инженерия программного обеспечения

Шуйский Илья Юрьевич

УДК 004.891.2

РАЗРАБОТКА СИСТЕМЫ INTELLIGENT ADVISOR ON SOFTWARE DEVELOPMENT НА ОСНОВЕ KNOWLEDGE.NET

Аннотация

В статье освещаются существующие методологии и подходы к разработке программного обеспечения. Отмечаются их преимущества и недостатки и подчеркивается необходимость выбора правильного метода разработки. Описывается концепция экспертной системы для определения такого метода на основе технологии Knowledge.NET, разработанной на кафедре информатики СПбГУ под руководством проф. В.О. Сафонова.

Ключевые слова: разработка, программное обеспечение, Knowledge.NET, экспертная система.

ВВЕДЕНИЕ

Процесс разработки программного обеспечения происходит постоянно. За несколько десятилетий были разработаны определенные методологии этого процесса, которые улучшают качество, надежность и продуктивность разработки. Компании, занимающиеся созданием, распространением и поддержкой ПО, как правило, выбирают наиболее подходящую для них по определенным параметрам методологию и следуют ей при работе над всеми проектами. Однако не существует универсального подхода, все методологии имеют как сильные, так и слабые стороны, и нередко, выбирая одну из них, мы теряем преимущества многих других. Сделать такой выбор трудно и, естественно, совершаются ошибки, при которых теряются как время и ресурсы, затраченные на разработку, так и престиж компании и ее сотрудников.

Система, которая давала бы экспертные рекомендации по выбору методологии разработки программного обеспечения, была бы очень полезна. Она позволила бы как улучшить процесс самой разработки, так и предохранить от возможных ошибок, исправление которых будет очень сложным и трудоемким.

СУЩЕСТВУЮЩИЕ МЕТОДОЛОГИИ И ПОДХОДЫ

Поиск повторяющихся, предупреждаемых процессов, облегчающих разработку ПО, а также формализация этих процессов и приведение их к понятному и применимому виду занял десятилетия. Но процесс этот был не напрасным, и в настоящее время существует несколько устоявшихся подходов, которыми пользуются организации при разработке программного обеспечения. Приведем эти подходы, указав их сильные и слабые стороны.

1. Водопадная модель разработки ПО [2]. При таком подходе процесс разработки разделяется на несколько этапов-ступеней, которые выполняются последовательно. Существует несколько вариантов наборов этих этапов, укажем наиболее полный и распространенный из них:

1.1. *Анализ требований к продукту.* На этом этапе аналитик знакомится с предметной областью разрабатываемого ПО, с ее спецификой и терминологией. После этого происходит анализ требований заказчика с целью выявления незаконченных и неясных моментов. Полученные и оговоренные требования необходимо отобразить в документе, что также нередко помогает в обсуждениях, возникших уже после разработки программного продукта.

1.2. *Спецификация*. На этой ступени происходит детальное описание разрабатываемого программного продукта вплоть до мелочей. Наибольшее значение спецификации имеют для внешних интерфейсов, стабильность которых наиболее важна.

1.3. *Проектирование (Архитектура)*. На этом этапе создается общая архитектура ПО и алгоритмы согласно спецификации. Также на этой стадии решаются проблемы интерфейсов между ПО и операционной системой или оборудованием.

1.4. *Реализация (Кодирование)*. На этой ступени создается сам программный код продукта.

1.5. *Тестирование.* Важная часть разработки ПО. На этом этапе происходит детальная проверка созданного продукта – его функциональности, надежности, безопасности и пр.

1.6. *Документирование.* Необходимо для того, чтобы в дальнейшем было проще поддерживать программный продукт. Может включать в себя описание внешних и внутренних интерфейсов.

1.7. *Распространение (установка).* Начинается после того, как программный код достаточно оттестирован и работает стабильно, что позволяет выпускать релиз.

1.8. *Техническая поддержка.* На этом этапе происходит решение проблем, возникающих при использовании ПО, а также его доработка и улучшение.

Модель водопада является самой известной методологией, на основных ее принципах построены многие другие подходы к разработке ПО. К плюсам можно отнести понятность, простоту и сходность такого подхода к подходам разработки других, не программных систем, а также четкий регламент процесса, который легко можно контролировать. Однако у этой методологии есть свои минусы: процесс разработки сложно распараллеливать; после каждого этапа необходимо представить законченный результат (спецификацию, проектирование и пр.), а на практике оказывается, что все этапы связаны между собой, и разделить их абсолютно – не так просто. Кроме того, интеграция всего проекта происходит в самом конце, что вызывает интеграционные проблемы. Также и заказчик видит работающую систему только после прохождения всех этапов и не имеет возможности влиять на ее разработку.

2. Итерационный подход. Этот метод предполагает разработку небольшого ядра разрабатываемой системы, к которому впоследствии добавляются остальные функции. Плюсы такого подхода очевидны: методология позволяет легко менять программный код при изменении требований, что не приведет к необходимости полностью перестраивать систему. Также стоит отметить, что итерационный подход часто применяется при гибкой модели разработки, которая предполагает наличие обратной связи от заказчика как контролирующий механизм (взамен анализа и спецификации требований), что приводит к созданию более удобной и подходящей системы. Однако у такого подхода есть и минусы: если в самом начале не оговариваются все требования к продукту и они меняются по ходу его разработки, то такой процесс ничем не регламентирован и может продолжаться до бесконечности. Однако исследования показывают, что в целом наличие обратной связи оказывает положительное влияние на производительность.

3. Спиральная модель [5]. Эта модель была предложена в 1988 году Бэри Боэмом

для преодоления недостатков водопадной модели. По своей сути она похожа на итерационный подход. Здесь также процесс разделен на этапы – «витки», и на каждом из них создается некая работающая система, которая постепенно становится все ближе к разрабатываемой. Однако в спиральной модели каждый виток может быть построен по принципу водопада или еще по какой-то другой схеме, на нем всегда есть определение целей, разработка альтернатив, разработка и тестирование и планирование следующих итераций. На витке может быть определение и анализ требований, разработка новой функциональности или создание новой компоненты. В отличие от водопадной модели, в спиральной нет предопределенного и обязательного набора витков, их количество определяется в процессе разработки, как и при итерационном подходе. К минусам спиральной модели можно отнести все недостатки итерационного подхода. Однако она объединяет в себе преимущества водопадного и итерационного метода, становясь при этом достаточно сложной и громоздкой.

4. Экстремальное программирование [2]. Этот подход также относится к гибкой модели разработки и является одним из самых известных ее представителей. Он предполагает итеративность и тесное общение с заказчиками через обратную связь. Отличительные особенности экстремального программирования:

- Разработка делится на очень короткие циклы, которые могут занимать недели или дни (вместо месяцев и даже лет при водопадной модели)
- Каждый такой цикл начинается с написания модульных тестов, которые определяются целями и задачами данного этапа
- После этого происходит написание такого программного кода, который делает написанные тесты успешно выполняющимися, причем принимается всегда самое первое и простое подходящее решение, далее код может быть подвергнут переработке.

К плюсам экстремального программирования можно отнести его скорость, а также разработчики не загружены необходимостью писать различные отчеты, проектировать множество моделей. Процесс во многом построен на квалификации и самодисциплине. Однако, естественно это таит в себе и недостатки – экстремальное программирование легко может перерасти в неуправляемый хаос, который трудно контролировать. Также ошибки, которые допущены на ранних стадиях (например, при анализе требований), требуют огромных затрат на исправление в дальнейшем.

Существуют и другие методологии разработки ПО, однако все они так или иначе построены на указанных выше и имеют свои недостатки и преимущества.

ПОСТАНОВКА ЗАДАЧИ

В предыдущей главе мы рассмотрели основные подходы к процессу разработки программного обеспечения. Все они имеют как сильные, так и слабые стороны, поэтому при выборе методологии разработки конкретного продукта следует тщательно исследовать существующие, использовать их преимущества и избегать недостатков, комбинируя и выбирая до начала работы и в процессе при перехода от одного этапа к другому. В настоящее время этим занимается команда разработчиков ПО, принимая решения на основе собственного опыта и практики компании. Существование экспертной системы, осуществляющей эти функции и являющейся своего рода помощником разработчика, сильно улучшит этот процесс. Предполагаемое название системы – Intelligent advisor on software development. Она будет содержать данные об архитектуре разрабатываемого ПО и ее особенностях, а также о всевозможных методологиях процесса разработки. На основе этой информации, система будет давать экспертные рекомендации по выбору того или иного подхода к процессу разработки на различных его этапах. Создание этой системы можно разбить на две основные подзадачи:

- Детальное изучение существующих методологий с целью выявления возможностей их взаимной интеграции. Поиск точек перехода, то есть тех моментов в процессе

разработки ПО, в которых возможен переход от одной методологии к другой, а также поиск зависимостей преимуществ методологий от архитектуры разрабатываемой системы.

- Разработка такой системы на основе системы Knowledge.NET и одноименного языка представления знаний.

Intelligent advisor on software development, являясь экспертной системой, хранит свои данные в базе знаний, для этого необходим язык представления знаний. В качестве такого языка решено было взять Knowledge.NET – перспективный язык представления гибридных знаний, который является расширением языка C# и легко интегрируется с платформой .Net. Выбор обусловлен тем, что методы инженерии знаний и программной инженерии стоят друг от друга достаточно далеко, а технология Knowledge.NET позволяет интегрировать систему разработки и использования баз знаний с одной из наиболее перспективных технологий программной инженерии – Microsoft.Net. Опишем основные принципы и конструкции языка Knowledge.NET.

ОСНОВНЫЕ КОНСТРУКЦИИ KNOWLEDGE.NET

Язык Knowledge.NET [3] является расширением языка C# средствами представления гибридных знаний. В их основе лежит концепция *онтологии* – разновидность спецификации предметной области, выраженной в терминах ее концептов (понятий) и их взаимосвязей. Приведем кратко основные конструкции языка Knowledge.NET, позволяющие представлять онтологические знания [1].

1. Концепты. Для представления множества экземпляров одного типа в онтологической модели используются концепты, аналоги классов в объектно-ориентированном подходе

1.1. *Подконцепты.* Концепты также могут наследоваться (конкретизироваться):

```
A si_subconcept_of B,C
```

1.2. *Перечеслимые концепты* могут быть определены указанием всех своих экземпляров:

```
Enumerated concept A is_one_of
Individual1, Individual2…
```

1.3. *Объединение, пересечение и дополнение концептов*. Концепт по своей сути является множеством, и с ним можно проводить стандартные операции из теории множеств:

```
Concept A is_union_of B,C
Concept A is_intersection_of B,C
Concept A is_complement_of B,C
```

2. Свойства. Свойство представляет связь между двумя сущностями. Для каждого свойства должен быть определен домен **domain** и область значений **range**

```
datatype property PropertyName
{
        domain B;
        range int;
}
```

Здесь *B* – идентификатор концепта, *int* – простой тип. Возможно множественное указание как типов, так и концептов.

2.1. *Простые свойства*. Простые свойства связывают экземпляр со значением некоторого простого типа данных, например с целым числом *int*, как в вышеуказанном примере.

2.2. *Объектные свойства*. Объектное свойство связывает между собой два экземпляра

```
object property PropertyName
{
        domain B;
        range R;
}
```

Здесь *R* – концепт.

Свойства, так же как и концепты, образуют иерархическую структуру (подсвойства). Домен и область значения соответственно сужаются

```
object property P
          is_subproperty_of Q
{
        domain B;
        range R;
}
```

Множественное наследование свойств невозможно.

Объектное свойство может иметь соответствующее обратное свойство, обозначается словом *inverse*

```
object property P
{
        domain B;
        range R;
        inverse isP;
}
```

Еще одна разновидность объектных свойств – функциональные. Отличительной особенностью функциональных свойств является то, что они связывают экземпляр не более чем с одним экземпляром концепта, для этого существует идентификатор *functional*

```
functional object property
          ObjectPropertyName
```

Существуют и обратные функциональные свойства

```
functional inverse
        isObjectPropertyName
```

2.3. *Аннотации*. Аннотации являются важным типом свойств. С их помощью можно добавить описание к концептам, свойствам и пр.

```
Annotation
{
        version info «text»;
        label «text»;
        comment «text»;
        created_by «text»;
}
```

3. Ограничения. Свойства могут быть использованы для задания ограничений. Каждому ограничению удовлетворяет ноль и более экземпляров, поэтому ограничения можно рассматривать в качестве концептов. Поддерживаются три основных типа ограничений:

3.1. *Ограничения по квантору*. Это ограничение состоит из трех компонентов: свойства, квантора (*some_values_from* или *al_values_from*) и наполнителя (концепт или простой тип)

```
PropertyName some_values_from
                ConceptName
```

3.2. Ограничение по мощности. Это ограничение на количество связей с другими экземплярами или простыми типами. Поддерживаются такие ограничения: не более чем (*max_cardinality*), не менее чем (*min_cardinality*), точное количество (*cardinality*)

```
PropertyName max_cardinality 8
```

3.3. Ограничение по значению. С помощью этого ограничения можно создавать концепты, для определения которых необходимо существование отдельных экземпляров. Обозначаются ключевым словом *has_value*.

4. Экземпляры. Определяют объекты в выбранной онтологии, отличаются от объектов в ООП тем, что один экземпляр может иметь несколько имен. Задается ключевым словом *individual*; концепты, которые реализует этот экземпляр, ставятся после ключевого слова *is_a*:

```
Individual IndividualName
{
     is_a ConcepName;
}
```

5. Язык запросов. Язык запросов позволяет выбирать из онтологии экземпляры по некоторому заданному критерию, он является инструментом для экспертной деятельности на заданной онтологии. Критерий запросов может быть основан как на некотором наборе концептов и значениях их свойств, так и только на значениях свойств или только содержит перечисление концептов.

5.1. *Синтаксис запроса*. Синтаксически запрос представляет собой текстовую строку, которая в общем виде выглядит следующим образом:

```
Individuals of concepts
  ConceptName1, ConceptName2…
        Where Expression1
        Where expression2
        ...
```

В разделе *Expression* задается критерий – ограничение на свойства выбираемых объектов.

5.2. *Выполнение запроса*. Выполнение запроса происходит с помошью класса *QueryEngine*, используется статический метод *ICollection<Individual> QueryEngine .execute(string query)*. Принимая в качестве параметра строку запроса, данный метод возвращает коллекцию экземпляров, которая ему соответствует.

ПРИНЦИПЫ РАБОТЫ СИСТЕМЫ

Система «Intelligent advisor on software development на основе Knowledge.NET» предназначена для решения проблемы выбора метода разработки ПО. Она содержит в своей базе знаний несколько основных элементов, представленных в виде онтологий, каждая из которых описана на языке Knowledge.NET:

- Онтология 1 (Онтология системы) – описывает разрабатываемый программный комплекс. Данные этой онтологии могут быть получены с помощью системы извлечения знаний Knowledge Prospector [3] из технической и функциональной спецификации или других документов, после чего, при необходимости, обработаны вручную. Данная онтология состоит из объектов разрабатываемой системы, их отношений и определений предполагаемых действий над этими объектами. Онтология 1 может меняться со временем, если меняются требования к программному продукту. Это может произойти как по инициативе заказчика, так и по ряду каких-либо других причин, произошедших в ходе разработки.
- Онтология 2 (Онтология текущего состояния) – описывает разработанную на данный момент систему (в начале разработки эта онтология пуста), постепенно в нее добавляются элементы, процесс разработки которых закончен.
- Онтология 3 (Онтология предметной области) – содержит знания о специфике предметной области. В нее входит информация о взаимной зависимости объектов, в частности о том, что одни из них используют в своей работе другие. Это позволит определить порядок разработки программного обеспечения.
- Онтология 4 (Онтология методов разработки) – содержит экспертные знания о том, какие методологии больше всего подходят для разработки тех или иных элементов онтологии.

Работа экспертной системы происходит по виткам. Каждый виток можно разделить на три основных этапа:

1. Определения различий между Онтологией 1 и Онтологией 2, тем самым, определяется та часть системы, разработка которой еще не произведена.
2. На основе полученной информации и Онтологии 3 система определяет порядок разработки новых элементов.
3. Экспертные рекомендации. На этом этапе на основании Онтологии 4 система выдает рекомендации о том, какой набор элементов следует разрабатывать в первую очередь и каким методом разработки следует воспользоваться.

После разработки новых элементов они добавляются в Онтологию 2. Если она совпадает с Онтологией 1, то процесс разработки заканчивается, в другом случае начинается новый виток работы системы.

ЗАКЛЮЧЕНИЕ

В статье описана концепция системы «Intelligent advisor on software development на основе Knowledge.NET». Это экспертная система, предназначенная для определения методологии процесса разработки программного обеспечения. Такая система будет полезна, так как позволит использовать преимущества и избегать недостатков существующих подходов к разработке ПО.

Разработка такой системы состоит из двух этапов. На первом из них необходимо определить возможности интегрируемости существующих методологий друг с другом, второй состоит в реализации такой системы на базе системы Knowledge.NET.

Литература

1. *Сафонов В.О., Новиков А.В. Сигалин М.В., Смоляков А.Л., Черепанов Д.Г.* Интеграция методов инженерии знаний и инженерии программ: Система управления знаниями Knowledge.NET // Компьютерные инструменты в образовании, 2005, № 5. С. 52–68.

2. *Кознов Д.В., Бугайченко Д.Ю.* Введение в программную инженерию // Интернет-университет информационных технологий, 2009 // intuit.ru

3. *Safonov V.O., Novikov A.V., Smolyakov A.V., Cherepanov D.G., Sigalin M.A.* Knowledge.NET ontology-based knowledge management toolkit for Microsoft.NET // .NET Technologies 2006 International Conference. Univ. of West Bohemia Campus Bory, May 29 – June 1, 2006, Pilsen, Czech Republic. Poster Paper Proceedings.

4. Web-страницы проекта Knowledge.NET // http://www.knowledge-net.ru

5. Web-страницы сайта http://ru.wikipedia.org

Abstract

In this article the existing methodologies of Software Development are shown. Advantages and disadvantages of these methodologies are described. The necessity of choosing of the right method is highlighted. The concept of the expert system which is given such recommendations is described. The system is based on the Knowledge.NET technology, developed at the Department of Computer Science of Saint-Petersburg State University under the guidance of Prof. Safonov.

Keywords: development, software, Knowledge.NET, expert system.

Шуйский Илья Юрьевич,
аспирант 3 курса кафедры информатики математико-механического факультета СПбГУ,
ilja.shujskij@gmail.com