УДК 004.05

МОДЕЛЬ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ ПРИ РАЗРАБОТКЕ НА ОСНОВЕ ГИБКИХ МЕТОДОВ

В.В. БАХТИЗИН, С.Н. НЕБОРСКИЙ

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 5 июня 2008

Предлагается модель жизненного цикла программных средств при разработке на основе гибких методов. Процессы жизненного цикла разделены на три стадии: инициация, итеративная реализация требований и сопровождение. Итеративная реализация требований составляет основу предлагаемой модели, причем выделяются итерации с поставкой программных средств заказчику и внутренние итерации. Предлагается разрабатывать программные средства не на основе одного конкретного гибкого метода, а на основе нескольких. Выбрано необходимое подмножество гибких методов, состоящее из гибкого моделирования, экстремального программирования и разработки, ведомой функциями. Приведены аргументы в поддержку этого выбора.

Ключевые слова: жизненный цикл программных средств, модель жизненного цикла, гибкие методы разработки.

Введение

Базовым стандартом в области жизненного цикла (ЖЦ) программных средств (ПС) является стандарт ISO/IEC 12207. В Беларуси этот стандарт введен в действие в 2004 г. под обозначением СТБ ИСО/МЭК 12207-2003. Данный стандарт определяет наличие следующих основных процессов ЖЦ: заказ, поставка, разработка, эксплуатация и сопровождение [1]. Практически любая модель ЖЦ ПС в той или иной мере описывает процессы из всего набора процессов стандарта путем включения в нее соответствующих работ и задач.

Существуют различные модели процессов ЖЦ ПС, но три из них обычно квалифицируются как фундаментальные: каскадная; инкрементная; эволюционная. Каждая из указанных моделей может быть использована самостоятельно или скомбинирована с другими для создания гибридной модели ЖЦ. Стандарт ИСО/МЭК 12207 определяет общий случай разработки типового проекта. При разработке конкретных проектов может отсутствовать необходимость в тех или иных процессах, работах и задачах. На рис. 1 приведен пример применения стандарта ИСО/МЭК 12207 к модели ЖЦ, основанной на макетировании (прототипировании) небольшой системы [2].

Основой эффективного применения такого макетирования системы является максимально возможная детализация на ранних стадиях проекта (анализ требований к системе и проектирование системной архитектуры). Данные стадии выполняются в ЖЦ один раз. Требования к системе, в первую очередь, функции системы и внешние функции определяются пользователями в начале ЖЦ, деловые процессы уточняются при проведении пользователем серии оценок прототипов системы [3]. Однако далеко не для всех проектов удается зафиксировать требования в самом начале ЖЦ. Зачастую в процессе реализации ПС требования меняются кардинально, от определенной уже существующей функциональности отказываются,

появляются новые требования, реализация которых предполагает изменение текущей архитектуры системы. Гибкие методы разработки как раз и призваны решать подобные проблемы [4].

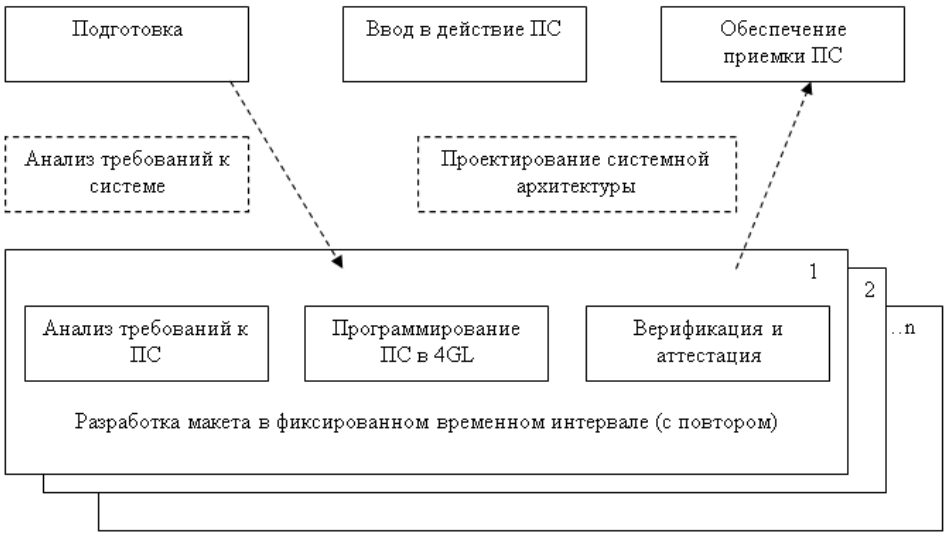

Рис. 1. Пример модели ЖЦ, использующей макетирование,
по ГОСТ Р ИСО/МЭК ТО 15271-2002

Модель жизненного цикла программных средств

В данной работе предлагается модель ЖЦ ПС при разработке на основе гибких методов. Эта модель, по сути, предлагает выделять три стадии разработки: инициация, итеративная реализация требований и сопровождение. На стадии инициации происходит определение концепции или потребности в системе, анализ требований к системе, проектирование системной архитектуры, рассмотрение вариантов реализации. Итеративная реализация требований – это реализация и поставка работающего ПС, где на каждой итерации реализуются новые требования и, возможно, изменяются старые. Стадия сопровождения ПС предполагает анализ обнаруживаемых проблем и внесение необходимых для их устранения изменений.

Предлагаемая модель может быть представлена в виде ориентированного графа $G(X,Y)$, где $X=\{x_i, i=\overline{1,n_X}\}$, $|X|=n_X$ — множество работ ИСО/МЭК 12207, $Y=\{y_j, j=\overline{1,n_Y}\}$, $|Y|=n_Y$ — множество их связей. На рис. 2 приведен общий вид данного графа, где работы $X=\{x_i, i=\overline{1,n_X}\}$ объединены и показаны как одно целое в рамках отдельной стадии.

Количество и типы работ, выполняемых на каждой стадии, могут корректироваться в зависимости от конкретного проекта. Могут быть изменены и связи между ними. Как результат, граф переходов работ стандарта ИСО/МЭК 12207 *G(X,Y)* может стать весьма сложным, что означает неэффективные, неслаженные процессы разработки ПС. В таком случае имеет смысл оптимизировать граф $G(X,Y)$, причем наиболее универсальным критерием оптимизации является критерий минимума числа связей между частями графа. *k*-я часть графа G_k — это подмножество работ стандарта ИСО/МЭК 12207 $X_k \subseteq X$ и их подмножество связей $Y_k \subseteq Y$, образующих стадию в предлагаемой модели: $G_k=(X_k,Y_k)$, $k \in K=\{1,2,...,N_k\}$, где N_k — число стадий реализации ПС. Связи между частями графа называют внешними; связи, соединяющие работы внутри отдельной стадии — внутренними.

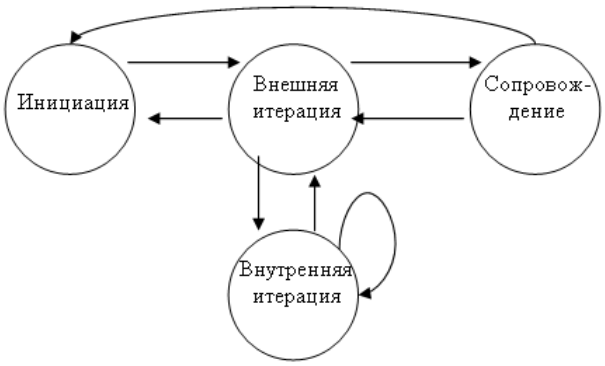

Рис. 2. Модель ЖЦ ПС при разработке на основе гибких методов

Задача нахождения минимального количества внешних связей между отдельными стадиями сводится к оптимальному разбиению графа $G(X,Y)$ на части $G_k=(X_k,Y_k)$, $X_k \subseteq X$, $Y_k \subseteq Y$. Результатом разбиения является совокупность стадий реализации *S(G)*:

$$\begin{aligned} &\forall G_k \in S(G), G_k \neq 0,\ k \in K;\ \forall G_k, G_l \in S(G), \\ &G_k \neq G_l \Rightarrow X_k \cap X_l = 0 \wedge (Y_k \cap Y_l = Y_{k,l} \vee Y_k \cap Y_l = 0), \\ &\bigcup_{k \in K} G_k = G \end{aligned} \quad (1)$$

Выражение (1) определяет подмножество дуг $Y_{k,l} \subseteq Y$, соединяющих вершины частей G_k и G_l графа G. Если количество связей между частями G_k и G_l графа G равно $|Y_{k,l}|=h_{k,l}$, то общее количество внешних связей графа G (дуг, соединяющих подграфы, на которые разбит G):

$$H = \frac{1}{2}\sum_{k=1}^{Nk}\sum_{l=1}^{Nk} h_{k,l}, k \neq l. \quad (2)$$

Для примера можно рассмотреть следующий граф (рис. 3):

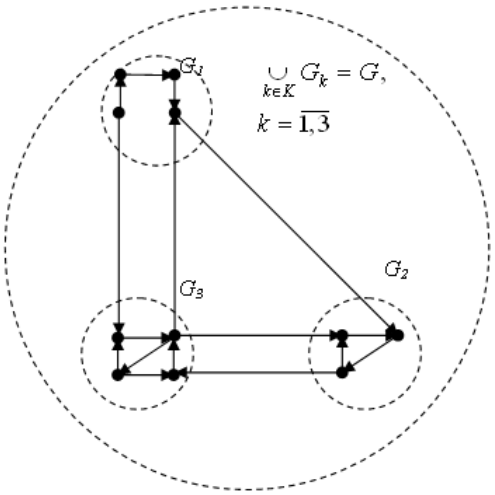

Рис. 3. Пример разбиения графа

Очевидно, количество связей между частями G_1, G_2 и G_3 таково: $h_{1,2}$=1; $h_{1,3}$=2; $h_{2,3}$=2. Тогда общее число внешних связей графа G:H=5. Как правило, всегда существует несколько вариантов разбиения некоторого графа $G(X,Y)$ на части $G_k=(X_k,Y_k)$. Для выбора наилучшего разбиения используется коэффициент разбиения графов:

$$\Delta(G) = \sum_{k=1}^{Nk} |Y_{k,k}| \Big/ \frac{1}{2}\sum_{k=1}^{Nk}\sum_{l=1}^{Nk} |Y_{k,l}| = \sum_{k=1}^{Nk} |Y_{k,k}| \Big/ H,$$

где $Y_{k,k}$ — подмножество внутренних связей.

Количество внешних связей H вычисляется по формуле (2). Сравнивая коэффициенты разбиения $\Delta(G)$, можно подобрать граф с минимальным количеством внешних связей, т.е. такой граф, который будет соответствовать наиболее эффективной организации ЖЦ ПС. Для графа на рис. 3 коэффициент разбиения $\Delta(G) = 2{,}2$. Очевидно, с ростом числа внешних связей H коэффициент разбиения $\Delta(G)$ уменьшается. Следовательно, чем больше коэффициент $\Delta(G)$, тем лучше. Можно нормировать коэффициент $\Delta(G)$ с тем, чтобы он изменялся на отрезке [0, 1]. Тем самым можно ввести метрику разбиения графа Ψ:

$$\Psi = \frac{\Delta G}{1+\Delta(G)}.$$

Условию $\Psi \to 1$ соответствует $\Delta(G) \to \infty, H \to 0$. Решая на практике задачу оптимального разбиения графа, можно посчитать предлагаемую в данной работе метрику разбиения Ψ и определить наилучший в заданных условиях граф.

Предлагаемая в данной работе модель отводит итеративной реализации требований главную роль. Предлагается использовать два вида итераций: итерации с поставкой ПС заказчику и внутренние итерации. Внутренняя итерация предназначена главным образом для команды инженеров по качеству. Внутренние итерации показывают прогресс разработки ПС и удостоверяют отсутствие ошибок в поставляемой заказчику версии ПС.

Результатом внутренней итерации является очередная версия функционирующего ПС. Результатом внешней итерации также является версия функционирующего ПС, причем эта версия является одной из тех, что получается в результате внутренних итераций. Пусть $D = \{d_i \mid i = \overline{1,m}\}$ — множество всех версий ПС (т.е. множество результатов внутренних итераций), d_i — внутренняя версия ПС, m – количество внутренних итераций. Тогда можно определить подмножество D' внешних версий ПС: $D' \subseteq D, |D'| = n$, n — количество внешних итераций, которые были поставлены пользователям.

Для сокращения времени поставки необходимо оптимизировать функцию $T: D \to D'$ поставки очередной версии ПС пользователю $T \to \min$.

При установленном и стабильном процессе разработки количество внутренних итераций m и количество внешних итераций n связаны линейно $m = an$, где a — количество внутренних итераций, образующих одну внешнюю.

Следовательно, минимальное время поставки ПС пользователю будет достигнуто при условии, что внешняя итерация содержит только одну внутреннюю итерацию m=n=1, $D = D' = \{d_1\}$ — множество мощности 1.

Так как на каждой внутренней итерации устраняются найденные в предыдущей версии ПС ошибки и несоответствия требованиям, то результат внешней итерации зависит от количества внутренних итераций. Следовательно, задача усложняется

$$\min_q T: D \to D',$$

где q — качество ПС.

Таким образом, разработчик и заказчик всегда вынуждены искать оптимальное время внешней итерации, результатом которой является ПС с необходимым уровнем качества.

Применение модели жизненного цикла программных средств при разработке на основе гибких методов

Предлагаемая модель предполагает, что разработка ПС ведется на основе гибких методов. Наиболее распространенными гибкими методами на сегодняшний день являются Скрум, разработка, ведомая функциями (FDD), экстремальное программирование (XP), гибкое моделирование (Agile Modeling), динамичный метод разработки систем (DSDM), разработка, ведомая тестами (TDD). Однако вряд ли удастся найти такой метод, который работал бы эффективно независимо от условий проекта. Для разработки ПС в данной работе предлагается использовать три гибких метода: гибкое моделирование, экстремальное программирование и разработку, ведомую функциями (FDD).

Когда бы ни требовалось создать архитектуру ПС, имеет смысл использовать гибкое моделирование. Получаемые модели обладают следующими свойствами: модели доступны для тех, кому они предназначены, но не обязательно для всех; они точны и непротиворечивы; модели представляют практическую ценность; они достаточно детальны и просты [5].

Гибкие методы разработки делают акцент непосредственно на реализации функциональных требований к ПС. Для эффективной реализации функциональных требований необходим набор определенных принципов и практических приемов. Именно поэтому стоит считаться с идеями XP. В основе XP лежат следующие четыре принципа: коммуникация, простота, обратная связь и требовательность к команде разработчиков, состоящей преимущественно из профессионалов [6]. Эти принципы привели к следующим идеям, составляющим основу XP [7]:

1) парное программирование;

2) всегда оставаться на связи с заказчиком;

3) тестирование упреждает разработку;

4) короткие итерации;

5) все должно быть максимально простым;

6) не разрабатывать ничего лишнего, пока не возникнет непосредственное требование заказчика;

7) коллективное владение проектом.

Безусловно, гибкое моделирование и XP способствуют эффективной разработке ПС. Существует даже такой метод, как XP масштаба предприятия (EXP — Enterprise XP), суть которого как раз состоит в объединении XP и гибкого моделирования [4]. Однако ни XP, ни гибкое моделирование не касаются планирования проекта. Но для заказчика ведь важно не только качество ПС, но также время его разработки и бюджет. Без хорошего планирования вероятность успеха любого проекта снижается многократно. FDD добавляет необходимый элемент контроля.

Согласно FDD, функция — это такое требование, реализация которого запланирована и связана с определенной активностью по его реализации. Функции связывают требования с планированием. FDD предполагает наличие жестко определенных временных рамок, которые отводятся на реализацию набора функций. Эти временные рамки, набор функций (запланированных к реализации требований), их приоритеты и определяют итерацию в FDD [8].

Таким образом, в данной работе предлагается разрабатывать ПС на основе:

1) метода гибкого моделирования, что позволяет создать архитектуру ПС;

2) метода XP, что позволяет эффективно разрабатывать ПС;

3) метода FDD, что дает управление проектом.

При постоянно изменяющихся требованиях архитектуру ПС предлагается получать исходя исключительно из требований. При этом должен использоваться метод гибкого моделирования. На его основе должны быть созданы модели, лишь поясняющие архитектуру ПС. Детальная же архитектура должна строиться на основе требований автоматически. Тогда задача получения программной архитектуры будет сведена к получению отображения требований в компоненты программной архитектуры [9]. Требования могут быть заданы в виде множества R:

$$R = \{ r_i \mid i = \overline{1, n_R} \},$$

где n_R — количество требований; $r_i = \{ a_{ij} \mid a_{ij} \in A_j, j = \overline{1, n_A} \}$ — i-ое требование; A_j — множество значений j-го атрибута; n_A — количество атрибутов.

Для задания связей между требованиями R используются матрицы трассируемости или списки трассируемости. Трассируемость требований документирует зависимости и логические связи отдельных требований и других элементов ПС [10]. Матрица трассируемости может быть представлена как квадратная матрица размерностью $n_R \times n_R$:

$$M = [m_{ij}]_{n_R \times n_R},$$

где $m_{ij} = \begin{cases} 1, \text{если } r_j \text{ зависит от } r_i; \\ 0, \text{если } r_j \text{ не зависит от } r_i; \end{cases}$, $r_i, r_j \in R$ — требования.

Отдельное требование или группа логически связанных требований реализуется в виде отдельного компонента ПС. Пусть C — множество таких компонентов:

$$C = \{ c_i \mid i = \overline{1, n_C} \},$$

где c_i — i-й компонент ПС; n_C — количество компонентов ПС.

Необходимо задать отображение множества требований R в множество компонентов ПС C: $F : R \to C$,

или

$$F = \begin{pmatrix} r_1 & r_2 & \dots & r_{n_R} \\ F(r_1) & F(r_2) & \dots & F(r_{n_R}) \end{pmatrix},$$

где $F(r_i) \in C, i = \overline{1, n_R}$.

Таким образом, программная архитектура может быть получена на основе требований. Более того, можно автоматически получать программную архитектуру на основе требований [9]. Такой подход гарантирует целостность (согласованность) требований и их реализации.

Заключение

В данной работе предложена модель ЖЦ ПС при разработке на основе гибких методов. Определены три стадии (инициация, итеративная реализация требований, сопровождение), показана их связь. Предложено разрабатывать ПС на основе трех гибких методов: гибкого моделирования, экстремального программирования и разработки, ведомой функциями. Гибкое моделирование позволяет эффективно создавать архитектуру ПС в условиях изменяющихся требований, причем получаемые модели носят поясняющий характер. Детальная же архитектура создается автоматически как прямое отображение требований в компоненты ПС. Для качественной реализации этих компонентов используются приемы и подходы экстремального программирования. Разработка, ведомая функциями, позволяет управлять созданием ПС. Практическое применение предложенной модели и выбранного подмножества гибких методов позволит эффективно разрабатывать качественные ПС.

SOFTWARE LIFECYCLE MODEL WHEN DEVELOPING WITH AGILE METHODS

V.V. BAKHTIZIN, S.N. NIABORSKI

Abstract

A software lifecycle model when software is being developed with agile methods is introduced. In a nutshell of this model there are three stages: initiation, iterative requirements implementation and support. During iterative requirements implementation stage, there are two types of iterations: external (those which deliver the functioning software to end-users) and internal (those which deliver the functioning software to testing team in order to assure the necessary quality level and understand the development progress). The introduced model is supposed to be used when developing software with agile methods. Instead of using some specific agile development method, it proves the benefits of using a mix of them. To get the right software, three agile methods have been chosen: agile modeling, extreme programming and feature-driven development.

Литература

1. СТБ ИСО/МЭК 12207-2003. Информационная технология. Процессы жизненного цикла программных средств.
2. ГОСТ Р ИСО/МЭК ТО 15271-2002. Информационная технология. Руководство по применению ГОСТ Р ИСО/МЭК 12207 (Процессы жизненного цикла программных средств).
3. *Бахтизин В.В., Глухова Л.А.* Стандартизация и сертификация программного обеспечения: Учеб. пособие Минск, 2006.
4. *Hunt J.* Agile Software Construction. Springer-Verlag London Limited, 2006.
5. *Ambler S., Jeffries R.* Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Wiley; 1st edition, 2002.
6. *Beck K., Andres Ch.* Extreme Programming Explained: Embrace Change 2nd ed., Addison Wesley, 2004.
7. *Burke E., Coyner B.* Java Extreme Programming Cookbook. O'Reilly Media, Inc.; 1st edition, 2003.
8. *Palmer S.R., Felsing J.M.* A Practical Guide to Feature-Driven Development – Prentice Hall, 2002.
9. *Бахтизин В.В., Неборский С.Н.* // Программные продукты и системы № 3 (75), 2006. С. 2–5.
10. *Вигерс К.* Разработка требований к программному обеспечению. 2004.