделять программные объекты по слоям.

Ключевые слова: *программное обеспечение, вид программного обеспечения, необходимость, ценность, классификация.*

V.L. Arshinskiy, L.V. Arshinskiy, S.V. Bakhvalov

THE PRINCIPLES OF NECESSITY AND SUFFICIENCY TO SYSTEMATIZE SOFTWARE

The article describes the approach to systematization of software based on the principle of necessity. The traditional approach implicitly relies on the principle of sufficiency: what a program can do. The principle of necessity underlies the need for a program to solve any problem, as well as the mutual need for software in the information and computer system. Three types of necessity are introduced: existential, functional and target with its own internal hierarchy, which allows separating program objects by layers.

Keywords: *software, type of software, necessity, value, classification.*

Введение

Одним из небезынтересных вопросов теории программного обеспечения является его классификация. Разделяя многообразие изучаемых объектов на классы, систематизируя их, удается внести порядок в кажущиеся случайными предметы и явления, оказывается возможным разработать методы, применимые к данному классу или группе классов, упорядочить объекты в том или ином отношении, выявить генетические связи, и так далее. Важность и нужность классификации вопросов не вызывает. Классическим примером выступает биологическая классификация.

В сфере программного обеспечения (ПО) классификация присутствует тоже. Однако уже первое знакомство с ним показывает, что единого подхода здесь нет. Одни авторы разбивают его на системное, прикладное и инструментальное, с последующим разбиением на более мелкие классы [1-5], другие – на базовое и специальное, объединяя в базовом системное и инструментальное, а также сервисные программы (антивирусное ПО, утилиты, оболочки ОС) и ПО технического обслуживания, а в специальном – прикладное ПО [5, 6], третьи – базовое, системное, служебное (утилиты) и прикладное ПО [7, 8]. И так далее.

ПО классифицируют по назначению, платформе, способу взаимодействия с аппаратными средствами, способу распространения, переносимости, сфере применения и т.п. Многообразие классификаций. Причем и способы классификации, и наименования классов зачастую разнятся от автора к автору и даже в одних и тех же материалах может излагаться сразу несколько взглядов на классификацию (см. напр. [1, 5]).

Непросто выглядят и таксономические связи, иерархия ПО. В одних случаях разбиение, к примеру, на системное, прикладное и специальное ПО является плоским – все множество компьютерных программ членится как однородное целое. В других – устанавливается приоритетность программ, например, по степени близости к аппаратной части [5, 7].

Создается впечатление, что многие авторы исходят из интуитивных представлений о классификации программных объектов или из сложившихся традиций. Порой нарушается и базовые принципы классификации: один элемент – один класс, принцип полноты разбиения (соразмерность) и т.д. [9]. Все это существенно усложняет вопрос. Не стремясь выяснять правоту тех или иных взглядов, попробуем изучить его с, условно говоря, формально-логических позиций.

1. Необходимость и достаточность

В основу классификации положим не задачи, решаемые той или иной программой, а ее *необходимость* для решения этих задач, а также взаимную необходимость программ внутри информационно-вычислительно системы (ИВС) в целом.

Рассуждая о тех или иных объектах, включая программные объекты, выделяем три аспекта их существования:

1) базовый (экзистенциальный) – объект существует;
2) функциональный – объект (существуя) функционирует, хотя бы частично работоспособен;
3) целевой – объект (функционируя) достигает поставленных перед ним целей.

Вообще говоря, с этих позиций можно рассматривать любые объекты. В том числе, входящие в такое сложное образование, как информационно – вычислительные системы (ИВС): оборудование, программное обеспечение, пользователей ИВС, обслуживающий персонал, разработчиков, документацию и т.д. При этом объекты могут вступать в отношения необходимости и достаточности.

Определение 1. Будем говорить, что объект A *необходим* для объекта B, если существование, функционирование или достижение цели объектом B невозможно в полной мере вне объекта A, когда утрата объекта A либо части его функциональности означает хотя бы частичный ущерб для B.

В этом случае говорим также, что объект A представляет *ценность* для B. Записывать отношение необходимости будем как:

$$\neg A \to \neg B. \quad (1)$$

Необходимость может существовать в любом из трёх вышеуказанных смыслов или их комбинации. Например, существование аппаратной часть ЭВМ необходимо для функционирования и целедостижения любого программного продукта. Там, где это важно разные типы необходимости обозначаем, соответственно, как $\to_e$, $\to_f$, $\to_g$.

Заметим, что между аспектами существования объектов также существует отношение необходимости: 1 необходим для 2; 2 необходим для 3. Это важно.

Определение 2. Будем говорить, что объект A *достаточен* для объекта B, если существование, функционирование или достижение цели объектом A означает также существования, функционирование или достижение цели (хотя бы частично) объектом B. Характер достаточности можно оговаривать отдельно.

В случае достаточности говорим также, что объект A *полезен* для B. Записываем это как:

$$A \to B. \quad (2)$$

Например, использование любой из программ-редакторов текстов достаточно для создания пользователем текстового файла. Соответствующая программа полезна для пользователя.

Сразу заметим, что в общем случае ценность A для B не означает его полезности и наоборот. Например, если в системе имеется две равнозначных по функциональности программы A_1 и A_2, то каждая по отдельности ценности не имеет, так как удаление, снижение или утрата работоспособности, либо не достижение цели программой A_1 при сохранении A_2 (и наоборот) не скажется на B. В то же время каждая из них может обладать полезностью.

Аналогично, если A_1 и A_2 необходимы для B только совместно, полезность каждой отдельной отсутствует.

Рассмотрим, как эти понятия работают при классификации ПО.

2. Систематизация ПО

Основой функционирования ЭВМ является аппаратные средства (АС), которые принимаем во внимание. Также принимаем во внимание пользователя. Это две крайние точки предлагаемой систематизации. Все объекты, участвующие в этом процессе, делим на слои. АС отнесём к слою S_0.

Существование АС является необходимым условием существования и функционирования первого слоя программного обеспечения, который обозначим S_1. Речь идет о программах, обеспечивающих работу периферийных устройств, устройств ввода-вывода информации, программах, необходимых для функционирования ядра ИВС. Например, материнской платы ПК с установленными на ней процессором, памятью и иными основными архитектурными компонентами ЭВМ. Утрата любой из таких программ приведет к частичной или полной неработоспособности ИВС. Эти программы создаются разработчиками соответствующих АС и часто являются уникальными для каждого устройства. При этом АС (физически) могут существовать без программных объектов из S_1, однако S_1 вне АС невозможен.

Отношение необходимости между S_1 и S_0 взаимно. АС необходимо для S_1 экзистенциально, S_1 для АС – функционально. Учитывая, что экзистенциальная необходимость более фундаментальная, считаем, что отношение необходимости между S_0 и S_1 следующее:

$$\neg S_0 \to \neg S_1. \quad (3)$$

При этом программные объекты слоя S_1 необходимы для решения задачи управления отдельными составляющими АС.

Следующий слой ПО – S_2, необходимым для которого является слой S_1. ПО данного слоя применяется для управления комплексом АС как системой. К нему можно отнести операционные системы (ОС) в традиционном понимании этого термина (UNIX, DOS, Windows, Linux и т.п.). Отсутствие ОС не влияет на работоспособность отдельных АС в ИВС. ОС не является необходимым ПО для конкретного узла, но без нее невозможно связать АС в единую систему, способную решать поставленные задачи. ОС необходима для решения задачи управления ЭВМ как системой. Функционально ОС опирается на S_1, обращаясь к нему при работе с памятью, процессором, другими компонентами АС. В каких-то аспектах ОС может работать с АС минуя S_1. Однако полностью от этого слоя отказаться не может. Задавая правило:

Если для некоторого объекта O необходимы объекты слоев i_1 ,… i_k, где $i_1 <\ldots< i_k$,

то O принадлежит слою $i_k + 1$,

относим ОС к слою S_2.

Далее рассмотрим прикладное и инструментальное ПО (ППО и ИПО). Кто-то относят их к разным группам ПО, кто-то объединяет вместе. С точки зрения решаемых задач это

действительно разные классы программных объектов. Посмотрим на них в свете обсуждаемого подхода.

Необходимым условием функционирования ИПО является ОС, т.е. слой S_2. С этих позиций оно относится к слою S_3.

ППО также не способно функционировать без ОС. При этом ИПО не является для него функционально-необходимым. Однако если целью существования ППО является качественное предоставление информационно-вычислительных услуг, то ИПО необходимо для ППО в целевом смысле (качество ППО, написанного с помощью инструментальных средств, полагаем выше написанного без его помощи). Т.е. присутствует третий – целевой тип необходимости. Это дает основания размещать ППО в слое S_4.

Следует заметить, что ОС также могут разрабатываться с помощью ИПО. Т.е. в дополнение к функциональной необходимости

$$\neg\text{ОС} \rightarrow_f \neg\text{ИПО}, \quad (5)$$

существует отношение целевой необходимости:

$$\neg\text{ИПО} \rightarrow_g \neg\text{ОС}. \quad (6)$$

Более приоритетна функциональная необходимость, поэтому относим ИПО к слою, следующему за ОС, т.е. к S_3.

Есть еще одна группа программных объектов, называемых интерпретаторами. Их тоже можно рассматривать как своего рода ИПО. Но в отличие от ИПО, включающих компиляторы, ИПО в форме интерпретаторов функционально необходимо для работы прикладного ПО. Программные объекты, получающееся после компиляции, в соответствующем ИПО не нуждается. Интерпретируемое ПО работает только при наличии интерпретатора. Если инструментальная система интерпретирующего типа (ИПОИ) получена с помощью компилирующего ИПО (ИПОК), можно записать отношение:

$$\neg\text{ИПОК} \rightarrow_g \neg\text{ИПОИ}. \quad (7)$$

В этом же слое находится ППО компилируемого типа (ППОК):

$$\neg\text{ИПОК} \rightarrow_g \neg\text{ППОК}. \quad (8)$$

Т.е. ИПОИ попадает в один слой с ППО, транслированном на основе компиляции – S_4. В свою очередь ППО интерпретируемого типа оказывается в слое S_5:

$$\neg\text{ИПОИ} \rightarrow_f \neg\text{ППОИ}. \quad (9)$$

Отсюда следует вывод, что хотя ППОК и ППОИ выполняют одну и целевую ту же задачу – удовлетворение потребностей пользователя в информационно-вычислительных услугах, структурно они относятся к разным слоям. К слою S_5, в частности, относятся программы-макросы, работающие в составе некоторых офисных пакетов, скрипты и иные аналоги.

Замыкающим компонентом всей этой иерархии выступает пользователь (П). Ему для работы с ЭВМ необходимы программные объекты любого из указанных слоев. Наибольшим из них выступает слой S_5. Исходя из ранее приведенного правила делаем вывод, что пользователь, как объект, относится к слою S_6.

В случае более подробного деления ПО по слоям, пользователя относим к слою S_{N+1}, где N номер наивысшего слоя ПО.

Принцип необходимости можно применить и для классификации ПО по решаемым задачам. Вопрос не праздный, поскольку программные объекты часто классифицируются по их *возможностям*: что может делать та или иная программа. В этом виден *принцип достаточности*: если программная система способна решать некую задачу, её можно зачислить в соответствующий класс. Например редактор изображений можно причислить к классу программ обработки изображений и к классу досуговых программ. Какую-то полезную утилиту, в которую внедрены вредоносные функции, отнести и к сервисным, и к вредоносным программам. Табличный процессор рассматривать как средство работы с электронными таблицами и как вариант СУБД. И так далее. Таких примеров много. Если опираться на принцип достаточности, одна и та же программа может попасть более чем в

одну группу.

Прикладное ПО предназначено для решения задач пользователя, к числу которых чаще всего относятся:

1) выполнение профессиональных функций;
2) досуг;
3) обслуживание ЭВМ и повышение удобства работы на компьютере (утилиты);

Отдельно следует упомянуть вредоносное ПО, группа 4), где пользователями выступают его разработчики.

Как уже говорилось, при систематизации по принципу достаточности классификация размывается. Один и тот же объект может попасть в разные классы. К примеру, Adobe Photoshop может являться и профессиональной и досуговой. Вредоносное ПО становится профессиональным, если с его помощью кем-то достигаются профессиональные цели. Экспертные системы, офисные приложения, СУБД, если они используется в профессиональных целях – профессиональное ПО, однако какие-то из этих продуктов могут использоваться в целях досуга, или для иных задач. Классификация по необходимости выглядит более последовательной. Например, по необходимости (ценности) для осуществления конкретных видов деятельности (обработка изображений, обработка текстов, организованное хранение данных, работа с числовыми таблицами, и т.п.). Так, если программа А необходима пользователю (представляет ценность) для решения его профессиональных задач, она относится к группе 1). Если для отдыха и досуга вне профессиональных обязанностей – к группе 2). Если без неё снижается качество, удобство работы на компьютере – к группе 3). Наконец, если она необходима для несанкционированного вмешательства в работу ЭВМ – к группе 4).

Принцип необходимости кладет в основу те функции, для которых программа необходим пользователю, что она *необходимо должна делать*. Тогда табличный процессор – это средство работы с электронными таблицами, утилита с заложенными в нее вредоносными функциями – вредоносная программа (основной пользователь – получатель несанкционированных возможностей), программа обработки изображений, необходимая для решения именно этих задач, – редактор изображений. И так далее. Это позволяет уточнять систематизацию в каждом слое, вводя в них классы ПО, например, деля драйверы по обслуживаемым устройствам, ОС по аппаратным платформам, прикладные программы по задачам, для решения которых они необходимы. Для этого следует разделять главные и второстепенные функции ПО, выполняя классификацию по главным (необходимым) задачам. Классификация по достаточным (что может делать программа) размывает границы классов. Более формально, если декларируются отношения:

$$\begin{aligned} O_1 &\to F\&F_1; \\ O_2 &\to F\&F_2, \end{aligned} \tag{10}$$

где O_i – программные объекты, а F и F_i – реализуемая объектами функциональность, и

$$\begin{aligned} \neg O_1 &\to \neg F; \\ \neg O_2 &\to \neg F, \end{aligned} \tag{11}$$

то объекты O_1 и O_2 объявляются принадлежащими классу, связанному с функциональностью F, если F, F_1, F_2 – возможности, реализуемые программой, а F – возможность, которая пользователю необходима.

Обычно прикладные программы не зависят друг от друга, не представляют друг для друга ценности и могут быть отнесены к одному уровню иерархии. Если же между какой-то пары программ такое отношение возникает, для них может быть введена локальная иерархия. Считаем, что программа A и программа B, установленные в конкретной ИВС, находятся на одном уровне классификации, если для них не реализуется принцип необходимости (т.е. ни одна из них не является необходимой для другой), либо они являются взаимно ценными друг для друга.

Несколько слов нужно сказать о распространенной сегодня тенденции использования виртуальных машин. Отдельная виртуальная машина – это самостоятельный объект, в котором существуют те же слои рассматриваемых объектов, что и в реальной, но только в «гостевом» варианте. Однако в расширенном контексте, включающем реальные АС, хост и гипервизор, последний занимает слой S_4 (полагаем, что он разрабатывался с помощью инструментального слоя S_3), а значит слои «гостевой» машины, включая виртуальный вариант слоя S_0, должны нумероваться как S_{4+1}, S_{4+2}, и так далее.

3. Сетевое ПО

Отдельно следует обсудить систематизацию сетевого ПО. Его особенностью, как и особенностью сетевого оборудования, является то, что «нулевым слоем» здесь оказываются локальные ЭВМ, объединяемые сетью. Они являются необходимым элементом всей инфраструктуры. Тогда слоем S_1 выступает сетевое оборудование, которое, в свою очередь, можно рассматривать как необходимое условие функционирования слоя S_2 – программных объектов, непосредственно управляющих аппаратными средствами. Следующий уровень – S_3 занимает сетевая ОС, если она установлена поверх локальной ОС, или соответствующие компоненты локальной ОС, когда локальная ОС изначально является сетевой. Слой S_4 – программная реализация сетевых интерфейсов и протоколов (для упрощения систему стеков не рассматриваем). Слой S_5 образован прикладным сетевым ПО, включая сетевые утилиты, открытое ПО и в целом программы, работающие через сеть, программы-браузеры. Наконец, в слой S_6 можно вынести ПО, для функционирования которого необходимы программы-браузеры по аналогии с программами-макросами, работающими поверх некоторых офисных программ. Пользователя при таком расслоении относим к слою S_7.

Прикладное сетевое ПО, как и в случае изолированных ЭВМ, делим на классы согласно принципу необходимости с точки зрения пользователя.

Заключение

В работе представлен возможный подход к систематизации (включая классификацию) современного ПО. Большинство авторов при систематизации неявно исходят из принципа достаточности: программа зачисляется в тот или иной класс исходя из того, что она может делать, какие задачи способна решать. Это размывает границы классов, поскольку один и тот же программный объект иногда соответствует различным задачам. В работе предлагается исходить из принципа необходимости: программа относится к тому классу, какие задачи она необходимо должна решать. Тогда второстепенные возможности ПО отходят на второй план и при классификации не учитываются. Также принимается во внимание отношение необходимости между самими программными объектами. Это позволяет систематизировать программные объекты, «расслаивая» ПО по принципу взаимной необходимости. Причем вводятся разные виды необходимости: экзистенциальный, функциональный и целевой, с иерархией между ними. Последнее позволяет сохранить иерархичность по необходимости, когда программные объекты взаимно необходимы друг для друга, но в разном смысле. В результате выстраивается иерархия внутри ПО похожая на известную: аппаратная часть, системное ПО, прикладное ПО, но со своими особенностями. В частности, инструментальное ПО, предназначенное для создания программных объектов, расщепляется на два слоя: инструменты для создания пользовательских программ компилируемого типа и инструменты для создания программ интерпретируемого типа. Соответственно расслаивается и прикладное ПО. В целом принцип необходимости следует рассматривать как подход к классификации, позволяющий внести определенный порядок в вопрос систематизации ПО и его деления на классы.

Список литературы:

1. Алексеев Е.Г., Богатырев С.Д. Информатика. Мультимедийный электронный учебник [Электронный ресурс]. – Режим доступа: http://inf.e-alekseev.ru/text/Klassif_po.html.
2. Классификация программного обеспечения [Электронный ресурс]. – Режим доступа: http://mirznanii.com/a/308713/klassifikatsiya-programmnogo-obespecheniya-2.
3. Программное обеспечение ПК. Классификация ПО [Электронный ресурс]. – Режим доступа: http://mylektsii.ru/13-66978.html
4. Классификация программного обеспечения по назначению [Электронный ресурс]. – Режим доступа: https://wd-x.ru/klassifikaciya-programmnogo-obespecheniya-po-naznacheniyu.
5. Классификация программного обеспечения [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/3632/874/lecture/14291
6. Классификация программного обеспечения [Электронный ресурс]. – Режим доступа: https://pandia.ru/text/78/325/2652.php
7. Классификация программного обеспечения [Электронный ресурс]. – Режим доступа: http://ggf.tsu.ru/content/faculty/structure/chair/meteorology/publications/Прикладные_пакеты_программ_в_метеорологии/text/P1-1.htm.
8. Классификация программного обеспечения. Виды программного обеспечения и их характеристики [Электронный ресурс]. – Режим доступа: https://www.altstu.ru/media/f/Tema-6-Klassifikaciya-PO.pdf.
9. Грядовой Д.И. Логика: структурированный учебник (для вузов). 2-е изд., перераб. и доп. М.: ЮНИТИ-ДИАНА, 2003. 271 с.

References:

1. Alekseyev E.G.. Bogatyrev S.D. Informatika. Multimediynyy elektronnyy uchebnik [Multimedia electronic textbook]. – http://inf.e-alekseev.ru/text/Klassif_po.html. (in Russian)
2. Klassifikatsiya programmnogo obespecheniya [Classification of Software]. http://mirznanii.com/a/308713/ klassifikatsiya-programmnogo-obespecheniya-2. (in Russian)
3. Programmnoye obespecheniye PK. Klassifikatsiya PO [PC software. Classification of software]. – http://mylektsii.ru/13-66978.html. (in Russian)
4. Klassifikatsiya programmnogo obespecheniya po naznacheniyu [Software classification by purpose]. – https://wd-x.ru/klassifikaciya-programmnogo-obespecheniya-po-naznacheniyu. (in Russian)
5. Klassifikatsiya programmnogo obespecheniya [Software classification]. – http://www.intuit.ru/studies/courses/3632/874/lecture/ 14291. (in Russian)
6. Klassifikatsiya programmnogo obespecheniya [Software classification]. – https://pandia.ru/text/78/325/2652.php. (in Russian).
7. Klassifikatsiya programmnogo obespecheniya [Software classification]. – http://ggf.tsu.ru/content/faculty/structure/chair/meteorology/publications/text/P1-1.htm. (in Russian)
8. Klassifikatsiya programmnogo obespecheniya. Vidy programmnogo obespecheniya i ikh kharakteristiki [Software classification. Types of software and their characteristics] – https://www.altstu.ru/media/f/Tema-6-Klassifikaciya-PO.pdf. (in Russian)
9. Gryadovoy D.I. Logika: strukturirovannyy uchebnik (dlya vuzov) [Logic: structured textbook (for universities)]. 2-e izd.. pererab. i dop. M.: YuNITI-DIANA. 2003. 271 p. (in Russian)

Статья поступила в редколлегию 18.03.19.
Рецензент: к.т.н., доцент Брянского государственного технического университета Леонов Е.А.
Статья принята к публикации 30.04.19.

Сведения об авторах

Аршинский Вадим Леонидович
к.т.н., доцент кафедры автоматизированных систем Иркутского национального исследовательского технического университета,
тел.: +7 (3952) 40 15 64,
E-mail: pochemzria@mail.ru

Аршинский Леонид Вадимович
д.т.н., доцент, заведующий кафедрой информационных систем и защиты информации Иркутского государственного университета путей сообщения,
тел.: +7 (3952) 63 83 59,
E-mail: larsh@mail.ru

Бахвалов Сергей Владимирович
к.т.н., доцент, заведующий кафедрой автоматизированных систем Иркутского национального исследовательского технического университета,
тел.: +7 (3952) 40 51 65,
E-mail: bsv@istu.edu

Information about authors:

Arshinskiy Vadim Leonidovich
PhD., Associate Professor of the Department of Automated Systems of the Irkutsk National Research Technical University
tel.: +7 (3952) 40 15 64,
E-mail: pochemzria@mail.ru

Arshinskiy Leonid Vadimovich
Doctor of Technical Sciences, Head of the Department of Information Systems and Information Security of the Irkutsk State Transport University
tel.: +7 (3952) 63 83 59,
E-mail: larsh@mail.ru

Bakhvalov Sergey Vladimirovich
PhD, Head of the Department of Automated Systems of the Irkutsk National Research Technical University,
tel.: +7 (3952) 40 51 65,
E-mail: bsv@istu.edu