УДК 004.4’2

В.В. Трясцин

ПОСТРОЕНИЕ МОДЕЛИ УТИЛИЗИРОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Трясцин Виталий Викторович, *аспирант кафедры телекоммуникационных технологий и сетей Ульяновского государственного университета, закончил факультет математики и информационных технологий Ульяновского государственного университета. Инженер-программист ФНПЦ ОАО «НПО «Марс». Имеет статьи в области моделирования и исследования операций в организационно-технических системах. [e-mail: sauninvv@gmail.com].*

Аннотация

Статья посвящена вопросам повышения эффективности разработки программного обеспечения, а именно: анализу существующих методов автоматизации разработки программ, описанию применения методов утилизации программного обеспечения, разработке алгоритма утилизации, описанию критериев отбора программного обеспечения, пригодного для утилизации, а также описанию предметной области, построению моделей программного обеспечения и его артефактов.

Ключевые слова: программное обеспечение, артефакт программного обеспечения, утилизация программного обеспечения, переработка, восстановление, повторное использование.

Введение

В наши дни актуальность утилизации как этапа жизненного цикла материальных производств трудно недооценить. Ограниченность во времени и ресурсах, растущая конкуренция, уменьшение продолжительности жизненного цикла продукции вынуждают производить все более совершенную продукцию как можно быстрее и с минимальными затратами. С похожими требованиями столкнулся и рынок информационной продукции. Согласно существующим стандартам, описывающим жизненный цикл программного обеспечения (ПО), этап его переработки и утилизации у производителей отсутствует, а жизненный цикл завершается этапами ввода продукта в эксплуатацию и его сопровождением. Наметившаяся устойчивая тенденция к интеграции программных продуктов и созданию легких условий для их обновления и расширения вынуждает производителей ПО вырабатывать новые подходы к разработке и улучшению информационных проектов. Среди этих подходов – модульная разработка, формирование унифицированных интерфейсов взаимодействия между модулями, заимствование уже готовых продуктов при производстве новых, ведение и контроль версий ПО и т. д. Другими словами, современное ПО с позиции разработчика – это интеллектуальный продукт, собранный из взаимосвязанных расширяемых компонентов, которые допу-

стимо использовать в других программах. Таким образом, становится очевидной необходимость в определении, структуризации и систематизации еще одного этапа жизненного цикла ПО – его утилизации.

Утилизация как этап жизненного цикла

Если в сфере материального производства, в связи с ограниченностью природных ресурсов, уже давно применяется практика утилизации устаревших продуктов, их переработки с целью повторного использования, то производство ПО, даже с позиции существующих стандартов [1, 2], этапа утилизации лишено. Суть любой утилизации состоит в оценке состояния продукта, выявлении состава и взаимосвязей внутри него, а также в определении возможности повторного использования продукта целиком или же входящих в него элементов по отдельности. Применение методов утилизации к ПО приведет к повышению эффективности производства готового ПО, что позволит более результативно использовать уже готовые информационные ресурсы – артефакты – как при сопровождении уже готовых продуктов, так и при разработке новых [3]. Под артефактами понимаются объекты, полученные в ходе выполнения проекта, стадии или процесса [4]. Из артефактов особо выделяют продукты. Продукт – это артефакт, который произведен, измерим и может быть либо конечным объектом сам по себе, либо объектом-компонентом.

Согласно методологии RUP, процесс разработки ПО – это процесс разработки различных артефактов и их переработки друг в друга с целью получения готового программного продукта. Стоит отметить, что артефакты ПО, введенные в методологии RUP, существуют, взаимодействуют и преобразуются в итоговый продукт также и в других методологиях. Знание состава ПО и взаимосвязей элементов внутри него позволяет выделить в составе продукта артефакты и использовать их в дальнейшем. В результате анализа ПО была получена модель утилизируемого ПО (рис. 1), в которой отражены как его артефакты, так и важные факторы, оказывающие влияние на итоговое состояние ресурсов и сам продукт в целом. Так, отсутствие информации о целевой платформе или операционной системе, под которые был разработан продукт, ставит запуск и работу ПО под большое сомнение, не говоря уже о качестве и надежности. В то же время, в силу проблемы незримости ПО [5], если разработчик какого-то артефакта по каким-либо причинам недоступен, то возможность повторного использования исходного кода стремится к нулю или приводит к дополнительным затратам.

Утилизация ПО – это необходимый для его разработчика процесс, потому что он дает на выходе артефакты ПО, которые можно использовать при разработке новых программных продуктов. У конечного пользователя программного продукта отсутствует необходимость в доступе к информационным ресурсам, из которых состоит ПО, следовательно, процесс утилизации происходит на стороне разработчика. Таким образом, утилизация – это процесс, выполняемый на стороне разработчика и при его участии в целях анализа структуры готового ПО, выявления информационных ресурсов – артефактов и оценки возможности их повторного использования.

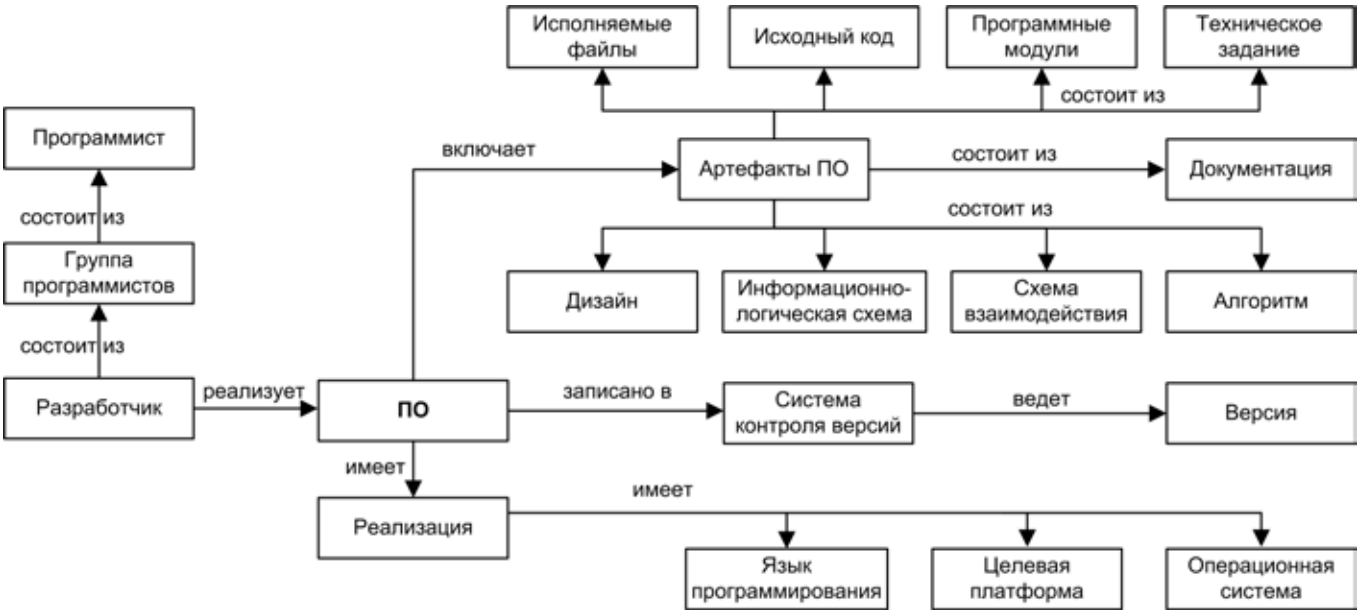

Рис. 1. Модель утилизируемого ПО

Основными методами утилизации являются:

1. Восстановление продукта – осуществляется в том случае, когда продукт в ходе эксплуатации утратил свои полезные свойства, но при этом целесообразнее восстановить функциональность продукта, чем производить или приобретать новый аналог. Применительно к ПО означает, что какое-то ПО не в полной мере соответствует предъявленным к нему требованиям и при этом доступно для изменения. После выяснения списка расхождений ПО дорабатывается под новые требования. Данный метод применяется, когда изменения затрагивают лишь небольшой фрагмент проекта и не приведут к изменению структуры ПО (к примеру, исправление ошибок в документации, перекомпиляция программ под другую операционную систему и т. п.).

2. Повторное использование – применяется в тех случаях, когда у продукта заканчивается его жизненный цикл, но его некие полезные свойства все еще существуют, есть возможность использовать продукт или его составные части повторно. Для ПО – это заимствование уже разработанного артефакта в новом проекте из существующего.

3. Переработка – продукт разбивается на ключевые элементы, после чего каждый из элементов перерабатывается в ресурс, необходимый для производства. Является основой для повторного использования ПО, особенно для крупных проектов, а также основным методом утилизации ПО.

4. Ликвидация – применяется в том случае, когда продукт нельзя переработать, он физически уничтожается и его нельзя использовать повторно. В случае ПО это означает сдачу проекта в архив или физическое удаление с информационного носителя.

До того момента, как начнется выявление артефактов ПО и их анализ (переработка), необходимо оценить ПО в целом с позиции пригодности его к утилизации. После этого начинается анализ структуры ПО в тех частях, которые пригодны к утилизации. В том случае, когда ПО целиком не пригодно к утилизации, весь проект перемещается в архив, снабженный соответствующими отметками. Анализ структуры ПО позволяет определить прежде всего его ключевые элементы, их взаимосвязь и иерархию, а также наличие тех или иных артефактов.

Среди артефактов ПО с позиции возможности их повторного использования и общей значимости в последующих проектах можно выделить:

1. **Необходимые** (алгоритм, исходный код, документация).
2. **Желательные** (универсальные программные модули, исполняемые файлы общего назначения, дизайн, информационно-логическая схема).
3. **Опциональные** (уникальные программные модули, исполняемые файлы частного назначения, файлы настроек и конфигурации, ТЗ, схема взаимодействия).

Необходимые артефакты характеризуются прежде всего тем, что их легко получить из готового проекта и использовать в новых. При этом современные системы автоматизации разработки ПО позволяют из данных артефактов получать (генерировать) новые. Так, к примеру, имея алгоритм программы, записанный на языке UML, можно сгенерировать структуру будущего программного кода и про-

екта в целом. Ключевой проблемой для необходимых артефактов ПО для утилизации является необходимость работы оператора, что повышает, в свою очередь, риски возникновения ошибок, связанных с человеческим фактором. Суть проблемы в том, что при получении данных информационных ресурсов автоматизированным способом сложно оценить возможность их повторного использования в силу большого разнообразия форматов хранения и внутренней структуры самих артефактов. Оператор же сможет наделить артефакты необходимыми тэгами, пояснениями и комментариями, объясняющими их роль, суть и структуру, что значительно упростит их поиск и повторное использование в дальнейшем.

Желательные артефакты отличаются от необходимых тем, что по своей сути представляют уже законченный продукт, в который весьма сложно вносить изменения. Успешность и целесообразность внесения изменения в желательные артефакты определяются целиком и полностью возможностями формата, в котором они хранятся. На практике это означает, что данные информационные ресурсы используются лишь в том виде, в каком они были заложены в проекте изначально. Данные артефакты характеризуются прежде всего ограничениями, которые в них заложены.

Наконец, опциональные артефакты представляют собой узкоспециализированные информационные ресурсы, повторное их использование даже теоретически маловероятно. На практике они используются повторно лишь тогда, когда рассматриваемый программный продукт является одним из нескольких, разрабатываемых в рамках одного большого проекта.

Построение модели исходного ПО

ПО как информационный проект на стороне заказчика представлен в виде совокупности документации, репозиториев исходного кода, поставляемых заказчику дистрибутивов. Документация хранится в файловой системе ЭВМ разработчика как совокупность каталогов с цифровыми обозначениями. В каждом каталоге в зависимости от шифра хранятся те или иные компоненты и фрагменты артефакта. Структура каталогов определяется по ГОСТ 19.101-77, обозначения регламентируются ГОСТ 19.103-77 и Единой системой программной документации (ЕСПД) в целом. При этом каждый из артефактов описывается или содержится в одном или нескольких документах ЕСПД. Обозначим множество всех документов ПО, оформленного по ЕСПД, через $D : D = (D_1 ... D_i ... D_N)$.

Множество репозиториев исходного кода представляет собой совокупность проектов программных средств и компонентов, реализованных на языках программирования и хранящихся на сервере в системе контроля версий, либо просто совокупность каталогов. Определение принадлежности репозитория к тому или иному ПО и его состав описаны в документации по ПО. Обозначим множество всех репозиториев ПО через $R : R = (R_1 ... R_j ... R_O)$, при этом $O < N$, для каждого репозитория R_j из R существует такой документ D_i из D, что реализуется отображение $M : R_j \rightarrow D_i$.

Дистрибутив ПО представляет собой совокупность программных компонентов и средств, полученных из репозиториев исходного кода, упакованных в удобном для установки на клиентские машины виде. В том случае, когда ПО разрабатывается целиком и полностью с нуля, мощность множества программных компонентов и средств в репозитории равна мощности репозиториев программного кода. В противном случае, множество программных компонентов и средств мощнее множества репозиториев. Обозначим множество программных компонентов и средств через $S : S = (S_1...S_k...S_P)$, при этом $P \geq O$, существуют такие S_k из S, что для всех R_j из R реализуется отображение $N : R_j \rightarrow S_k$.

Таким образом, модель ПО в том виде, в котором оно разрабатывается и хранится у разработчика, можно представить следующим образом:

$$SW = F(D_1...D_N, R_1..R_O, S_1...S_P, M, N). \quad (1)$$

Анализ состава и структуры ПО

После определения пригодности ПО для утилизации необходимо определить его состав и взаимосвязи. Вне зависимости от архитектуры ПО в его составе можно выделить следующие артефакты:

– алгоритм работы ПО;
– информационно-логическая схема ПО;
– схема взаимодействия с другими программными компонентами;
– документация, описывающая ПО;
– техническое задание;
– универсальные программные модули (библиотеки, шаблоны);
– уникальные программные модули;
– исполняемые файлы общего назначения;
– исполняемые файлы частного назначения;
– файлы настроек (настройки интерфейса, настройки операционной системы, куда интегрируется ПО);
– дизайн интерфейса ПО;
– исходный код.

При этом стоит учитывать, что само ПО и его ресурсы в том виде, в котором они существуют на предприятиях, занимающихся разработкой ПО, неоднородны. В зависимости от наличия ресурсов уже готовое ПО может быть классифицировано по трем основным группам:

1. Полностью документированное ПО. К такому ПО, как правило, относятся актуальные работы, в ходе которых разрабатывается или сопровождается программный продукт. Оно характеризуется наличием алгоритма, исходного кода, дизайна интерфейса, технического задания, документации, схемы взаимодействия и информационно-логической схемы.

2. Среднедокументированное ПО или иначе ПО, хранящееся в архиве. Для такого ПО характерно наличие документации и исходного кода. Алгоритм работы, дизайн интерфейса и информационно-логическая схема интерфейса при этом могут как быть, так и не быть в архиве в наличии.

3. Малодокументированное ПО. Как правило, содержит только исходный код, присланный соисполнителями.

На основании вышеизложенного можно сделать вывод о том, что не каждое ПО можно утилизировать всеми методами. Более того, даже если мы имеем полностью документированное ПО, то это не значит, что мы его можем утилизировать целиком одним единственным методом. Также стоит учитывать, что с позиции разработчика ресурсы неравнозначны по своей роли при разработке нового ПО с использованием уже существующих наработок.

Построение модели утилизированного ПО

Утилизированное ПО в отличие от модели исходного ПО представляет собой совокупность артефактов, их взаимосвязей, а также содержит оценку возможности утилизации как всего проекта в целом, так и его составляющих. Артефакты утилизированного ПО – это информационные ресурсы, характеризующиеся форматом, в котором хранится артефакт, самой информацией (контекстом), хранимой в артефакте, а также перечнем всех ограничений и требований, применяемых к данному виду артефакта. Некоторые артефакты ПО сопровождаются различными документами, иные входят в состав документации других артефактов (например, алгоритм программы можно сгенерировать на основании исходного кода, можно встретить в документации на продукт или он вообще хранится как самостоятельная единица) или проекта в целом (например, алгоритм работы всего проекта содержит описание алгоритмов работы входящих в него компонентов, то есть артефактов), для прочих документация не требуется. Также для иных артефактов, с учетом их взаимосвязи с другими информационными ресурсами, требуется определение входных и выходных параметров артефакта с целью его успешного встраивания в структуру проекта (например, для универсальных программных модулей крайне важно знать точки доступа к ним, а также вид и форму результирующих данных). Кроме того, на артефакт накладываются ограничения в виде требований к среде запуска артефакта (например, исполняемые файлы, скомпилированные под ОС Windows, не будут работать под ОС Linux). Наконец, в результате работы алгоритма утилизации каждый артефакт получит краткие обозначения – ключевые слова (тэги), позволяющие его идентифицировать и отфильтровать среди множества прочих, а также оценку пригодности данного артефакта к повторному использованию в целом. Таким образом, артефакт утилизированного ПО A можно описать в виде следующего объекта:

$$A = \{F, K, P, D', I, O, KW, E^A, U\}, \quad (2)$$

где F – формат артефакта,

K – информация, хранящаяся в артефакте,

P – информация о разработчике артефакта,

D' – множество документов, описывающих артефакт,

I – множество входных параметров,

O – множество выходных параметров,

KW – множество ключевых слов (тэгов), которыми описывается (кодируется) артефакт,

E^A – множество требований к среде запуска артефакта,

U – множество возможных путей утилизации [3].

ПО представляет собой документированную связанную совокупность артефактов, учитывающую среду разработки артефактов и версию ПО, требования к среде запуска и функционирования (как системную, так и аппаратную части), требования к зависимому ПО, а также его архитектуру в целом. Тогда модель утилизированного ПО SW' с учетом формулы (1) примет следующий вид:

$$SW' = F'(A, D'', I^R, L, SW, HW, E, S, V), \quad (3)$$

где A – множество артефактов утилизированного ПО,

D'' – множество документов, описывающих утилизированное ПО,

I^R – интерфейсы (правила), описывающие взаимодействие артефактов ПО,

L – среды разработки артефактов,

SW – множество требований к программной части (к примеру, Acrobat Reader, Flash, браузеры и т. п.),

HW – множество требований к структуре и составу требуемой аппаратной части,

E – множество требований к среде запуска ПО,

S – архитектура ПО,

V – версия ПО.

Важно отметить, что артефакты в модели ПО могут быть как самостоятельными артефактами, так и являться, в свою очередь, подпрограммами, которые могут описываться и моделью артефакта, и моделью ПО.

Схематичное изображение процесса утилизации можно представить в следующем виде (рис. 2):

Таким образом, утилизация ПО сводится к нахождению отображения U:

$$U : SW \rightarrow SW'. \quad (4)$$

При этом необходимо получить максимально возможное число артефактов из исходного ПО и обеспечить максимально возможное количество путей утилизации по каждому из артефактов.

Рис. 2. Схема утилизации

Таким образом, в процессе утилизации можно выделить следующие ключевые этапы:

1. Анализ исходного ПО с позиции возможности его утилизации.
2. Анализ состава и структуры исходного ПО с целью построения модели исходных данных (рис. 1).
3. Поиск и анализ артефактов (таблица).
4. Построение модели утилизированного ПО, оценка эффективности утилизации.

Таблица

Возможные пути преобразования артефактов

Входные артефакты	Выходные артефакты
Исходный код	Алгоритм
	Документация (заготовка)
	Дизайн
	ПМ и ИП
Алгоритм	Исходный код
	Документация (заготовка)
Документация	Алгоритм
	Исходный код
	Дизайн
	Информационно-логическая схема
	ТЗ
	Постановка
	Схема взаимодействия

Эффективность утилизации

Производство современных материальных продуктов – это циклический процесс выполнения одних и тех же операций по преобразованию ресурсов в готовый продукт.

Жизненный цикл технически сложных продуктов при этом длится, как правило, от года до десятков лет, и для современных производств завершается этапом утилизации, при котором снижается экологическое воздействие остатков продукта на окружающую среду, а часть ресурсов, выделенных из продукта, возвращается обратно в производство. Такой порядок закреплен как в отечественных, так и в зарубежных стандартах жизненного цикла изделия [6, 7]. Вместе с тем, согласно стандартам жизненного цикла программного изделия этап утилизации отсутствует, а производство ПО завершается этапами ввода в эксплуатацию и сопровождения, которые длятся от месяца до нескольких лет. Но что же происходит с самими программными продуктами? Программы как продукты более подвержены изменениям в части состава и структуры, нежели продукты материальных про-

изводств. Программные изделия постоянно изменяются, дорабатываются, перерабатываются, при этом, независимо от выбранной архитектуры, состав ПО (как разрабатываемого, так и готового продукта) всегда один и тот же. Данный состав и его структура описаны в стандартах [1, 2].

Стоит отметить, что программы – результаты интеллектуального труда, а значит, в них нет материальных ресурсов, которые могут навредить природе, но вместе с тем ключевой ресурс, расходуемый при производстве программ, – это время. Согласно существующим физическим концепциям время нельзя обратить вспять, переработать или изменить, но это не значит, что его нельзя экономить при производстве нового программного продукта. Вне зависимости от архитектуры, готовое ПО формируется из одних и тех же артефактов, полученных на этапах его жизненного цикла. Тогда утилизацией ПО можно назвать процесс выявления и фиксирования данных артефактов для дальнейшего повторного использования при производстве нового продукта. Разумеется, не все артефакты целесообразно использовать повторно, но в целом выделение и использование готовых артефактов способствует более эффективному производству нового ПО путем снижения временных затрат на производство.

Кроме того, необходимо понимать, что многие средства автоматизации разработки и преобразования ПО и его артефактов являются платными. То есть могут потребоваться определенные затраты на приобретение и установку дополнительного ПО, необходимого для утилизации.

Также эффективное использование данных средств предполагает наличие достаточной квалификации разработчика ПО, требуемых навыков и знаний. Это также выражается во временных и стоимостных затратах. Также очевидно, что объем работ по утилизации, выполняемых в ручном режиме, подлежит оплате и что эффективность утилизации напрямую зависит от объема работ. Помимо этого, в отдельных случаях работы алгоритма утилизации ПО мы получим артефакты, которые не подходят к использованию, в силу того что формат их хранения устарел или не соответствует современным требованиям, поэтому также необходимо оценивать возможность преобразования формата артефакта в актуальный вид.

Заключение

Построенная модель утилизированного ПО и описанный алгоритм утилизации позволяют ввести новый этап в жизненный цикл разработки программных продуктов, тем самым повышая эффективность разработки нового ПО, а именно:

– снизить время на разработку;

– уменьшить вероятность появления новых ошибок, используя уже отлаженные и проверенные артефакты;

– снизить затраты на разработку;

– автоматизировать этап переработки ПО.

СПИСОК ЛИТЕРАТУРЫ

1. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания.

2. ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств.

3. Трясцин В.В. Особенности разработки программного обеспечения с использованием существующих наработок // Ученые записки Ульяновского государственного университета. Серия «Математика и информационные технологии». Вып. 1(4) / Под ред. проф. А.А. Смагина. – Ульяновск, УлГУ, 2012. – 286 с. – С. 262–269.

4. Крачтен Ф. Введение в Rational Unified Process. – 2-е изд. : пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 240 с.

5. Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы = The mythical Man-Month: Essays on Software Engineering. – Санкт-Петербург : Символ-Плюс, 2010. – 304 с.

6. ГОСТ 22487-77. Проектирование автоматизированное. Термины и определения.

7. ИСО 9004-1-94. Управление качеством и элементы системы качества.