УДК 650.1

В.В. ЛИТВИНОВ[*], А.А. ЗАДОРОЖНИЙ[*]

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА СОЗДАНИЯ МОДЕЛЕЙ В УСЛОВИЯХ НЕПОЛНОТЫ ДАННЫХ

[*]Черниговский национальный технологический университет, Чернигов, Украина

Анотація. *У роботі розглядається життєвий цикл моделей у порівнянні з життєвим циклом програм. Подібність життєвого циклу моделей, створених з використанням дедуктивного підходу до життєвого циклу програм, дає можливість використовувати при створенні моделей інструментальні засоби і моделі життєвого циклу, що використовуються при створенні програм. Також у роботі запропоновані засоби автоматизованого експериментування й тестування моделей і розглядаються визначення оптимальності дедуктивних моделей.*
Ключові слова: *життєвий цикл моделі, життєвий цикл програми, верифікація моделі, валідація моделі, модульне тестування, модульне експериментування.*

Аннотация. *В статье рассматривается жизненный цикл моделей в сравнении с жизненным циклом программ. Подобность жизненного цикла моделей, созданных с использованием дедуктивного подхода к жизненному циклу программ, дает возможность использовать при создании моделей инструментальные средства и модели жизненного цикла, которые используются при создании программ. Также в статье предложены средства автоматизации экспериментирования и тестирования моделей и рассматриваются условия определения оптимальности модели.*
Ключевые слова: *жизненный цикл модели, жизненный цикл программы, верификация модели, валидация модели, модульное тестирование, модульное экспериментирование.*

Abstract. *This paper considers model life cycle in comparison with program life cycle. The similarity of model life cycle created using deductive approach to program life cycle gives ability of using tools and life cycle model which are used in programming. Tools for automated experimentation and testing of models are considered in this paper as well and the definitions of optimality deductive models are discussed.*
Keywords: *model life cycle, program life cycle, model verification, model validation, unit testing, modular experimentation.*

1. Введение

Термин «объект» в современном смысле появилось в MIT (Massachusetts Institute of Technology) в конце 1950, начале 1960 годов. В среде специалистов по искусственному интеллекту термин «объект» мог относиться к идентифицированным элементам (атомы Lisp) со свойствами (атрибутами). Другим ранним примером ООП (объектно-ориентированного программирования) в MIT был Sketchpad, созданный Иваном Сазерлендом в 1960-61 гг. В глоссарии подготовленного в 1963 г. технического отчета, основанного на его диссертации о Sketchpad, Сазерленд определяет понятия «объект» и «экземпляр» с концепцией классов на основе «мастера» или «определения», хотя все эти термины относились к графическому представлению объектов.

Объекты как формализованный концепт появились в программировании в Simula 67, модернизированной версии Simula I, языка программирования, ориентированного на дискретно-событийное моделирование. Simula разрабатывалась под влиянием SIMSCRIPT и предложенной концепции записей-классов. Simula включала в себя понятие классов и экземпляров (или объектов), а также подклассов, виртуальных методов, сопрограмм и дискретно-событийное моделирование как часть собственной парадигмы программирования. В языке использовался автоматический «сборщик мусора», который был изобретен ранее для функционального языка Lisp. Simula использовалась тогда преимущественно для фи-

зического моделирования. Идеи Simula оказали серьезное влияние на более поздние языки, такие как Smalltalk, варианты Lisp (CLOS), Object Pascal и C++.

Язык Smalltalk фактически навязывал использование «объектов» и «сообщений» как базиса для вычислений. Создателей Smalltalk вдохновляли некоторые идеи Simula, но Smalltalk разрабатывался как полностью динамичная система, в которой классы могут создаваться и изменяться динамически, а не только статически, как в Simula.

Объектно-ориентированное программирование развилось в доминирующую методологию программирования в начале и середине 1990 годов, когда стали широкодоступны поддерживающие ее языки программирования, такие как Visual FoxPro 3.0, C++ и Delphi.

Возможности ООП добавлялись во многие языки того времени, включая Ada, Basic, Fortran, Pascal и др. Их добавление в языки, изначально не разрабатывавшиеся для поддержки ООП, часто приводило к проблемам с совместимостью и поддержкой кода.

Позднее стали появляться языки, поддерживающие как объектно-ориентированный подход, так и процедурный вроде Pythonи Ruby. Пожалуй, самыми коммерчески успешными объектно-ориентированными языками можно назвать Visual Basic.NET, C# и Java. И .NET и Java демонстрируют превосходство ООП. Таким образом, можно считать, что объектно-ориентированный подход и первые объектно-ориентированные инструменты появились благодаря задачам, которые стояли перед моделированием. Использование объектно-ориентированного подхода и его инструментов наилучшим образом подходит для целей моделирования.

Некоторые этапы жизненного цикла программного обеспечения схожи с этапами жизненного цикла (ЖЦ) моделей. Это означает, что инструментальные средства, которые используются на этапах жизненного цикла разработки программного обеспечения, можно использовать на схожих этапах жизненного цикла моделей.

2. Сравнение этапов жизненного цикла моделей и программ

Вероятно, самым распространенным мотивом обращения к понятию жизненного цикла является потребность в систематизации работ в соответствии с технологическим процессом. Этому назначению хорошо соответствует так называемая общепринятая модель жизненного цикла программного обеспечения, согласно которой программные системы проходят в своем развитии две фазы: фазу разработки и фазу эксплуатации и сопровождения [1] (рис. 1).

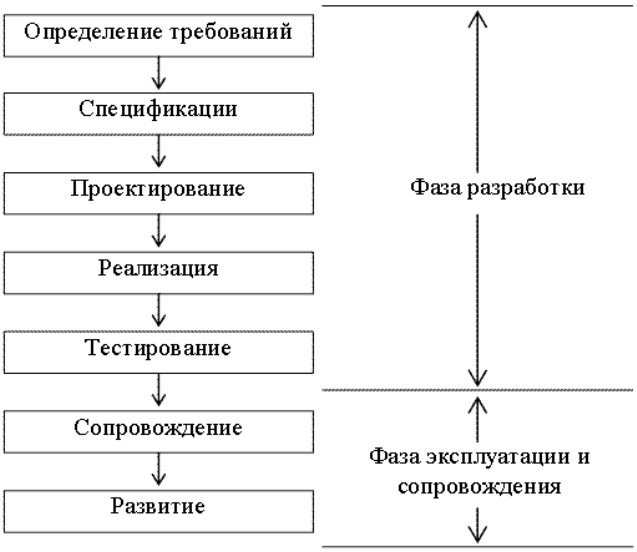

Рис. 1. Этапы жизненного цикла программного обеспечения

Разработка начинается с идентификации потребности в новом приложении, а заканчивается передачей продукта разработки в эксплуатацию.

Первым этапом фазы разработки является постановка задачи и определение требований. Определение требований включает описание общего контекста задачи, ожидаемых функций системы и ее ограничений. На этом этапе заказчик совместно с разработчиками принимает решение о создании системы. Особенно существен этот этап для нетрадиционных приложений.

В случае положительного решения начинается этап спецификации системы в соответствии с требованиями. Разработчики программного обеспечения пытаются осмыслить

выдвигаемые заказчиком требования и зафиксировать их в виде спецификаций системы. Важно подчеркнуть, что назначение этих спецификаций — описывать внешнее поведение разрабатываемой системы, а не ее внутреннюю организацию, то есть отвечать на вопрос, что она должна делать, а не как это будет реализовано. Здесь говорится о назначении, а не о форме спецификаций, поскольку на практике при отсутствии подходящего языка спецификаций, к сожалению, нередко приходится прибегать к описанию «что» посредством «как». Прежде чем приступать к созданию проекта по спецификациям, они должны быть тщательно проверены на соответствие исходным целям, полноту, совместимость (непротиворечивость) и однозначность.

Проблемы языка спецификаций не в том, что нельзя (или трудно) строго и четко описать, что требуется в проекте. В большей степени они связаны с необходимостью добиваться и поддерживать соответствие описания «что» нечетким, неточным и часто противоречивым требованиям со стороны внешних по отношению к проекту людей. Нет оснований полагать, что эти люди будут знакомы с «самым хорошим языком спецификаций», что они будут заботиться о корректности своих требований. Задача этапа спецификаций в том и состоит, чтобы описание программы выстроить в виде логически выверенной системы, понятной как для заказчика данной разработки, будущих пользователей, так и для исполнителей проекта.

Разработка проектных решений, отвечающих на вопрос, как должна быть реализована система, чтобы она могла удовлетворять специфицированным требованиям, выполняется на этапе проектирования. Поскольку сложность системы в целом может быть очень большой, главной задачей этого этапа является последовательная декомпозиция системы до уровня очевидно реализуемых модулей или процедур.

На следующем этапе реализации или кодирования каждый из этих модулей программируется на наиболее подходящем для данного приложения языке. С точки зрения автоматизации этот этап традиционно является наиболее развитым.

В рассматриваемой модели фаза разработки заканчивается этапом тестирования (автономного и комплексного) и передачей системы в эксплуатацию.

Фаза эксплуатации и сопровождения включает в себя всю деятельность по обеспечению нормального функционирования программных систем, в том числе фиксирование вскрытых во время исполнения программ ошибок, поиск их причин и исправление, повышение эксплуатационных характеристик системы, адаптацию системы к окружающей среде, а также, при необходимости, и более существенные работы по совершенствованию системы. Все это дает право говорить об эволюции системы. В связи с этим фаза эксплуатации и сопровождения разбивается на два этапа: собственно сопровождение и развитие. В ряде случаев на данную фазу приходится большая часть средств, расходуемых в процессе жизненного цикла программного обеспечения.

Понятно, что внимание программистов к тем или иным этапам разработки зависит от конкретного проекта. Часто разработчику нет необходимости проходить через все этапы, например, если создается небольшая, хорошо понятная, программа с ясно поставленной целью. Проблемы сопровождения, плохо понимаемые разработчиками небольших программ для личного пользования, являются в то же время очень важными для больших систем.

Такова краткая характеристика общепринятой модели. В литературе встречается много вариантов, развивающих ее в сторону детализации и добавления промежуточных фаз, этапов, стадий и отдельных работ (например, по документированию и технологической подготовке проектов) в зависимости от особенностей программных проектов или предпочтений разработчиков.

Рассмотрим этапы жизненного цикла модели. Основой моделирования является методология системного анализа [2]. Это дает возможность исследовать систему или процесс

с использованием технологии операционного исследования, включая следующие взаимосвязанные этапы (рис. 2).

Рис. 2. Этапы жизненного цикла модели

На первом этапе заказчик формулирует проблему. Организовываются встречи руководителя проекта с заказчиком, аналитиками по моделированию и экспертами по проблеме, которая изучается. Изучаются цели исследования и специальные вопросы, ответы на которые будут получены по результатам исследования; устанавливаются критерии оценивания работы, которые используются для изучения эффективности различных конфигураций системы; рассматриваются такие показатели системы, как масштаб модели, период исследований и необходимые ресурсы; определяются конфигурации системы, а также необходимое программное обеспечение.

На этом же этапе проводится целенаправленное исследование моделируемой системы или процесса, привлекаются эксперты по проблеме, которые владеют достоверной информацией. Собирается информация о конфигурации системы, и рассматриваются способы эксплуатации для определения параметров модели и выходных распределений вероятностей.

На втором этапе разрабатывается концептуальная модель – абстрактная модель, которая дает возможность выявить причинно-следственные связи, свойства исследуемого объекта в пределах, определенных целями исследования. По сути, это формальное описание объекта моделирования, который отображает концепцию (взгляд исследователя на проблему). Она включает в явном виде логику, алгоритмы, предположения и ограничения.

В соответствии с целями моделирования определяются выходные значения, которые необходимо собирать во время моделирования, степень детализации, необходимые входные данные для моделирования.

Уровень детализации модели зависит от таких факторов: цели проекта, критерии оценки показателей работы, доступность данных, достоверность результатов, компьютерные возможности, мнения экспертов по решаемой проблеме, ограничения, связанные со временем и финансированием. Производится структурный анализ концептуальной модели, предлагается описание допущений, которые обговариваются с заказчиком, руководителем проекта, аналитиками и экспертами по решаемой проблеме.

Разрабатываются модели входных данных, проводится их статистический анализ, по результатам которого определяется распределение вероятностей, регрессионные, корреляционные и другие зависимости. На этом этапе для предварительного анализа данных широко используются различные пакеты, такие как Statistica.

Для динамических систем проводится предварительный анализ функционирования моделируемой системы с детальным описанием работы элементов системы. По результатам такого анализа можно узнать, можно ли решить проблему без использования моделирования. Детально проработанная концептуальная модель дает возможность заказчику с другой стороны посмотреть на работу системы и, например, определить узкие места системы, которые вызывают снижение ее пропускной способности.

Одна из наиболее сложных проблем, решаемых аналитиком моделирования, состоит в определении, является ли модель адекватной системе. Если имитационная модель адекватна, ее можно использовать для принятия решений к системе, которую она представляет, как будто она принималась на основании результатов проведения экспериментов с реальной системой. Модель сложной системы может только приблизительно отвечать оригиналу, независимо от того, сколько усилий было затрачено, потому что абсолютно адекватных моделей не существует.

Поскольку модель всегда должна разрабатываться для определенного множества целей, то модель, которая является адекватной для одной цели, может не быть таковой для исследования другой цели. Следует отметить, что адекватная модель не обязательно является достоверной и наоборот. Модель может быть достоверной, но в этом случае не использоваться для принятия решений. Например, достоверная модель не может быть адекватной по политическим или экономическим причинам.

После каждого из вышеуказанных этапов проверяется достоверность модели. Проверку условно можно разделить на два этапа: проверка корректности создания концептуальной модели – валидация; проверка корректности реализации модели – верификация. Во время проверки достоверности необходимо ответить на вопрос о соответствии модели моделируемой системе, то есть определить, в какой мере модель и система изоморфны. Как правило, в случае моделирования требование изоморфизма модели и объекта чрезмерны, потому что в этом случае сложность модели должна соответствовать сложности объекта. Поэтому и строят гомоморфные модели, в которых выполняется условие однозначного соответствия модели объекту. На этапе верификации рассматривают, правильно ли преобразована концептуальная модель (модельные предположения) на компьютерную программу, то есть производят настройку программы моделирования. Это сложное задание, поскольку может существовать множество логических путей. Этап проверки корректности реализации модели включает проверку эквивалентности преобразования модели на каждом из этапов ее реализации и сравнение состояний. В этом случае модель претерпевает такие изменения: концептуальная модель – математическая модель – алгоритм моделирования – программная реализация модели.

Валидация – это процесс, который дает возможность определить, является ли модель (а не компьютерная программа) точным отображением системы для конкретных целей исследований. Разрабатывают план проведения экспериментов с моделью для достижения поставленных целей. Основная цель планирования экспериментов – изучение поведения моделируемой системы при наименьших затратах во время экспериментирования. Наиболее часто проводят следующие эксперименты:

• сравнивают средние значения и дисперсии разных альтернатив;

• определяют важность учета влияния переменных и ограничений, которые накладываются на эти переменные;

• определяются оптимальные значения с определенного множества возможных значений переменных.

Проведение экспериментов планируют для поиска незначимых факторов. В случае оптимизации какого-нибудь числового критерия формируют гипотезу относительно выбора наилучших вариантов структур моделируемой системы или режимов ее функционирования, определяют диапазон значений параметров (режимов функционирования) модели, в пределах которого находится оптимальное решение. Определяют количество реализаций и время прогона модели каждой реализации. Проводят экстремальный эксперимент, по результатам которого находят оптимальное значение критерия и соответствующие значения параметров. Для оценки точности стохастических моделей строят доверительные интервалы для получения выходных переменных.

Далее анализируют и оценивают результаты. Приводят результаты компьютерных экспериментов в виде графиков, таблиц, распечаток, а также оценивают качественные и количественные оценки результатов моделирования. Для иллюстрации модели используют анимацию. Обсуждают процесс создания модели и ее достоверность, чтобы повысить уровень доверия к ней.

По полученным результатам формируют выводы по проведенным исследованиям и определяют рекомендации относительно использования моделей и принятия решений.

Вышеприведенные этапы моделирования взаимосвязаны, а сама процедура создания итерационна. Это объясняется тем, что после выполнения каждого этапа проверяются правильность и достоверность модели, и в случае несоответствия модели объекту производится возвращение к предыдущим этапам з целью корректировки и настройки модели. В зависимости от характера внесенных изменений возвращаются непосредственно к предыдущему этапу или к более ранним этапам.

На последнем этапе моделирования документально формулируются результаты исследования и готовится программная документация для использования результатов во время разработки текущих и будущих проектов.

Рассмотрим, какие этапы жизненного цикла программ и моделей схожи между собой (рис. 3).

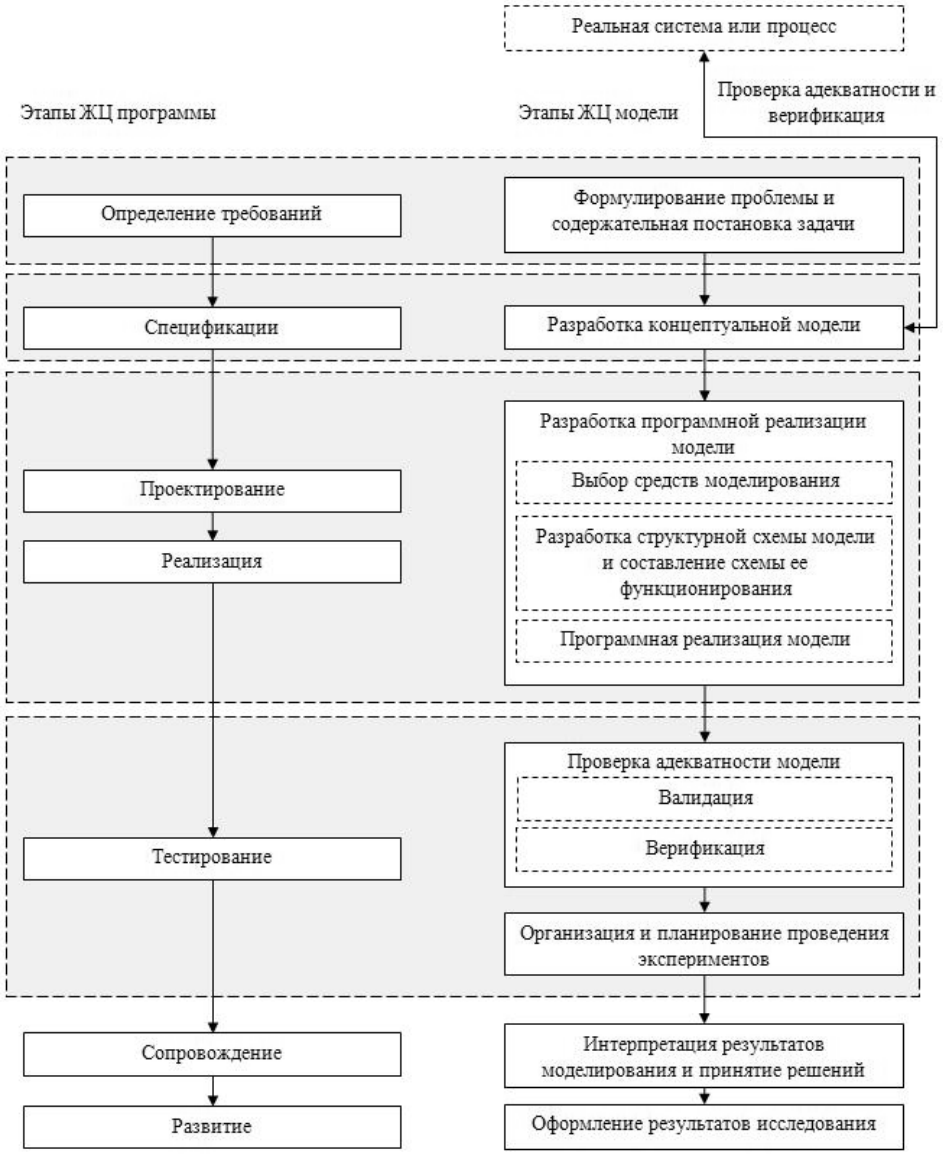

Рис. 3. Сравнение этапов жизненного цикла программы и модели

Если для моделирования сложной системы создается имитационная модель, то некоторые этапы жизненного цикла моделей очень схожи с этапами жизненного цикла программ.

Например, если говорить об этапе формулирования проблемы и содержательной постановки задачи, то одной из важных задач, которую необходимо решить на этом этапе при создании имитационной модели системы является определение требования к модели и к моделируемой системе. Таким образом, при создании модели системы можно выделить два вида требований: требования к моделируемой системе и требования к модели. Требования к системе можно получить из готовой системы или из разрабатываемой системы. Например, если мы создаем модель программной системы, то, скорее всего, требования к ней уже были выдвинуты на этапе анализа требований к системе. После того, как требования к системе и к модели системы были выдвинуты, их необходимо проверить, чтобы они были непротиворечивы. И для этого можно использовать те же инструменты, которые используются в жизненном цикле программы на этапе анализа требований к системе. Также на этапе формулирования проблемы решаются вопросы, связанные с определением целей создания программы либо модели, определением временных рамок разработки и ресурсов, которые можно затратить на разработку. Производится активное взаимодействие с заказчиком для более четкого формирования требований.

Этап спецификации жизненного цикла разработки программы соответствует этапу разработки концептуальной модели в жизненном цикле создания модели. Спецификацию программы можно считать ее концептуальной моделью, особенно если спецификация представлена в виде сущности, описанной на некотором специализированном языке спецификаций. Задачей этапа спецификации являются сбор всех требований и устранение противоречивости этих требований. Этап разработки концептуальной модели также в некоторой мере предназначен для устранения противоречивостей, таких как создание максимально точной модели при как можно большем ее упрощении по сравнению с реальным объектом, что позволит сэкономить ресурсы, затрачиваемые при создании модели, а также ресурсы, затрачиваемые на прогоны модели.

Наиболее схожими являются этапы проектирования и реализации в ЖЦ создания программ и этап разработки программной реализации модели. Схожесть проявляется особенно четко в том случае, если для создания был выбран прикладной язык программирования. Этап разработки программной реализации в модели тесно связан с разработкой концептуальной модели. Хотя разработка имитационной модели в значительной степени похожа на разработку обычной программы, но разработка модели происходит на основании зафиксированной архитектуры, которая выбирается на основании разработки концептуальной модели.

Этап разработки концептуальной модели при создании имитационной модели системы состоит из трех шагов:

1. Выбор степени детализации описания объекта моделирования, на котором определяется, насколько детально должен быть описан объект моделирования. На этом шаге необходимо выбрать между стоимостью модели и погрешностями моделирования. Чем выше будет степень детализации объекта моделирования, тем дольше будет происходить процесс ее разработки и калибровки и тем меньше будут отклонения от реальной системы полученных результатов.

2. Описание переменных модели, на котором определяются входные, выходные, внутренние параметры и их распределения.

3. Формализованное изображение концептуальной модели, которое является самым важным шагом на этапе разработки концептуальной модели, и которое тесно связывает этот этап с этапом разработки программной реализации модели.

Необходимо рассмотреть, каким образом формализованное изображение концептуальной модели влияет на этап программной реализации модели, а также определить, что такое программа с фиксированной архитектурой. Фиксированная архитектура имитационной программы появляется в тот момент, когда происходит формальное изображение концептуальной модели. После того, как был выбран формализм для описания имитационной модели, у этого формализма, как правило, есть набор подходов, которые применяют, чтобы на их основании создать программную реализацию модели.

Этап тестирования ЖЦ программы соответствует этапам проверки адекватности модели, организации и планирования проведения экспериментов. Также можно говорить о том, что экспериментирование над программной реализацией модели следует проводить с использованием таких же инструментальных средств, как и при ее тестировании. Можно говорить о том, что процесс экспериментирования над моделью схож с процессом ее тестирования, но без сравнения результатов эксперимента с заведомо известным результатом. Вместо того, чтобы сравнивать результат прогона модели с заведомо известным результатом, производятся их постобработка и вывод окончательного результата на экран.

3. Инструментальные средства валидации, верификации и экспериментирования

Проверка адекватности модели выполняется с использованием формальных статистических критериев. Однако такая проверка возможна при наличии надежных статистических параметров как оригинала (объекта прогнозирования), так и модели. Если по каким-то причинам такие оценки отсутствуют, то осуществляют сравнение отдельных свойств оригинала и модели. При этом первоначально должна проверяться истинность реализуемых функций, затем истинность структуры и, наконец, истинность достигаемых при этом значений параметров. Для этого, помимо модели, необходимо иметь функционирующий оригинал, то есть проводить сопровождающее моделирование.

Верификация модели – оценка ее функциональной полноты, точности и достоверности с использованием всей доступной информации в тех случаях, когда проверка адекватности по тем или иным причинам невозможна.

Чаще всего используется верификация моделей, так как в большинстве случаев реальный объект отсутствует или разрабатываются новые (еще не существующие) функции объекта моделирования.

Степень совершенства модели выражают через различные измерители точности. Точность точечного результата эксперимента в момент f определяется разностью между значением точечного эксперимента P и фактическим значением Fh моделируемого показателя в определенный момент времени. Отдельный точечный эксперимент не определяет точность процедуры моделирования в целом, то есть потребуется некоторая выборка {(Pj, FJ)}, на основе которой рассчитывается значение некоторого измерителя точности моделирования.

В настоящее время нет достаточно полного исследования всевозможных критериев точности, что затрудняет оценивание адекватности различных моделей и опыта их применения в моделировании систем и процессов.

Для проверки адекватности моделей можно использовать любой коэффициент парной корреляции между последовательностями смоделированных и фактических значений. Классический критерий проверки адекватности модели – коэффициент корреляции Пирсона.

Проверка модели на адекватность является одним из критериев завершения итерации жизненного цикла модели наряду с модульным и интеграционным тестированием программной реализации модели. Проверка модели на адекватность выполняется в том случае, если было проведено экспериментирование с реальным объектом либо процессом. В случае, если данные наблюдений над реальным объектом либо процессом отсутствуют,

необходимо проводить верификацию модели. Проверка модели на адекватность (верификация в случае отсутствия данных наблюдения) похожа по своей природе на тестирование. Задачей тестирования является сравнение выходных данных с эталоном, задачей проверки модели на адекватность является проверка адекватности отклика при заданных входных значениях фактора. В случае верификации модели, когда отсутствуют данные наблюдения за реальным объектом либо процессом, сравнение данных не происходит, а производится исследование набора откликов модели с использованием набора специальных методов.

Таким образом, отличием процесса тестирования модели от процесса проверки ее на адекватность является то, что при проверке модели на адекватность производится проверка наборов входных и выходных данных, а не значений единичного эксперимента. При тестировании чаще всего происходит сравнение выходного результата функции с эталоном. Таким образом, для того чтобы обеспечить автоматизацию процесса проверки модели на адекватность (верификацию), необходимо инструментально расширить средство тестирования дополнительным функционалом для подачи на вход теста наборов данных, например с базы данных, а также функционалом для проведения корреляционного анализа, на основании которого будет применяться решение об адекватности модели.

Рассмотрим пример модульного теста, написанного на языке Java с использованием фреймворка TestNG.

```
//Класс для тестирования
public class Example {

  public Integer add(Integer a, Integer b) {
    return a + b;
  }
}

//Модульный тест для класса
public class ExampleTest {

  @Test
  public void testAdd() throws Exception {
    // given
    Example example = new Example();
    // when
    int result = example.add(10, 10);
    // then
    Assert.assertEquals(20, result);
  }
}
```

Тестовый метод помечен специальным атрибутом @Test, который указывает, что данный метод необходимо запускать под управлением инструментальных средств TestNG. С использованием данного подхода можно создавать как модельные, так и интеграционные тесты в зависимости от того, была ли тестируемая логика изолированной. TestNG предоставляет возможность создавать отчеты с подробной информацией о результатах тестирования.

Для проверки модели на адекватность введем дополнительный атрибут с дополнительными параметрами для задания @Correlation(etalonData=etalonDataSource, resultData=resultDataSource), где etalonData и resultData – параметры для задания входного

и выходного источников данных соответственно. Пример блока кода, предназначенного для проведения автоматизированной проверки модели на адекватность, представлен ниже:

```
//Пример автоматизированной проверки на адекватность
public class ExampleCorrelation {

  @Correlation(etalonData=etalonDataSource, resultDa-
ta=resultDataSource)
  public void correlationAnalysis() throws Exception {
    // given
    Model model = new Model();
    // when
    model.run(100);
    resultData = model.getResponse();
    // then
    Assert.assertCorrelation(myResultDataSource, myEtalonData-
Source);
  }
}
```

Аналогично с автоматизированной проверкой модели на адекватность предложено добавить атрибут @Experiment(resultData=resultDataSource) для запуска автоматизированных экспериментов. resultData – является источником данных, в котором будут сохраняться результаты экспериментов. doRegressionAnalysis – метод, производящий обработку результатов прогона модели и формирующий отчет TestNG.

```
//Пример автоматизированного эксперимента
public class ExampleExperiment {

  @Experiment(resultData=resultDataSource)
  public void myExperiment() throws Exception {
    // given
    Model model = new Model();
    // when
    model.run(100);
    resultData = model.getResponse();
    // then
    doRegressionAnalysis(myResultDataSource);
  }
}
```

Методы, обозначенные атрибутами @Correlation и @Experiment, можно запускать под управлением расширенных инструментальных средств автоматизированного тестирования TestNG, а также запускать под управлением сервера непрерывной интеграции, что даст возможность всем членам команды, принимающим участие в разработке модели, иметь доступ к результатам автоматизированной проверки модели на адекватность и результатам автоматизированных экспериментов. Результатом запуска методов, как и при запуске методов, обозначенных атрибутом @Test, являются отчеты, которые, в отличие от отчетов, сформированных в результате выполнения тестового метода, содержат графики.

4. Модели жизненного цикла, пригодные для создания моделей, и инструментальные средства их поддержки

Последовательные модели жизненного цикла (те, в которых движение происходит от одного этапа к другому без возвратов) являются идеализированными, так как только очень простые задачи проходят все этапы без каких-либо итераций. При программировании, например, может обнаружиться, что реализация некоторой функции очень громоздка, неэффективна и вступает в противоречие с требуемой от системы производительностью. В этом случае требуется перепроектирование, а, может быть, и изменение спецификаций. При разработке больших нетрадиционных систем необходимость в итерациях возникает регулярно на любом этапе жизненного цикла как из-за допущенных на предыдущих шагах ошибок и неточностей, так и из-за изменений внешних требований к условиям эксплуатации системы. Наиболее развитым в плане понятия итерации является объектно-ориентированный подход. Для традиционных подходов итерация – это исправление ошибок, то есть процесс, который с трудом поддается технологическим нормам и регламентам. При объектно-ориентированном подходе к итерации никогда не отменяют результаты друг друга, а всегда только дополняют и развивают их.

При создании моделей, так же как и при разработке программного обеспечения, применяется командный подход: когда над проектом работают несколько разработчиков одновременно. Командный подход требует набора специальных инструментальных средств, которые обеспечивают эффективную работу над проектом. Важными инструментами при командной работе над проектом являются система контроля версий, сервер непрерывной интеграции и фреймворки модульного и функционального тестирования. Связь инструментальных средств командной разработки программ представлена на рис. 4.

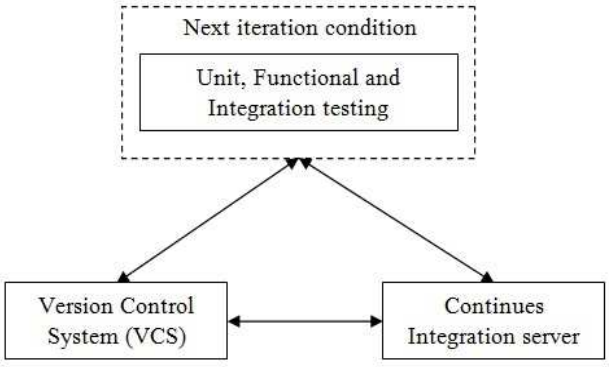

Рис. 4. Связь инструментальных средств командной разработки программ

Данные инструментальные средства помогают сохранять текущее состояние файлов, связанных с проектом (систем контроля версий VCS), а также производить прогон тестов (сервер непрерывной интеграции CI). Модульные и функциональные тесты используются для того, чтобы определить, не наступил ли регресс в программном коде, а также для определения возможности завершения текущей итерации.

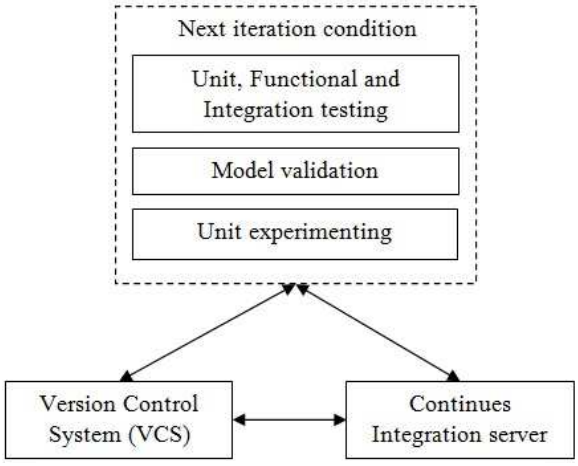

Рис. 5. Связь инструментальных средств командной разработки моделей

Учитывая то, что для определения завершения итерации при создании модели необходимо не только произвести тестирование программной реализации модели, а также произвести ее верификацию, необходимо расширить набор инструментальных средств командной разработки инструментальными средствами автоматической верификации моделей. При этом связь инструментальных средств командной разработки моделей будет иметь вид,

представленный на рис. 5. Дополнительные инструментальные средства для автоматической верификации позволят производить верификацию и тестирование моделей в автоматическом режиме в процессе создания модели, а также по завершении итерации модели жизненного цикла.

5. Выводы

Первое упоминание об объектно-ориентированной парадигме появилось в языке Simula, предназначенном для моделирования. Таким образом, моделирование дало начало объектно-ориентированному подходу. Объектно-ориентированные языки хорошо подходят для создания моделей систем и процессов, а применение объектно-ориентированной парадигмы в моделировании привело к созданию агентного подхода в моделировании.

Сравнение жизненных циклов программ и моделей показало, что этапы жизненного цикла программы подобны этапам жизненного цикла модели. Для создания моделей наиболее походящим является итерационная модель жизненного цикла, которая также используется для создания программ. Особенностью применения итерационной модели ЖЦ для создания моделей является применение процессов проверки модели на адекватность и валидации модели в качестве условий остановки итерации.

Поскольку этапы жизненного цикла создания моделей схожи с этапами жизненного цикла создания программ, то инструментальные средства, которые используются на этапах жизненного цикла программ, можно использовать на схожих этапах жизненного цикла моделей.

Результат процесса проверки модели на адекватность является одним из критериев завершения итерации жизненного цикла модели. По аналогии к атрибуту @Test инструментального средства TestNG были предложены атрибуты @Correlation и @Experiment, которые предназначены для проведения автоматизированного корреляционного анализа и автоматизированного тестирования.

СПИСОК ЛИТЕРАТУРЫ

1. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: учебник / Вендров А.М. – [2-е изд., перераб. и доп.]. – М.: Финансы и статистика, 2006. – 544 с.
2. Томашевський В.М. Моделювання систем / Томашевський В.М. – К.: Видавнича група BHV, 2007. – 352 с.

Стаття надішла до редакції 14.10.2014