А. А. Вичугова, канд. техн. наук, Томский политехнический университет, vichugovaaa@tpu.ru

Автоматизация процесса разработки программного обеспечения: методы и средства

В статье рассмотрены тенденции сокращения рутинных операций при проектировании, кодировании, тестировании, развертывании и документировании программного обеспечения (ПО), проанализировано воплощение этих трендов в виде ряда инструментов, автоматизирующих один или несколько аспектов разработки ПО. Приведены примеры трансформации классических понятий разработки ПО к современному состоянию с учетом текущих потребностей и возможностей рынка информационных технологий (ИТ). Описаны возможные пути качественного изменения пользователя и профессионала в ИТ-сфере.

Ключевые слова: среды разработки программного обеспечения, фреймворки, парадигмы и языки программирования.

Введение

Функциональные этапы процесса разработки программного обеспечения (ПО) реализуются в виде различных моделей и методологий. Принято выделять следующий типовой набор шагов осуществления данной деятельности [1]:

1) анализ предметной области и формализация бизнес-процессов;
2) анализ и разработка требований к ПО;
3) разработка архитектуры ПО и планирование ее реализации;
4) кодирование;
5) тестирование и отладка;
6) разработка программной документации (ПД);
7) сертификация продукта (ПО и ПД);
8) внедрение;
9) сопровождение.

Последовательное выполнение вышеуказанных фаз оформилось в водопадную или каскадную модель разработки ПО, впервые описанную У. У. Ройсом в 1970 г. [2]. Схематичное представление данной модели показано на рис. 1.

С учетом критики каскадной модели относительно отсутствия гибкости и длительности разработки вследствие невозможности начала нового этапа без завершения предыдущего тем же У. У. Ройсом была предложена модификация процесса [2]. Данная модель получила название итеративной разработки вследствие распараллеливания шагов с непрерывным анализом полученных результатов и корректировкой предыдущих этапов [3]. На рис. 2 показана схема итеративной модели.

Дальнейшим развитием итеративной модели можно считать спиральную модель разработки ПО, предложенную Б. Боэмом в 1986 г. В рамках этой модели на протяжении всего периода разработки ПО непрерывно выполняется анализ рисков, проектирование и постадийное прототипирование с учетом исходных и вновь возникающих требований к продукту. Аналогичный принцип непрерывного уточнения и детализации проекта лежит в основе V-образной модели, предложенной в 90-х годах XX в. [4].

Практическая реализация спиральной и V-образной моделей, включая непосред-

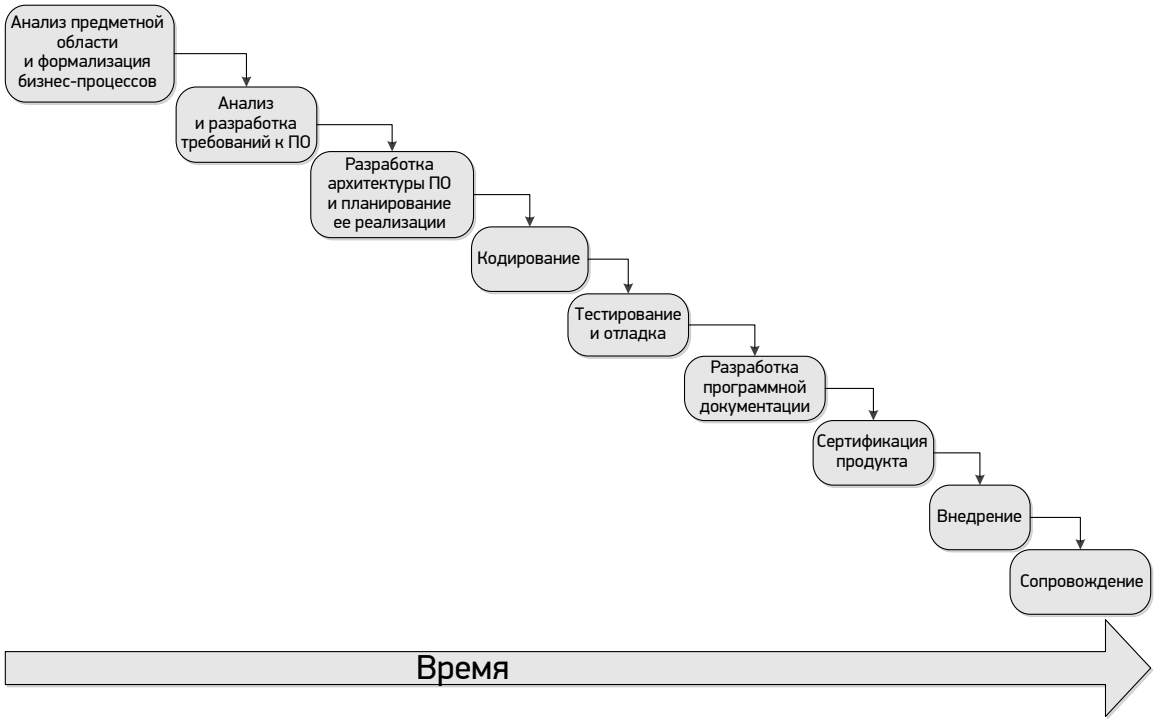

Рис. 1. Водопадная модель разработки ПО

Fig. 1. Waterfall model of software development process

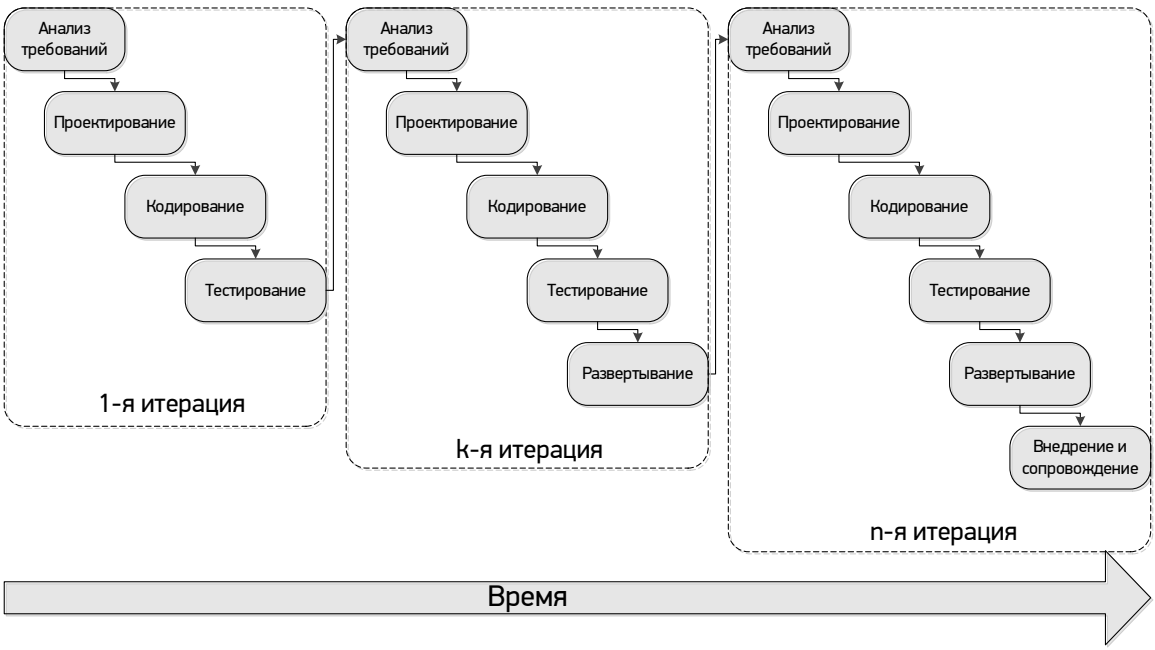

Рис. 2. Итерационная модель разработки ПО

Fig. 2. Iterative model of software development process

ственную организацию деятельности, воплощена в виде методологий гибкой разработки ПО: разработка через тестирование, экстремальное программирование, SCRUM, Lean, DSDM и многих других подходов категории Agile [5]. Например, на рис. 3 показан процесс разработки ПО в рамках гибкой методологии TDD — *test-driven development*, разработка через тестирование, в соответствии с V-образной моделью процесса.

Популяризация гибких методологий разработки ПО обусловлена требованиями рынка к сокращению времени выпуска продукции и повышению ее качества. Поэтому в настоящее время на практике используются модели постадийной разработки ПО с непрерывным прототипированием. Это влечет за собой стирание границ в ролевом разделении профессионалов, решающих конкретные задачи разработки и анализа требований, кодирования, тестирования и т. д. Однако помимо методологических и организационных аспектов реализации процесса программирования, как и в любой прикладной деятельности, важным фактором ее эффективного выполнения является использование удобных инструментов. В разработке ПО такие средства — языки программирования и среды разработки, а также сопутствующие методы, приемы и инструменты автоматизации данного процесса. Однако развитие подобного инструментария обусловлено не только текущими потребностями бизнеса, но и перспективами качественных изменений в ИТ-отрасли. Таким образом, наряду с аналитическим обзором современного уровня автоматизации процессов разработки ПО, в данной работе ставится задача прогнозирования возможных трендов развития ИТ-сферы с учетом выявленных тенденций.

Автоматизация процесса разработки

В настоящее время в прикладной области разработки ПО существует ярко выраженная тенденция повышения уровня абстракции программирования и проектиро-

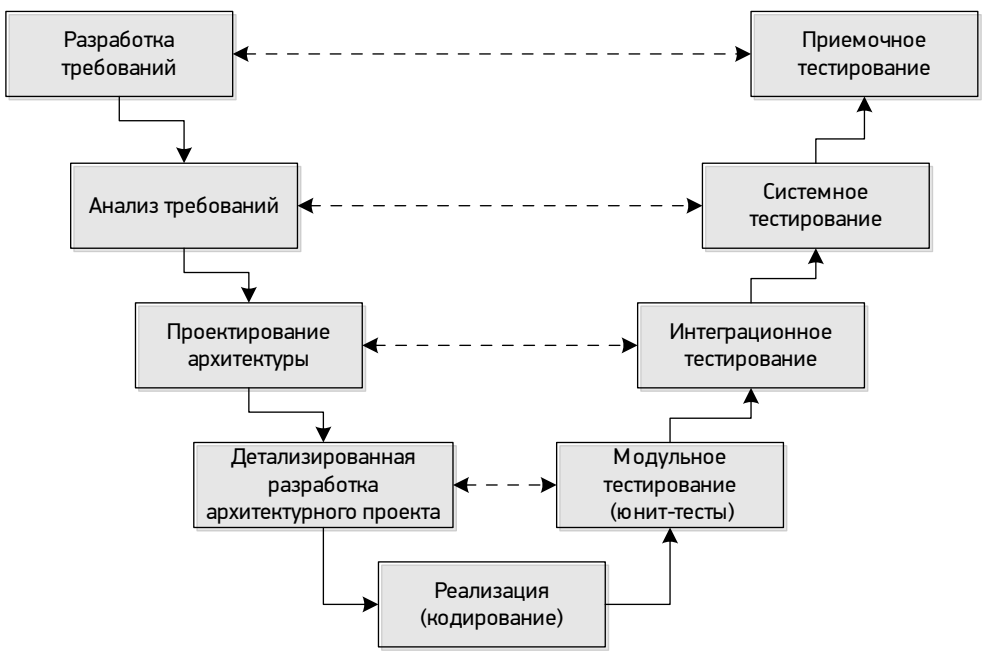

Рис. 3. Разработка через тестирование в рамках V-образной модели
Fig. 3. Test-driven development under V-model of the process

вания систем. Отсутствуют четкие границы между типовыми этапами процесса разработки, например, архитектурные конструкции в виде шаблонов уже включают в себя некоторые аспекты реализации: генерацию программного кода, моделей и интерфейсов представления данных, формирование проверочных тестов и программной документации. На методическом уровне классические объектно-ориентированные паттерны проектирования ПО («фасад», «наблюдатель», «компоновщик» и т. д.) воплощены в виде схем реализации архитектуры приложений: Model-View-Controller, Model-View-Presenter, Model-View-ViewModel. Каждый из указанных паттернов представляет собой систему из нескольких взаимодействующих компонентов, разделенных по функциональному признаку, например отделение бизнес-логики от данных и интерфейсных представлений [6]. Таким образом, реализуются основные SOLID-принципы объектно-ориентированного проектирования ПО, сформулированные Р. Мартином [7]:

- single responsibility (единственная ответственность класса/компонента);
- open-closed (программные сущности открыты для расширения, но закрыты для изменения);
- Liskov Substitution (принцип подстановки Барбары Лисков: функции, использующие ссылки на базовые классы, могут использовать объекты производных классов);
- interface segregation (разделение интерфейсов) и dependency inversion (декомпозиционная зависимость деталей от абстракций).

Перечисленные принципы также соответствуют положению о повторном использовании независимых модулей в рамках компонентно-ориентированного подхода [8].

Таким образом, лучшие практики проектирования и программирования унифицируются и в дальнейшем используются при разработке новых продуктов в качестве так называемых «кирпичиков», как стандартизованная единица материала при постройке нового здания.

На прикладном уровне такие архитектурные решения поддержаны конкретными инструментами разработки ПО. При этом последние представлены в виде модульных программных каркасов — фреймворков, объединяющих множество отдельных компонентов для решения прикладных задач. Расширение типовых функциональных возможностей интегрированной среды разработки возможно за счет подключения дополнительных библиотек и пакетов. Наиболее наглядными примерами подобных инструментальных средств сегодня являются фреймворки для разработки веб-приложений: Ruby on Rails, Pylons, Phalcon, Django, Zend Framework, Symfony, Yii и т. д. [9].

Использование таких программных каркасов позволяет ускорить процесс разработки за счет автоматизированного формирования базы данных на основе моделей, пользовательского интерфейса и функциональных компонентов. Дополнительным преимуществом является наличие у современных языков программирования и сред разработки инструментов для тестирования ПО. Например, модули doctext и unittest в стандартной библиотеке языка Python, TextTest — кроссплатформенный фреймворк для тестирования графического интерфейса пользователя, Selenium WebDriver — для автоматического тестирования веб-интефейсов и другие подобные системы.

Также современные средства автоматизации процесса разработки ПО позволяют сократить длительность такого достаточно трудоемкого процесса, как разработка программной документации. Это обеспечивается с помощью генераторов документации — программ, анализирующих языковые конструкции и комментарии в исходном коде.

В результате на выходе формируется программная документация в унифицированном формате (HTML, HTMLHelper, PDF, RTF и т. д.), предназначенная для технических специалистов или конечных пользо-

вателей продукта. Вследствие синтаксических особенностей различных языков программирования и прикладных отличий сред разработки друг от друга сегодня фактически для каждого популярного фреймворка существует несколько совместимых с ним генераторов документации. Например, кроссплатформенная система документирования исходных текстов Doxygen поддерживает языки C++, C, Objective-C, Python, Java, IDL и PHP; генератор документации и поисковой движок Sphinx для языка Python интегрирован с веб-фреймворками Django и Pylons; VSdocman — коммерческий генератор документации для исходных текстов на языках VisualBasic.NET и C#, реализованный в виде дополнения к среде разработки Microsoft Visual Studio. Также следует упомянуть инструментарий дескрипторов Javadoc, ставший стандартом де-факто для документирования классов, интерфейсов, полей, конструкторов, методов и пакетов в программах, написанных на Java-подобных языках.

Возвращаясь к использованной ранее метафоре сравнения паттернов проектирования со строительными блоками, можно продолжить аналогию и представить процесс разработки ПО как сборку здания из готовых железобетонных панелей в технологии панельного домостроения. Разработка по принципу конструктора Lego®, при этом практически все этапы создания продукта могут быть реализованы в рамках одного инструментального средства — с помощью фреймворка.

Таким образом, типовая последовательность шагов создания предметно-ориентированной информационной системы с базой данных и веб-интерфейсом, показанная на рис. 4, фактически принимает вид, представленный на рис. 5.

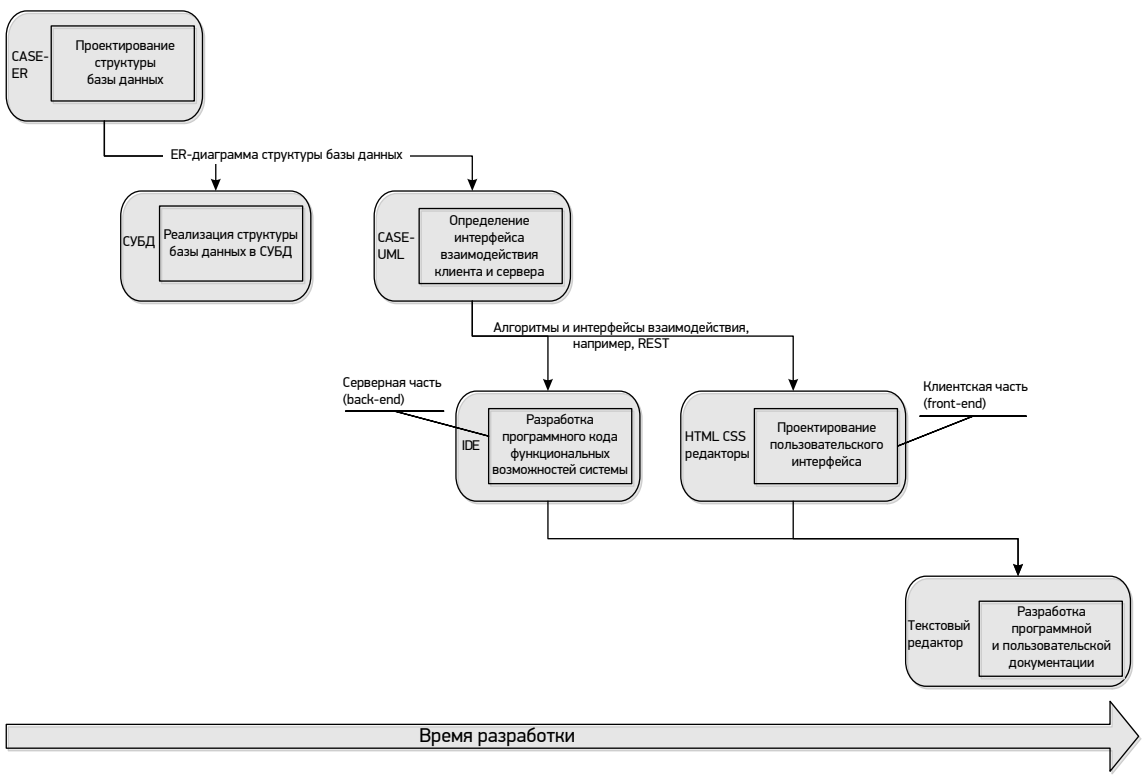

Рис. 4. Классические этапы и средства разработки веб-приложения
Fig. 4. Classic stages and tools of development Web-application

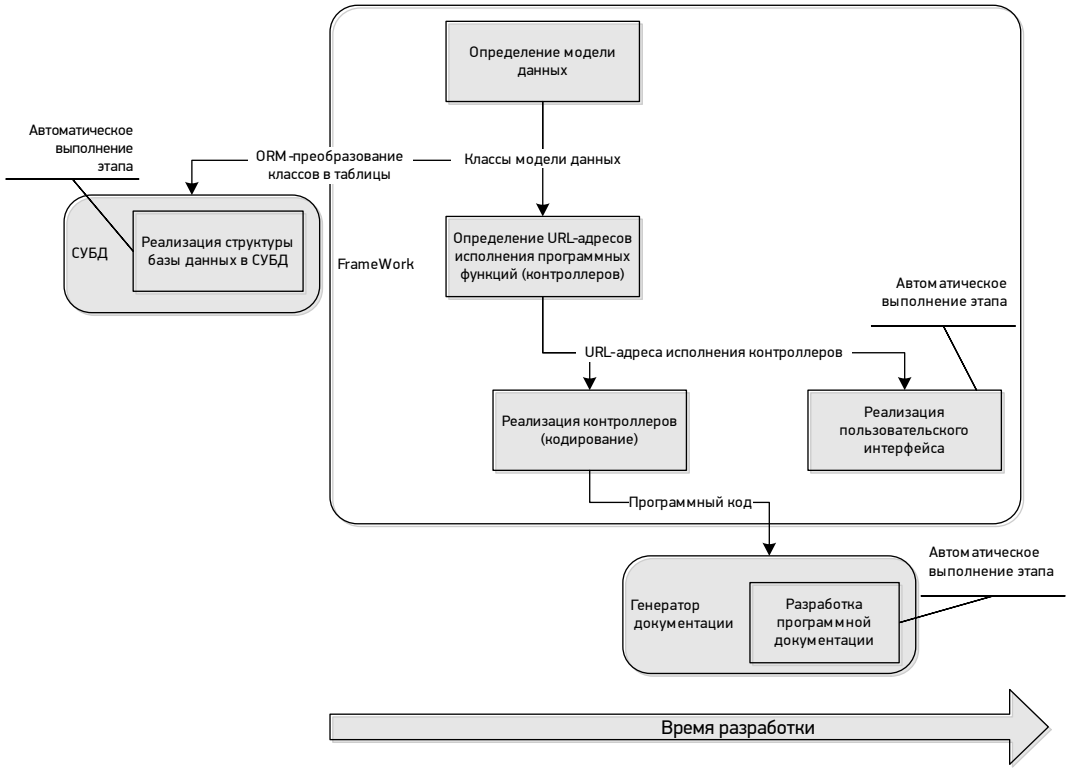

Рис. 5. Модернизация процесса разработки веб-приложений в рамках фреймворка
Fig. 5. The modernization of development web application process within the framework

Сокращения времени разработки достигают за счет автоматизации некоторых этапов, единого (сквозного) инструмента разработки и отсутствия необходимости итеративного выполнения каждого процесса при внесении изменений. Последнее обеспечивается благодаря наличию центрального источника данных (модели) и автоматическому исправлению необходимых связанных с ней элементов. Эта концепция сквозной интеграции данных, создаваемых и используемых на различных этапах процесса разработки ПО, также лежит в основе веб-систем управления знаниями и ИТ-проектами. Как правило, подобные базы знаний основываются на вики-движках и представляют собой веб-сайт, структуру и содержимое которого пользователи могут самостоятельно изменять с помощью встроенных инструментов. Среди наиболее популярных систем этой категории следует упомянуть DokuWiki, XWiki, ThoughtFarmer, TikiWiki, eTouchSamePage, Mediawiki, Kerio Samepage, Socialtext, PBWorks, Foswiki, Rizzoma, Zoho Wiki, Confluence [10].

Для автоматизации процессов разработки ПО базы знаний интегрированы с системами отслеживания ошибок, контроля версий и управления ИТ-проектами. Например, открытое веб-приложение Redmine, Kerio Samepage или XWiki, сочетающие оба этих функциональных направления, связка продуктов компании Atlassian: Confluence и JIRA и другие подобные сопряжения, включающие ссылки на участки кода в рамках веб-хостинга проектов типа Bitbucket и GitHub [11]. Таким образом, автоматизируются не только этапы разработки ПО, но и деятельность по управлению этими процессами.

Дальнейшие этапы жизненного цикла ПО: развертывание, внедрение и сопровождение — также автоматизируются. В связи с активным распространением облачных

технологий, включая все возможные модели их использования (SaaS — ПО как услуга — Softwareas a Service, PaaS — платформа как услуга — Platformas a Service, IaaS — инфраструктура как услуга — Infrastructureas a Service, DaaS — виртуальное рабочее место — Desktopas a Service), этап развертывания сводится к выбору наиболее подходящей модели облака (частное, публичное, общественное или гибридное) и объекта оказания услуги (ПО, платформа, инфраструктура, рабочее место) [1]. В такой ситуации капитальные вложения в ИТ-комплекс существенно снижаются при одновременном увеличении операционных издержек на оплату услуг облачных провайдеров.

Если рассматривать развертывание вновь созданного ПО как этап, логически следующий за разработкой и тестированием, то наибольший интерес представляет PaaS-модель предоставления облачных технологий. В этом случае облачный провайдер предоставляет программную платформу, оснащенную компонентами, необходимыми для функционирования разворачиваемого ПО (операционная система, СУБД, среды исполнения языков программирования и пр.), и готовую для размещения новых приложений. Для этого используются технологии виртуализации приложений и оркестрации сервисов. С 2010 г. по настоящее время активно развиваются средства контейнеризации и концепция микросервисов, когда одно приложение состоит из множества микросервисов и разворачивается в виде множества отдельных, но связанных между собой контейнеров.

В свою очередь, контейнер представляет собой программно-логический пакет, в который «упаковано» приложение со всем его окружением и зависимостями. Это позволяет переносить контейнеры между физическими машинами с целью сокращения нагрузки на узлы распределенной сети. Популярными в настоящее время средствами контейнеризации являются:

- Maas/juju — стек Ubuntu, позволяет управлять множеством географически распределенных физических серверов и запускать приложения из готовых шаблонов;
- проприетарные облака: Red Hat (OpenShift), VMWare (Cloud Foundry), Google (App Engine);
- решения для совместного хостинга (CloudLinux с изолированными окружениями);
- ПО для виртуализации на уровне операционной системы (Docker от компании Docker, Rocket от фирмы CoreOS).

Перечисленные продукты и подобные им сервисы постоянно развиваются, например, в Rocket версии 0.2.0 добавлен функционал зашифрованной подписи по умолчанию, средства проверки состояния контейнера и заключенного в него приложения. При этом функция автоматической подписи поддерживается системой хранения открытых ключей [12].

Следующий за развертыванием этап внедрения ПО в деятельность предприятия, как правило, — в большей степени организационно-методический проект, включающий обучение пользователей, а иногда и изменение порядка выполнения их привычной работы. Тем не менее данный процесс частично автоматизирован за счет типовых процедур и методик внедрения, специфичных для конкретного продукта. Например, компании SAP, 1С, Директум и другие фирмы — производители ПО предлагают готовые шаблоны и лучшие практики внедрения своих продуктов [13]. Из стандартизованных решений следует упомянуть методологию «Rapid Concept Development & Implementation» (RCDI), которая совместима с идеологиями управления проектами и качеством (PMI PMBOK и TQM), стандартами серии ISO 9001:2000 (ГОСТ Р ИСО 9001:2001), SA 8000:1997 [14].

Аналогичным образом организован процесс сопровождения ПО, частично автоматизированный с помощью методических и программных средств. Методологическим бази-

сом процесса сопровождения считается ITIL (от англ. *IT Infrastructure Library* — библиотека инфраструктуры ИТ) — структурированное описание лучших практик и способов организации работы по оказанию услуг в области информационных технологий. В соответствии с процессным подходом ITIL представляет процессы сопровождения ПО как совокупность взаимосвязанных сервисов управления (проблемами, инцидентами, конфигурациями, релизами, изменениями и т. д.) и поддержки пользователей. Техническая поддержка пользователей согласно ITIL организована по принципу централизации сбора обращений конечных пользователей и их распределения по уровням в зависимости от глубины проблемы [15]. Техническими средствами реализации этого процесса являются системы и сервисы обработки обращений (инцидентов), например Итилиум, ITSM 365, Okdesk, HelpDeskEddy и прочие аналоги [16]. При этом данная категория инструментальных средств также активно мигрирует от локально-производственных систем в сторону SaaS-решений.

Развитие языков программирования

В рамках этапа непосредственной разработки ПО — кодирования целесообразно рассмотреть вопрос выбора языка программирования (ЯП), позволяющего использовать средства автоматизации этого процесса (шаблоны, фреймворки и т. д.). Традиционная классификация ЯП основана на их принадлежности к базовой модели вычислений и парадигме. Однако на практике важным критерием выбора парадигмы программирования и языка как средства ее реализации является набор решаемых задач, обусловленный областью применения создаваемого ПО. Например, для создания распределенных приложений подходят языки, включающие средства порождения параллельных облегченных процессов и их взаимодействия через обмен асинхронными сообщениями в соответствии с моделью акторов [17]. Наиболее популярными в настоящее время примерами таких языков считаются Erlang и Go.

Другим важным критерием выбора ЯП как основного средства разработки ПО является область, в которой можно пожертвовать скоростью взамен получения специфических преимуществ, которые дает выбранный язык. В соответствии с существующими способами реализации ЯП принято укрупненно делить на компилируемые (программа преобразуется в машинный код и далее собирается в исполняемый модуль) и интерпретируемые (обработка программы происходит во время ее исполнения). Такая классификация предполагает два типа «медленных» элементов в процессе разработки:

- «медленный компилятор» (примеры ЯП: C++, Rust, C#);
- «медленная среда исполнения» (примеры ЯП: Java, JavaScript, .Net, Clojure, Groovy и другие интерпретируемые языки для платформ JVM и CLR).

Однако можно также выделить третий, не менее важный аспект потери производительности в процессе разработки ПО — «медленный программист». Если ЯП имеет невысокий уровень абстракции и считается достаточно сложным, то даже при высокой скорости компилирования процесс разработки кода будет достаточно медленным. Классическим примером такого ЯП считается язык C, созданный с целью написания портируемых программ/библиотек, быстродействующий и сопоставимый по скорости и сложности разработки с ассемблером. Таким образом, обратной стороной достоинства этого языка — быстродействия исполнения программ — является длительное время создания, чтения и поддержки кода. Аналогичное следствие возникает в случае использования молодых ЯП, специфические особенности которых обусловливают их уровень семантической сложности при создании/чтении и поддержке кода.

На основании проведенного анализа тенденций появления новых ЯП и развития существующих можно сделать вывод о движении в сторону мультипарадигмальности, в частности, сочетание императивного и декларативного подходов в виде возможностей объектно-ориентированного, процедурного, структурного, а также функционального и логического программирования. Примерами таких достаточно молодых, но уже завоевавших популярность у разработчиков, являются языки Scala (2003 г.), F# (2005 г.), Go (2009 г.), Rust (2010 г.), Swift (2014 г.) и т. д.

Еще одна ярко выраженная тенденция развития рынка ЯП последних лет — популяризация декларативной парадигмы: все больше языков поддерживают функциональный и логический подходы. В частности, языки запросов и разметки (XSLT, SQL, CSS, HTML и т. д.) — наглядная демонстрация подобного декларативного подхода. Причина такой популяризации декларативных языков — повышение уровня сложности программ и ранее описанное понятие программирования «по принципу конструктора Lego©». Именно декларативные ЯП для решения задачи позволяют не задавать пошаговый алгоритм — достаточно лишь описать желаемый результат в виде системы правил (ограничений). Кроме того, семантика декларативных ЯП предполагает независимость смысла используемого оператора от его положения в структуре программы. Это обстоятельство делает декларативные программы проще императивных, что также соответствует вышеозначенной тенденции автоматизации процесса разработки ПО.

Вышеозначенные тенденции представлены не только появлением новых ЯП, но и развитием существующих, что подтверждают новые выпуски языков с долгой историей успешного использования. Например, зародившиеся в 80–90-х годах XX в. и популярные до настоящего времени C++, Java, Python, Ruby, JavaScript, диалекты и расширения Haskell, Mercury и т. д. Также следует отметить повышение интереса к ЯП, ориентированным на сложные математические вычисления и обработку статистических данных (R, Julia, MATLAB, J, S, SaS и т. д.). Такие ЯП активно используются в качестве базовых средств для информационно-аналитических систем с большим объемом данных (Big Data) в различных прикладных областях: финансы, медицина, торговля, задачи обработки графики и т. д.

В свою очередь, популяризация отдельных тенденций ЯП поддерживается соответствующими методами и средствами обеспечения процесса разработки ПО, о которых было сказано выше: методологии, модели, лучшие практики, шаблоны и фреймворки.

Сервисная модель процесса разработки ПО

На основании рассмотренных тенденций развития технических и организационных изменений в области разработки ПО можно предположить дальнейшее усиление тренда автоматизации всех процессов данной области деятельности уже в ближайшее время. С учетом проникновения ИТ во все сферы человеческой жизни тезис о навыках программирования как необходимом качестве современного человека, впервые озвученный в начале 80-х годов. XX в. А. П. Ершовым на Третьей всемирной конференции ИФИП и ЮНЕСКО по применению ЭВМ в обучении [18], становится все более актуальным. Многие бытовые явления, термины и подходы ведения бизнеса, например виртуальное общение в рамках социальных сетей, сервисные модели предоставления услуг и ведения проектов, пришли в реальную жизнь и бизнес именно из сферы ИТ. Однако любительский и профессиональный уровни владения данным инженерным ремеслом или искусством [19] отличаются друг от друга прежде всего наличием системных знаний о теоретических основах предмета. Тем не менее масштабное оснащение ИТ-инструментами всех отраслей

человеческой деятельности (интернет-магазины, учетно-управляющие информационные системы и пр.) приводит к популяризации навыков ИТ-специалистов и их распространению среди представителей других профессий.

Таким образом, в настоящее время потенциальный пользователь программного продукта может не только сформулировать свои пожелания к нему, но и описать их в формальном виде, например с помощью унифицированных графических диаграмм (IDEF, UML, PBMN, EPC и т. д.). Также заказчик может представить свое видение пользовательского интерфейса желаемой системы в виде макетов веб-сайта или мобильного приложения, созданных с помощью соответствующих программ и сервисов, подробно рассмотренных в [20]. Пользователь не только знает, что он хочет, но и как — с объявлением технических деталей, таких как ЯП, фреймворк и т. д. Более того, некоторые заказчики могут представить начальный прототип системы, разработанный самостоятельно. Дальнейшее усиление этой тенденции с учетом превалирующего сегодня SaaS-подхода к предоставлению продукта как услуги возможно в виде появления сервисов по автоматическому формированию программ на основании требований заказчика и пользователя в одном лице. Применяя данный тренд к ранее рассмотренному примеру автоматизации этапов разработки веб-приложений, показанному на рис. 5, получаем сервисную модель автоматической генерации программного продукта, функционирующего по принципу декларативного программирования от объявления требований к результату, т. е. модель «черного» или «серого» ящика (рис. 6). Степень «черноты» ящика определяется уровнем знаний и умений заказчика-пользователя в сфере ИТ.

Следует отметить, что описанная сервисная модель будет предоставлять услуги по генерации готового продукта «под ключ», включая разработку сопутствующей программной

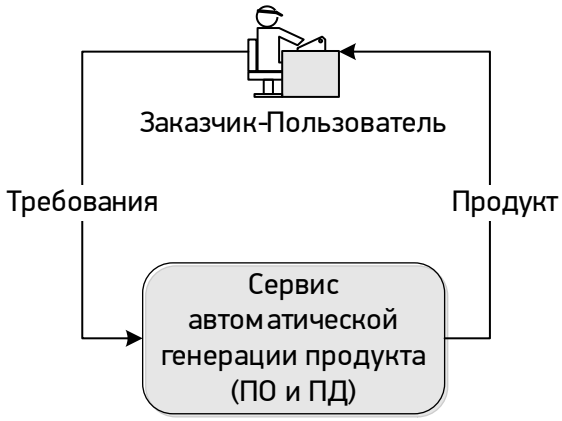

Рис. 6. Черный ящик сервисной модели разработки ПО

Fig. 6. The black box of service-oriented software development model

документации и сопровождение. В настоящее время подобные сервисы по автоматической генерации веб-приложений существуют, например wix.com, weebly.com, a5.ru и пр. В рамках «мобилизации» развиваются аналогичные платформы, предоставляющие готовые графические и программные решения для мобильных устройств [21]. С учетом дальнейшего возрастания доли веб- и мобильных приложений на рынке информационных систем, а также тенденций непрерывной автоматизации процессов ИТ-отрасли с применением инструментов так называемого «искусственного интеллекта» (машинное обучение, интеллектуальный анализ данных и пр.) можно сделать вывод о повсеместном распространении вышеописанной сервисной модели разработки ПО. Эта ситуация влечет за собой еще большее разделение профессий в сфере ИТ: не только по ролевому признаку решаемых задач (аналитики, архитектор, программист, тестировщик и т. д.) и используемым технологиям (языкам и платформам программирования, фреймворкам, паттернам и пр.), а также по уровню владения ими. Наглядно это иллюстрирует аналогия со всеобщей автомобилизацией: сегодня фактически каждый является водителем машины, но лишь немногие подробно знают ее внутреннее устройство, и только

единицы (профессиональные механики, электрики и т. д.) могут обнаружить и починить возникшую неисправность автомобиля конкретной марки и модели. Возможно, это естественный путь развития техники и технологий, в том числе информационных. Таким образом, тренд узкой специализации расширяется ортогональной концепцией об универсальном ИТ-профессионале, который разбирается практически во всех направлениях данной отрасли.

Заключение

Обобщая все вышесказанное о качественных изменениях процесса разработки ПО, можно выделить следующие тенденции автоматизации данной деятельности:

- унификация приемов и технологий — популяризация лучших практик и их представление в виде шаблонов, стандартов, методик и библиотек;
- контейнеризация разработки — скрытие и интеграция деталей в отдельные компоненты, пригодные к использованию по принципу конструктора;
- сервисный подход и миграция в облако — замена десктопных продуктов, средств разработки и развертывания SaaS-, PaaS-, IaaS- и DaaS-сервисами;
- сквозная интеграция процессов и данных — использование баз знаний, агрегирующих информацию о программном продукте на всех этапах и непосредственно связанных с прикладными инструментальными средствами (системы управления проектами, среды разработки и т. д.);
- популяризация декларативных подходов к программированию и языков, ориентированных на математические вычисления и статистическую обработку больших данных;
- мультипарадигмальность и повышение уровня абстракции языков;
- упаковка всех этапов разработки ПО в единую сервисную модель автоматической генерации продукта по типу «черного ящика»;
- объединение профессиональных ролей (заказчик, пользователь, менеджер, аналитик, программист, тестировщик и прочие в одном лице);
- разделение среди ИТ-профессионалов по используемым технологиям, а среди пользователей — по уровню знаний и владению популярными инструментами.

На основании выполненного аналитического обзора технологий, в рамках которых реализуются перечисленные тенденции, можно сформулировать основную общую характеристику современного инструментария разработки программного обеспечения. Она представляет собой непрерывное повышение уровня абстракции, при котором прикладные технические особенности, такие как, например, непосредственный доступ к внутренней памяти устройства и т. д., скрыты от разработчика. Выполнение таких функциональных операций обеспечивается средствами разработки (средой, языком программирования, фреймворком и т. п.) и фактически не является сферой ответственности человека. Подобная автоматизация, с одной стороны, заметно облегчает процесс разработки программного обеспечения, но также обусловливает повышение требований к компетенциям ИТ-специалистов, поскольку в случае возникновения каких-либо проблем с программным обеспечением, необходимо идентифицировать и устранить источник их появления.

Итак, несмотря на то что использование современных методических и прикладных инструментов разработки ПО позволяет автоматизировать выполнение всех шагов этого процесса: от анализа требований до внедрения и сопровождения, и годится как для ИТ-специалистов, так и для представителей других профессий, данный факт не освобождает разработчика от необходимости практического знания автоматизируемых этапов и умения выполнить их самостоятельно (в ручном режиме).

Список литературы

1. Инструментальные средства информационных систем: учебное пособие / А. А. Вичугова; Томский политехнический университет. Томск: Изд-во Томского политехнического университета, 2015. — 136 с.
2. *Royce Winston*. Managing the Development of Large Software Systems. 1970. URL: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
3. *Макконнелл Стив*. Влияние итеративных подходов на предварительные условия // Совершенный код = CodeComplete. Питер: Русская Редакция, 2005. — 896 с.
4. *Forsberg K. and Mooz H*. The Relationship of Systems Engineering to the Project Cycle // First Annual Symposium of the National Council On Systems Engineering (NCOSE). 1991. — 12 p.
5. *Мартин Роберт С*., *Ньюкирк Джеймс В*., *Косс Роберт С*. Быстрая разработка программ. Принципы, примеры, практика = Agile soft ware development. Principles, Patterns, and Practices. М.: Вильямс, 2004. — 752 с.
6. *Chris Anderson*. Pro Business Applications with Silverlight 5. Apress, 2012. — 708 p.
7. *Мартин Р*., *Мартин М*. Принципы, паттерны и методики гибкой разработки на языке C#: пер. с англ. СПб.: Символ-Плюс, 2011. — 768 с.
8. *Фаулер М*. Шаблоны корпоративных приложений (Signature Series) = Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series). М.: Вильямс, 2012. — 544 с.
9. *Martin Björemo*. PredragTrninić // Master's Thesis. Evaluation of web application frameworks with regards to rapid development. Gothenburg: Chalmers University of Technology. 2010. — 41 p. URL: http://publications.lib.chalmers.se/records/fulltext/123847.pdf
10. Wiki сервисы и движки. URL: http://www.intranetno.ru/tools/wiki/
11. 7 ways Github has changed the open source world // HonzaPokorny. 2011; Семь аспектов, как Github изменил мир OpenSource (перевод). URL: http://habrahabr.ru/post/115403/
12. *Гореткина Елена*. Соревнование контейнерных технологий: DockervsRocket // PCWeek/RE. 2015. 17 февраля. №2 (879).
13. *Вичугова А. А*. Этапы, методы и средства конфигурирования информационных систем // Прикладная информатика. 2015. №. 3 (57). С. 88–99.
14. Общая методология разработки и внедрения, сроки и цены. URL: http://www.htg.ru/pages/metodology.php
15. *Sanchez Andrew*. Technical Support Essentials: Advice to Succeed in Technical Support. Apress, 2010. — 260 p.
16. Сервисы поддержки клиентов. URL: http://www.helpdeski.ru/
17. *Чезарини Ф.*, *Томпсон С*. Программирование в Erlang = ErlangProgramming. М.: ДМК Пресс, 2012. — 488 с.
18. *Ershov A. P.* Programming, the Second Literacy // Computer and Education: Proc. IFIP TC-3 3rd World Conf. on Computer Education — WCCE 81. Lousanne, Amsterdam, 1981. Part 1. P. 1–17.
19. *Одинцов И. О*. Профессиональное программирование. Системный подход. 2-е изд., перераб. и доп. СПб.: БВХ-Петербург, 2006. — 624 с.
20. Обзор инструментов для прототипирования пользовательских интерфейсов // Хакер. 2014. Сентябрь. URL: https: //xakep.ru/2014/09/09/prototype-tools/
21. 27 лучших онлайн-конструкторов для создания мобильных приложений (20.01.2015). URL: http://www.coolmobmasters.com/mobilnaya-razrabotka/3847-mobile-apps-make-constructors. html

References

1. Vichugova A. A. *Instrumentalnie sredstva informatsionnih sistem: Uchebnoeposobie* [Tools to customize the information systems: A Textbook]. Tomsk, Tomsk Polytechnic University Publ., 2015. 140 p.
2. Royce Winston, 1970. Managing the Development of Large Software Systems. Available at: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall. pdf (accessed12.02.2016).
3. McConnell Steve. *Code Complete*. Microsoft Press, 2005. 896 p.
4. *Forsberg K. and Mooz H*. The Relationship of Systems Engineering to the Project Cycle // First Annual Symposium of the National Council On Systems Engineering (NCOSE). 1991. 12 p.
5. Martin Robert C. *Agile Software Development, Principles, Patterns, and Practices*. Pearson New International Edition, 2013. 752 p.
6. Chris Anderson. *Pro Business Applications with Silverlight* 5. 2012. Apress. 708 p.
7. Martin Robert C., Martin Micah. *Agile software development. Principles, Patterns, and Practices*. Prentice Hall, 2011. 768 p.
8. Fowler M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2012. 544 p.
9. Martin Björemo, PredragTrninić. *Master's Thesis. Evaluation of web application frameworks with regards to rapid development*. Gothenburg: Chalmers University of Technology. 2010. 41 p. Available at: http://publications.lib.chalmers.se/records/fulltext/123847.pdf (accessed 12.02.2016).
10. Wikiservices and drivers. Available at: http://www.intranetno.ru/tools/wiki/ (accessed12.02.2016).

11. 7 ways Github has changed the open source world. *Honza Pokorny*, 2011. Available at: http://habrahabr.ru/post/115403/ (accessed12.02.2016).
12. Goretkina Elena. Docker vs Rocket: competition of container technologies. *PC Week/RE*, 2015, no. 2 (879). Available at: http://www.pcweek.ru/foss/article/detail.php? ID=170306 (accessed12.02.2016).
13. Vichugova A. A. *Etapi, metodi i sredstvakonfigurirovaniyainformatsionnihsystem* [Stages, methodsandtoolsfor customization information systems]. *Prikladnaya Informatika* — Journal of Applied Informatics, 2015, no. 3 (57), pp. 88–99.
14. The general methodology for development and implementation of, the terms and prices. Available at: http://www.htg.ru/pages/metodology. php (accessed12.02.2016).
15. Sanchez Andrew. *Technical Support Essentials: Advice to Succeed in Technical Support*. Apress, 2010. 260 p.
16. Customer support services. Available at: http://www.helpdeski.ru/ (accessed12.02.2016).
17. Cesarini Francesco, Thompson Simon. *Erlang Programming*. O'Reilly Media, Inc., 2009. 498 p.
18. Ershov A. P. Programming, the Second Literacy. *Computer and Education*: *Proc. IFIP TC-3 3rd World Conf. on Computer Education — WCCE 81*. Lousanne, Amsterdam, 1981, part 1, pp. 1–17.
19. Odintsov I. O. *Professionalnoe programmirovanie. Sistemniy podhod*. [Professional programming. System approach]. Saint Petersburg, BVH-Petersburg, 2006. 624 p.
20. *Obzor instrumentov dlya prototipirovaniya polzovatelskih interfeisov* [Review of tools to develop prototypes of user interfaces]. Hacker. 2014. September. Available at: https: //xakep.ru/2014/09/09/prototype-tools/ (accessed 15.02.2016).
21. *27 luchihonlain-konstruktorov dlya sozdaniya mobilnih prilojeniy* [27 best tools to develop mobile apps] (20.01.2015). Available at: http://www.coolmobmasters.com/mobilnaya-razrabotka/3847-mobile-apps-make-constructors.html (accessed15.02.2016).

A. Vichugova, *Tomsk Polytechnic University, Tomsk, Russia, vichugovaaa@tpu.ru*

Methods and tools to automatize software development process

The complexity of information processing caused the increasing the level of software abstraction and forces the changes of the process of creating this type of products. Functional stages of the software development process in practice are implemented in a variety of models and methodologies. Needs to reduce the time of software development led to new approaches, methods and tools of the organization, implementation of this process. The paper outlines the modern trends in the reduction of routine operations in designing, coding, testing, deployment and documentation software, and analyzed the implementation of these tendencies in the form of a series of tools to automate one or several aspects of software development. The following aspects are discussed in the research: unification of techniques and technologies, containerization, migration to the cloud, continuing integration of processes and data, promotion of declarative programming approaches and languages, focused on math and statistical processing of BigData, multiparadigmality of languages and raising the level of their abstraction. There are examples of transformation of classical concepts of software development to the present state through all steps of the process: from requirements analysis to implementation and maintenanceaccording to the current needs and capabilities of the IT market. The possible ways of changing the users and professionals competencies in the field of information technology (IT) are described.

Keywords: integrated development environment, frameworks, paradigms and programming languages.

About author: A. Vichugova, *PhD in Technique*

For citation: Vichugova A. Methods and tools to automatize software development process. *Prikladnaya Informatika* — Journal of Applied Informatics, 2016, vol. 11, no. 3 (63), pp. 63–75 (in Russian).