УДК 004.04

А. С. Добрынин, Р. С. Койнов, С. М. Кулаков

МОДЕЛЬ НЕПОЛНОГО ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разработка крупных программных проектов требует значительных усилий. Известные нормативные модели полного жизненного цикла, применяющиеся крупными компаниями, охватывают все стадии и этапы жизни проекта и, в конечном счете, значительно увеличивают затраты на разработку. По мере роста сложности проектов возникает необходимость в использовании технологического процесса, который позволит более гибко реагировать на запросы заказчиков и уменьшить затраты. Рассматривается модель неполного жизненного цикла программно-технического обеспечения, основанная на итерациях и тестировании, пригодная для использования мобильными коллективами программистов в условиях жесткой оптимизации затрат и ресурсов. Приведенный подход опирается на принятую в системном анализе входвыходную модель «черного ящика», которая приобретает функциональность в процессе разработки в соответствии с бизнес-требованиями. Главное отличие от других нормативных моделей жизненного цикла программного обеспечения заключается в акцентировании внимания разработчиков на постоянно меняющихся потребностях заказчиков. Учитываются ситуации, когда все заинтересованные стороны непрерывно уточняют свое видение архитектуры и функций системы. Следуя закономерным тенденциям развития программного обеспечения, предлагаемая модель неполного жизненного цикла опирается на элементы гибкой разработки, в частности, разработки, управляемой тестированием, когда прикладной инструментарий позволяет органично встраивать в проекты модульные тесты.

Ключевые слова: жизненный цикл, модель, версия, программное обеспечение.

Введение

Многие формализованные подходы, в частности спиральная и каскадная (водопадная) модель [1], предусматривают последовательное прохождение программного продукта по определенным этапам жизненного цикла (ЖЦ): постановка задачи, анализ рисков, анализ требований, построение проектных спецификаций, проектирование, реализация, тестирование, ввод в эксплуатацию и сопровождение, вывод из эксплуатации. Такие громоздкие нормативные модели в условиях оперативных изменений требований заказчиков на практике создают проблемы, для решения которых изначально не были предназначены. Зачастую снижение производительности коллектива разработчиков влечет за собой изменение графика работ и увеличение бюджета проекта. Сама идеология предварительного алгоритмического проектирования изначально предполагает отсутствие гибкости, «монолитность» документации и первоначального кода, т. к. разработчики не всегда могут оперативно реагировать на изменения требований заказчика. К сожалению, многие коллективы разработчиков считают, что процесс создания программного обеспечения (ПО) пока еще является недостаточно всеобъемлющим и формализованным, тем самым они способствуют неконтролируемому росту разнообразных стандартов проектирования, диаграмм, нотаций и спецификаций. Так, например, стандарт UML 2.0 (Unified Modeling Language) [2] предусматривает возможность использования 14 типов диаграмм, из которых на практике обычно используется не больше половины.

Недостатки применения моделей полного жизненного цикла

Современная разработка сложного ПО и объектно-ориентированный подход предполагают в первую очередь наличие гибкой и масштабируемой архитектуры, способной оперативно реагировать на изменения потребностей заказчика в условиях, когда сам заказчик не всегда способен четко сформулировать требования к системе на этапе проектирования. Действительно, откуда заказчик может взять требования к пользовательскому интерфейсу приложения, когда обычно на практике эти требования должны эволюционировать в соответствии с пожеланиями постоянных пользователей? Откуда заказчик может знать заранее, какой интерфейс ему будет нужен?

Более того, недостаточная обратная связь на стадии программирования ведет к неконтролируемому росту затрат при сопровождении. Очевидно, что большинство моделей полного ЖЦ в достаточной степени избыточны, неэффективны и слишком дороги для использования в условиях конкурирующего и динамично развивающегося рынка ПО. В конечном счете, документация и диаграммы никогда не смогут устранить архитектурные недостатки дизайна системы, а также учесть постоянное накопление и развитие опыта разработчика.

Гибкие методики разработки ПО

Гибкие методики, такие как Crystal [2], Scrum [3], XP [2, 4], предусматривают быструю поставку начальной версии работающей программы заказчику для тестирования, игнорируя на начальном этапе многие проектные требования и подразумевая последующее расширение функциональности программы. Основная цель – обеспечение непрерывности развития проекта с возможностью оперативного реагирования на изменения требований бизнеса. Тяжеловесные модели полного ЖЦ слишком неуклюжи, чтобы предусмотреть полный пересмотр заказчиком своего «видения» на этапе завершения проекта. Гибкие методики предлагают коллективам разработчиков ПО сконцентрироваться на качестве выпускаемого продукта в ущерб документации и прочим артефактам проектно-процессной деятельности.

Программный продукт выпускается короткими итерациями, каждая из которых заканчивается определенной сборочной версией, которая поступает в эксплуатацию, а также используется заказчиком для уточнения требований и предварительного тестирования. Разработчики и заказчики формируют сроки и бюджет каждой итерации, оценивая предыдущие достигнутые результаты и требуемый прирост функционала от версии к версии. При этом предварительный сбор детальной информации о требованиях к системе, принятый в моделях полного ЖЦ будет пустой тратой времени и сил, поскольку конечная реальность может оказаться другой. Конкретные детали требований со временем могут и должны изменяться в процессе взаимодействия с ответственным представителем заказчика.

В связи с этим в практической деятельности целесообразно оперировать понятиями ЖЦ отдельной сборочной версии или итерации, что позволит сконцентрировать видение заказчиков и разработчиков на функциональном, более узком аспекте деятельности ИТ-коллективов, связанных в первую очередь с разработкой ПО. Таким образом, в гибкой разработке этапы проектирования, программирования и сопровождения непрерывны, неотделимы друг от друга и входят в состав отдельной короткой итерации разработки.

Модель неполного ЖЦ ПО

Предложена модель неполного жизненного цикла, которая основана на последовательном уточнении архитектуры программного комплекса, представленной, исходя из основополагающих принципов системного анализа, в виде «серого ящика» на *i*-й итерации изменения бизнес-требований. Нулевая итерация представляет собой пустую сборочную версию *r*(Ø), компоненты которой не определены. Обозначим одно бизнес-требование как br_m, где $m \in [1, N]$ формирует пространство всех требований заказчика. Таким образом, совокупность бизнес-требований (заказчика) для *i*-й итерации проектных изменений можно описать вектором $\overline{BR}_i$ (Business Requirements), где *i* изменяется от 1 до некоторого *k*, определяющего стадию ввода в эксплуатацию программного продукта. В условиях трасформации требований, когда некоторое их подмножество остается неизменным, а другое «уточняется», измененные требования можно представить как $\Delta\overline{BR}_i$ – уточнение и детализация заказчиком своих пожеланий. Необходимо отметить, что некоторые координаты $\Delta\overline{BR}_i$ вектора изменений равны нулю, т. к. изменения соответствующего br_m не требуются. Для каждого $\Delta\overline{BR}_i$ необходимо произвести рефакторинг кода и выполнить тестирование, поэтому вектору $\Delta\overline{BR}_i$ однозначно соответствует вектор $\overline{\Delta TF_i}$ на *i*-м шаге уточнения требований.

На вход ящика поступают изменения $\Delta\overline{BR}_i$, обусловленные текущими требованиями заказчика и оформленные в виде новых бизнес-потребностей $\overline{BR}_i$ – самых общих соображений по поводу функциональности, без фиксации конкретных деталей. Выходы «серого ящика» $\overline{TF}_i$ отражают прохождение (или непрохождение) тестов функциональности, используемых для актуализации архитектуры системы в рамках необходимых требований. Таким образом, на каждой итерации происходит уточнение «черного ящика» до тех пор, пока он не превращается в «белый ящик», пригодный для промышленного использования (рис. 1).

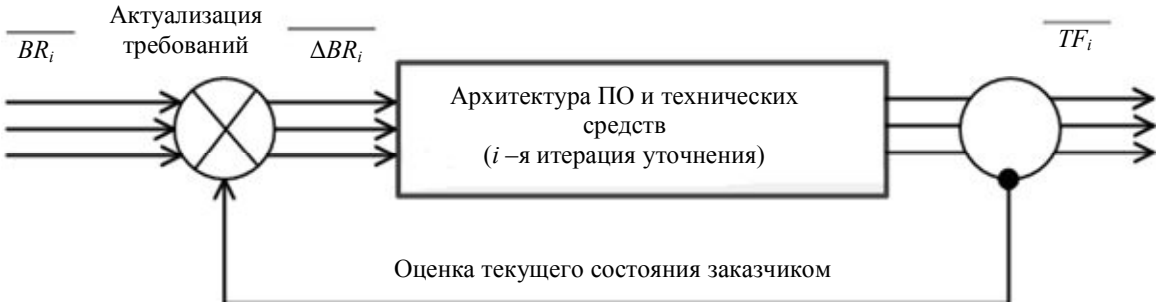

Рис. 1. Модель неполного ЖЦ итерации сборочной версии

Обозначим $R^p = \{r_i \mid i \in I\}$ как множество сборочных версий одного проекта, последовательно формируемых разработчиком в процессе уточнения бизнес-требований заказчика; p – номер проекта.

Также обозначим одну сборку как r_i, тогда начальную сборку как $r(\varnothing)$. Таким образом, начальную итерацию можно рассматривать как реализацию изменений требований относительно «черного ящика». На рис. 2 показаны два проекта, для каждого из которых наблюдается собственная линия сборок.

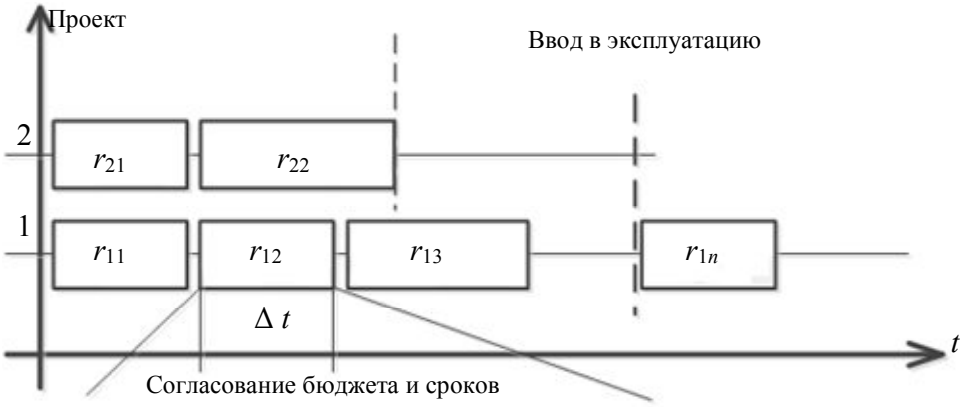

Рис. 2. Итерационная модель ЖЦ сборочной версии

По аналогии с версионной моделью ЖЦ услуги, представленной в [5], приведем итерационную модель ЖЦ сборочной версии ПО. Итерационная модель услуги описывает эволюцию отдельных версий сервиса по стадиям ЖЦ ITIL [6], к которым относится стратегия, проектирование, трансформация и эксплуатация услуг. Предлагаемая нами модель, в отличие от версионной [5], позволяет с максимальной степенью адаптировать разработку ПО под постоянно изменяющиеся потребности заказчика в условиях значительной неопределенности.

Модель ЖЦ сборочной версии опирается на современные тенденции в индустрии ПО, в частности, на управляемую модель на основе разработки [2, 4], паттерны проектирования [2] и поддерживается большинством сред разработки промышленного масштаба, таких как MS Visual Studio и Oracle JDeveloper. Процедура последовательного улучшения программно-технического обеспечения включает в себя учет добавленных (или изменяющихся) бизнес-требований для каждой i-й итерации формирования очередной сборочной версии (рис. 3).

Неполная модель ЖЦ ПО, опробованная нами в проектах [7–9], подходит для разработки крупных программных систем в условиях оптимизации затрат. Сравнение подходов к разработке производилось на основе двух подобных проектов, выполненных в 2014 г. в Сибирском государственном индустриальном университете.

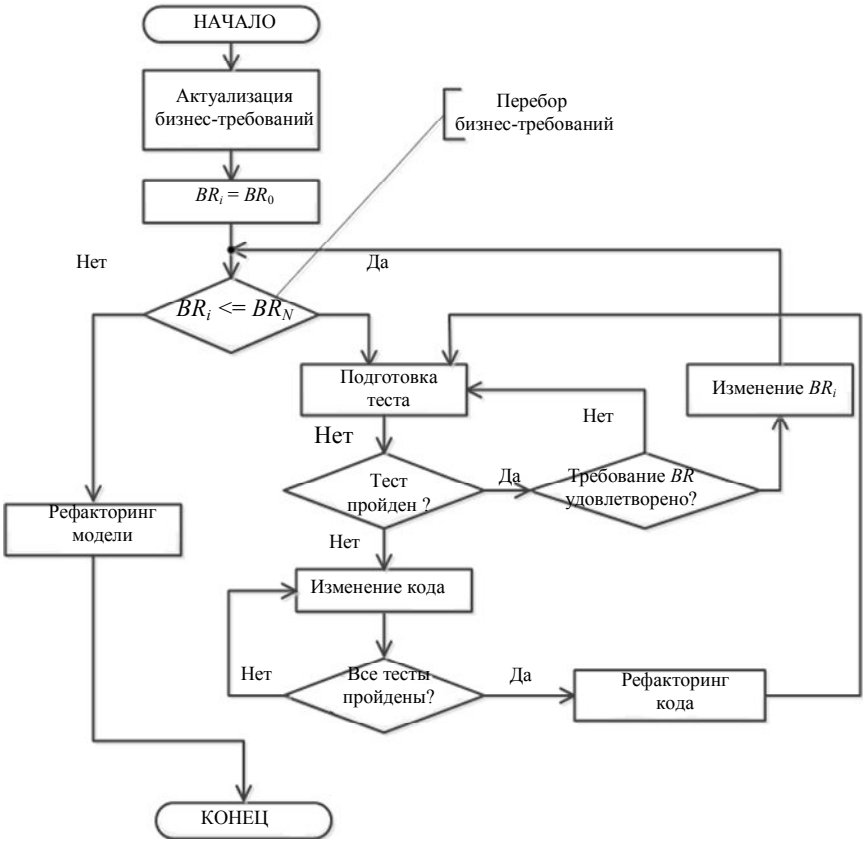

Рис. 3. Процедура последовательного (итерационного) улучшения ПО

Первый проект использовал спиральную модель полного ЖЦ, в котором наряду со штатными сотрудниками принимали участие сторонние разработчики. Он представляет собой web-сайт университета (http://www.sibsiu.ru), выполненный на базе системы управления CMS 1C «Битрикс». Проект прошёл множество (около 20) полных спиралей разработки:

- оценка и разрешение рисков;
- определение целей;
- разработка и тестирование;
- планирование и т. д.

А также все необходимые этапы:

- написание технического задания;
- выбор, закупка и установка CMS;
- разработка макета дизайна (сторонние специалисты);
- вёрстка шаблона;
- разработка инфоблоков и дополнительный модулей для CMS «Битрикс»;
- наполнение сайта (перенос информации со старого сайта);
- обучение персонала;
- написание отчёта.

Общее время разработки составило около 6 месяцев при работе двух программистов, одного дизайнера, двух контент-менеджеров и сторонних специалистов.

Второй проект – система мониторинга эффективности деятельности университета (http://monitoring.sibsiu.ru) – был выполнен полностью «с нуля» силами штатных разработчиков с применением модели неполного ЖЦ. В связи с мобильностью группы разработки и близостью к заказчику у разработчиков была возможность постоянного уточнять цель и задачи, что позволило избежать большого количества итераций разработки. Проект прошёл примерно 12 итераций уточнения требований и следующие этапы:

- первичная постановка цели;
- разработка макета дизайна;

– вёрстка шаблона;
– разработка функционала;
– обучение персонала;
– написание отчёта.

Техзадание было кратким, отражающим только цели разрабатываемой системы, поэтому степень неопределённости и риски срыва сроков проекта были высоки. Вся работа была выполнена за 4 месяца силами трех человек.

Заключение

Дано описание модели неполного ЖЦ программно-технического обеспечения, основанной на итерациях и тестировании, пригодной для использования мобильными коллективами программистов в условиях оптимизации затрат и ресурсов. Приведенный подход опирается на принятую в системном анализе вход-выходную модель «черного ящика», которая приобретает функциональность в процессе разработки в соответствии с бизнес-требованиями.

СПИСОК ЛИТЕРАТУРЫ

1. *Рассел Дж.* Жизненный цикл программного обеспечения / Дж. Рассел // Bookvika Publishing, 2012. 89 с.
2. *Мартин Р. С.* Agile software development / Р. С. Мартин, Дж. В. Ньюкирк, Р. С. Косс // Principles, Patterns and Practicies Principles, Patterns and Practicies. М.: Вильямс, 2004. 752 с.
3. *Cohn Mike.* Scrum: Succeedind with Agile: Software Development Using Scrum (Addison-Wesley Signature Series). М.: Вильямс, 2011. 576 с.
4. *URL:* http://www.jot.fm/books/review7.pdf.
5. *Зимин В. В.* Основы управления жизненным циклом сервиса систем информатики и автоматизации (лучшие практики ITIL): учеб. пособие / В. В. Зимин, А. А. Ивушкин, С. М. Кулаков, К. А. Ивушкин. Кемерово: Кузбассвузиздат, 2013. 500 с.
6. *OGC-ITIL* V3-6. Service Lifecycle. Introduction ITIL TSO, 2007. 173 p.
7. *Добрынин А. С.* О формировании комплекса инструментальных средств ИТ-провайдера для построения расписаний процесса внедрения сервиса / А. С. Добрынин, С. М. Кулаков, В. В. Зимин, Н. Ф. Бондарь // Научное обозрение. 2013. № 8. С. 93–101.
8. *Койнов Р. С.* Об использовании принципа согласованного управления в задачах внедрения ИТ-сервиса / Р. С. Койнов, А. С. Добрынин, С. М. Кулаков, В. В. Зимин // Вестн. развития науки и образования. 2013. № 6. С. 23–27.
9. *Добрынин А. С.* Формирование расписаний в задачах временного планирования / А. С. Добрынин, С. М. Кулаков, Р. С. Койнов, А. В. Грачёв // Вестн. Астрахан. гос. техн. ун-та. Сер.: Управление, вычислительная техника и информатика. 2014. № 4. С. 103–111.

Статья поступила в редакцию 02.12.2014,
в окончательном варианте – 30.03.2015

ИНФОРМАЦИЯ ОБ АВТОРАХ

Добрынин Алексей Сергеевич – Россия, 654007, Новокузнецк; Сибирский государственный индустриальный университет; старший преподаватель кафедры «Автоматизация и информационные системы»; serpentfly@mail.ru.

Койнов Роман Сергеевич – Россия, 654007, Новокузнецк; Сибирский государственный индустриальный университет; старший преподаватель кафедры «Автоматизация и информационные системы»; koynov_rs@mail.ru.

Кулаков Станислав Матвеевич – Россия, 654007, Новокузнецк; Сибирский государственный индустриальный университет; д-р. техн. наук, профессор; зав. кафедрой «Автоматизация и информационные системы»; kulakov-ais@mail.ru.

A. S. Dobrynin, R. S. Koynov, S. M. Kulakov

MODEL OF INCOMPLETE LIFE CYCLE OF SOFTWARE

Abstract. The development of large software projects requires considerable efforts. The known regulatory models of the entire life cycle, which are used by large companies, cover all the stages and phases of the life project and, ultimately, significantly increase the development costs. With the growth of the complexity of the projects, there is a necessity to use the technological process that imposes on the activities of the technical staff and decreases the costs. The article considers a model of incomplete life cycle of software and hardware based on the iterations and testing suitable for use by mobile teams of programmers in a highly optimizing costs and resources. The above-mentioned approach is based on the adopted in the system analysis input-output model of the "black box", which is acquires functionality during the process of development in line with the business requirements. The key difference from all previously considered normative models of the software life cycle is to make the developers focus on the ever-changing needs of the customers. The situations, when all the stakeholders clarify their vision of architecture and functions of the system, are taken into account. Following the natural tendencies of the development of the software, the presented model of partial life cycle is based on the elements of agile development, in particular, test-driven development, when the applied tool allows you organically to embed modular tests into the projects.

Key words: life cycle, model, release, software.

REFERENCES

1. Rassel Dzh. *Zhiznennyi tsikl programmnogo obespecheniia* [Life cycle of software]. Bookvika Publishing, 2012. 89 p.

2. Martin R. S., N'iukirk Dzh. V., Koss R. S. *Agile software development. Principles, Patterns and Practicies*. Moscow, Vil'iams Publ., 2004. 752 p.

3. Kon Maik. *Scrum: Succeedind with Agile: Software Development Using Scrum* (*Addison-Wesley Signature Series*). Moscow, Vil'iams Publ., 2011. 576 p.

4. *Available at*: http://www.jot.fm/books/review7.pdf.

5. Zimin V. V., Ivushkin A. A., Kulakov S. M., Ivushkin K. A. *Osnovy upravleniia zhiznennym tsiklom servisa sistem informatiki i avtomatizatsii (luchshie praktiki ITIL)* [The basis of control of life cycle of the information and automation system services]. Kemerovo, Kuzbassvuzizdat, 2013. 500 p.

6. OGC-ITIL V3- 6. *Service Lifecycle. Introduction ITIL TSO*, 2007. 173 p.

7. Dobrynin A. S., Kulakov S. M., Zimin V. V., Bondar' N. F. O formirovanii kompleksa instrumental'nykh sredstv IT-provaidera dlia postroeniia raspisanii protsessa vnedreniia servisa [On formation of the complex of the technical tools of IT-provider to construct the schedules of the service introduction process]. *Nauchnoe obozrenie*, 2013, no. 8, pp. 93–101.

8. Koinov R. S., Dobrynin A. S., Kulakov S. M., Zimin V. V. Ob ispol'zovanii printsipa soglasovannogo upravleniia v zadachakh vnedreniia IT-servisa [On using the principle of the coordinated control in tasks of IT-service introduction]. *Vestnik razvitiia nauki i obrazovaniia*, 2013, no. 6, pp. 23–27.

9. Dobrynin A. S., Kulakov S. M., Koinov R. S., Grachev A. V. Formirovanie raspisanii v zadachakh vremennogo planirovaniia [Formation of the schedules in tasks of temporary planning]. *Vestnik Astrakhanskogo gosudarstvennogo tekhnicheskogo universiteta. Seriia: Upravlenie, vychislitel'naia tekhnika i informatika*, 2014, no. 4, pp. 103–111.

The article submitted to the editors 02.12.2014,
in the final version – 30.03.2015

INFORMATION ABOUT THE AUTHORS

Dobrynin Alexey Sergeevich – Russia, 654007, Novokuznetsk; Siberian State Industrial University; Senior Lecturer of the Department "Automation and Information Systems"; serpentfly@mail.ru.

Koynov Roman Sergeevich – Russia, 654007, Novokuznetsk; Siberian State Industrial University; Senior Lecturer of the Department "Automation and Information Systems"; koynov_rs@mail.ru.

Kulakov Stanislav Matveevich – Russia, 654007, Novokuznetsk; Siberian State Industrial University; Doctor of Technical Sciences, Professor; Head of the Department "Automation and Information Systems"; kulakov-ais@mail.ru.