Программная инженерия

Инструменты и методы программной инженерии (Software Engineering Tools and Methods)

Глава базируется на IEEE Guide to the Software Engineering Body of Knowledge[1] - SWEBOK®, 2004. Содержит перевод описания области знаний SWEBOK® “Software Engineering Tools and Methods”, с комментариями и замечаниями[2].

Сергей Орлик.

Программная инженерия

Инструменты и методы программной инженерии (Software Engineering Tools and Methods)

Программные инструменты предназначены для обеспечения поддержки процессов жизненного цикла программного обеспечения. Инструменты позволяют автоматизировать определенные повторяющиеся действия, уменьшая загрузку инженеров рутинными операциями и помогая им сконцентрироваться на творческих, нестандартных аспектах реализации выполняемых процессов. Инструменты часто проектируются с целью поддержки конкретных (частных) методов программной инженерии, сокращая административную нагрузку, ассоциированную с “ручным” применением соответствующих методов. Так же, как и методы программной инженерии, инструменты призваны сделать программную инженерию более систематической деятельностью и по своему содержанию (предлагаемой функциональности) могут варьироваться от поддержки отдельных индивидуальных задач вплоть до охвата всего жизненного цикла (в этом случае часто говорят об инструментальной платформе или просто платформе разработки, *прим. автора*).

Методы программной инженерии накладывают определенные структурные ограничения на деятельность в рамках программной инженерии с целью приведения этой деятельности в соответствие с заданным систематическим подходом и более вероятным и скорым, с точки зрения соответствующего метода, достижением успеха. Методы обычно предоставляют соответствующие соглашения (нотацию), словарь <терминов и понятий> и процедуры выполнения идентифицированных (и охватываемых методом, *прим. автора*) задач, а также рекомендации по оценке и проверке <выполняемого> процесса и <получаемого в его результате> продукта. Методы, как и инструменты, варьируются по содержанию (охватываемой области применения, *прим. автора*) от отдельной фазы жизненного цикла (или даже процесса, *прим. автора*) до всего жизненного цикла. Данная область знаний касается только методов, охватывающих множество фаз (этапов) жизненного цикла. Те методы, применение которых фокусируется на отдельных фазах жизненного цикла или частных процессах, описаны в соответствующих областях знаний.

Существует множество детальных описаний и руководств по конкретным инструментам, и исследований, посвященных анализу (и категоризации, в первую очередь, со стороны аналитиков, *прим. автора*) уже применяемых и новых инструментальных средств (и вероятным направлениям их развития, *прим. автора*). В таком контексте, общее техническое описание инструментов программной инженерии, действительно, может отпугнуть. (В то же время, с точки зрения автора, при всей неоднозначности любой категоризации инструментов, может быть сформирован общий взгляд на их целевую функциональность, пусть в чем то и спорный, что, отразится в определенных случаях ниже в соответствующих авторских комментариях, *прим. автора*). Одна из основных сложностей такого описания, в общем случае, заключается в высокой изменчивости и быстром

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

эволюционировании программных инструментов. Конкретные аспекты функциональности инструментов достаточно быстро изменяются, что усложняет приведение конкретных актуальных примеров.

Данная область знаний охватывает все процессы жизненного цикла и, соответственно, связана со всеми другими областями знаний SWEBOK.

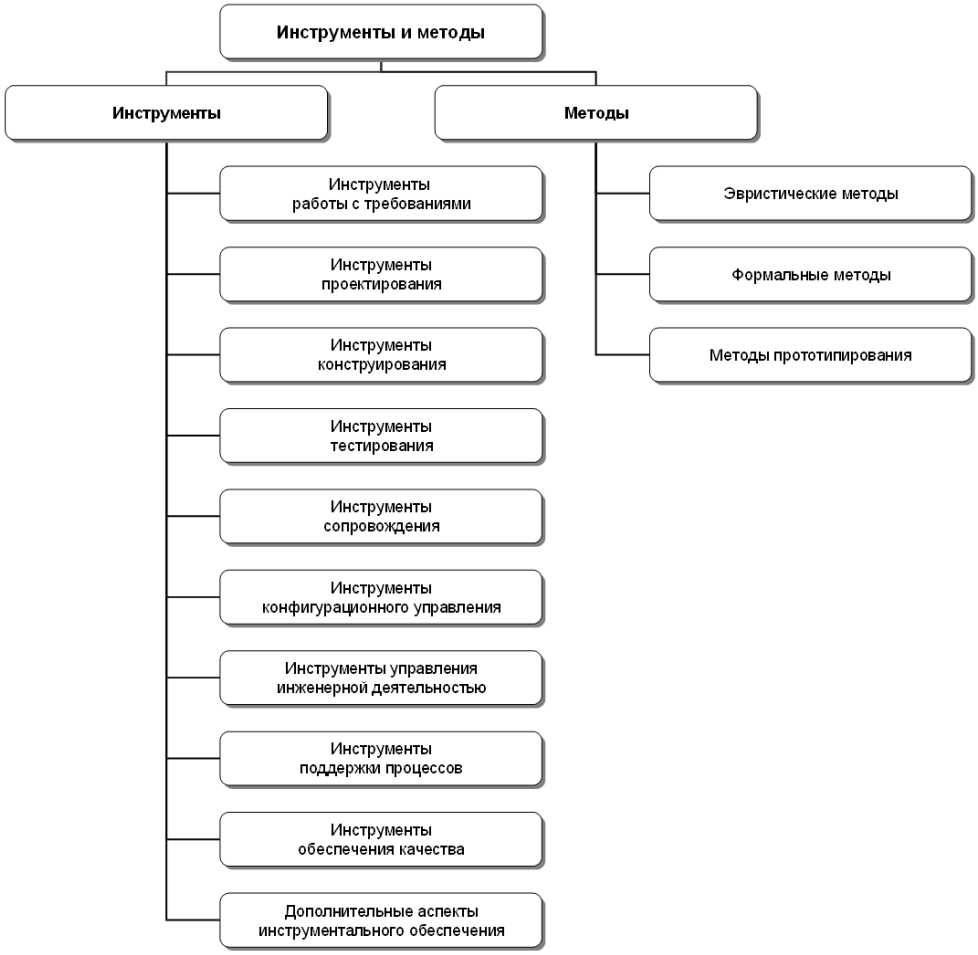

Рисунок 1. Область знаний “Инструменты и методы программной инженерии” [SWEBOK, 2004, с.10-1, рис. 1]

1. Инструменты программной инженерии (Software Engineering Tools)

Первые пять тем данной секции соответствуют первым пяти областям знаний SWEBOK - требования, проектирование, конструирование, тестирование и сопровождение. Следующие четыре темы касаются оставшихся четырех областей знаний – конфигурационного управления, управления программной инженерией, процессов и качества, соответственно. Также в данной секции представлена еще одна тема – “Дополнительные аспекты инструментального обеспечения” (в оригинале SWEBOK она называется “Miscellaneous”), посвященная таким вопросам как, например, интеграция инструментов, которые потенциально касаются всех классов инструментов.

1.1 Инструменты работы с требованиями (Software Requirements Tools)

Инструменты, применяемые для работы с требованиями могут быть классифицированы в две категории: средства моделирования (modeling) и средства трассировки (traceability). С точки зрения автора, моделирование требований, все же, является частью управления требований, как, кстати, и трассировка. В принципе, инструменты трассировки могут быть рассмотрены как самостоятельная категория, в силу своей значимости при проведении анализа требований, в первую очередь, анализа влияний требований и изменений (т.н. “impact analysis”). Поэтому, в приведенной ниже классификации авторская модификация состоит только в том, что вместо

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

“инструментов моделирования требований” используется термин “инструменты управления требованиями”, при сохранении оригинального содержания данной темы SWEBOK. В сегодняшней практике

- Инструменты управления требованиями (моделирования требований – Requirements modeling tools). Эти инструменты используются для извлечения (eliciting), анализа, специфицирования и проверки программных требований.

- Инструменты трассировки требований (Requirement traceability tools). Эти инструменты становятся все более важными по мере повышения сложности программного обеспечения. В силе того, что они также относятся и к другим процессам жизненного цикла, здесь они представлены в качестве самостоятельной категории средств работы с требованиями.

По-мнению автора, необходимо отметить, что в современной практике, трассировка является неотъемлемой частью полноценной работы с требованиями, что приводит к естественному объединению предлагаемых SWEBOK категорий инструментов в единый класс “инструментов управления требованиями”, функциональное содержание которых может варьироваться, например, в зависимости от сложности проектов и уровня зрелости процессов. Если мы обратимся, например, к модели CMMI Staged, мы увидим, что на 2-м уровне зрелости речь идет об “управлении требованиями” – Requirement Management, а на 3-м уровне зрелости обсуждается “разработка требований” – Requirement Development, обладающая более ёмким содержанием. В то же самое время, с технократической точки зрения, требования могут восприниматься и как элементы конфигураций, наравне с запросами на изменения и другими активами проекта (см. область знаний SWEBOK “Конфигурационное управление”). Таким образом, в ряде случаев (что подтверждается конкретными программными средствами, доступными на рынке программного обеспечения), в качестве инструмента работы с требованиями может выступать и система конфигурационного управления, если, конечно, она изначально не ограничена базовой функциональностью контроля версий <файлов>. С другой стороны, сегодняшние средства моделирования на основе UML и BPMN могут также рассматриваться как элементы инструментального обеспечения работы с требованиями, что часто отражается в их функциональности, включающей тесную интеграцию с “классическими” средствами управления требованиями, а сама интеграция воплощена не только в визуальном представлении работы с репозиториями требований, но и в автоматизации трассировки между моделями (и/или их элементами) и требованиями, соответственно.

1.2 Инструменты проектирования (Software Design Tools)

Эта тема охватывает инструменты для создания и проверки программного дизайна. Существует большое разнообразие таких инструментов, использующих различные нотации (соглашения, в том числе визуальные, *прим. автора*) и методы. Несмотря на такое разнообразие, <авторами SWEBOK> не было найдено <адекватной> классификации этих инструментов.

По мнению автора, в данном случае, все же возможно разделение инструментов по нескольким критериям, например, применяемым базовым нотациям моделирования и проектирования (SADT/IDEF, UML, BPMN/BPEL, Microsoft DSL и т.п.) или целевым задачам (бизнес-моделирование, проектирование БД, объектно-ориентированное проектирование, интеграционное/SOA-проектирование и т.п.).

1.3 Инструменты конструирования (Software Construction Tools)

Данная тема касается инструментальных средств конструирования программного обеспечения, в соответствии с пониманием “конструирования”, заданным соответствующей областью знаний SWEBOK, рассматривавшейся ранее. Эти инструменты используются для производства и трансляции программного представления (например, исходного кода), достаточно детального и явного для машинного выполнения.

- Редакторы (program editors). Эти инструменты используются для создания и модификации <исходного кода> программ и, возможно, ассоциированной с ними документации. Это могут быть как редакторы “общего назначения” (что на протяжении многих лет наблюдается в UNIX и unix-подобных средах, *прим. автора*) или специализированные

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

редакторы с поддержкой специфики целевого языка программирования (что является, в большинстве случаев, прерогативой интегрированных сред разработки – IDE, *прим. автора*).

- Компиляторы и генераторы кода (compilers and code generators). Традиционно, компиляторы являлись неинтерактивными (командными) трансляторами исходного кода. Однако, существует тенденция (с точки зрения автора, более чем явная, что отмечено ниже) интеграции компиляторов и редакторов в интегрированные среды программирования. К этому классу также относятся препроцессоры, линковщики/загрузчики, а также генераторы кода (за исключением, может быть, объектно-ориентированных средств проектирования, поддерживающих связь с исходным кодом и имеющих тенденцию быть тесно интегрированными с новым поколением IDE, *прим. автора*).

- Интерпретаторы (interpreters). Эти инструменты обеспечивают исполнение программ посредством эмуляции. Они могут поддерживать действия по конструированию программного обеспечения, предоставляя для исполнения программ окружение, более контролируемое и поддающееся наблюдению, чем это обычно способна сделать та или иная операционная система.

  Автору хочется отметить определенное “слияние”, если так можно выразиться, между компиляторами и интерпретаторами. Ярким тому свидетельством является использование так называемой just-in-time компиляции – компиляции “на лету”, когда промежуточный программный код, по мере исполнения или с опережением (например, в процессе запуска/загрузки программы) преобразуется в набор инструкций, исполняемых непосредственно средствами операционной системы, но под контролем среды исполнения, в первую очередь, с точки зрения безопасности. Такого рода подход стал родоначальником ряда современных программных платформ, например, Java и .NET. На этом фоне, с точки зрения автора, возможно было бы объединить интерпретаторы с компиляторами и генераторами кода, как средства непосредственной подготовки (трансляции) исходного кода к исполнению.

- Отладчики (debuggers). Эти инструменты было принято выделить в самостоятельную категорию, так как они поддерживают процесс конструирования программного обеспечения, но, в то же время, <функционально> отличаются от редакторов и компиляторов.

По мнению автора, *документирование* все же является не только и не столько частью редактора, сколько самостоятельной функциональностью, пусть часто и тесно интегрированной с редактором. С точки зрения классификации инструментов необходимо выделить явно и давно присутствующие на рынке: *“интегрированные средства разработки”* (IDE - integrated developers environment), а также *программные библиотеки/библиотеки компонент* (frameworks, libraries, components), без которых просто невозможно представить сегодняшний процесс разработки, да и рынок программных средств, в целом. Кроме того, с точки зрения автора, в данной теме можно говорить и о таких функционально ёмких “супер”-категориях, как *“программная платформа”* (например, Java, J2EE и Microsoft .NET) и *“платформа распределенных вычислений”* (например, CORBA и WebServices), которые включают наравне с инструментами, как таковыми, и определенные модели конструирования, преобразования и выполнения кода. При таком подходе, вероятно, обоснованным было бы введение класса “*элементарных инструментов конструирования*”, к которому можно было бы отнести редакторы, компиляторы, интерпретаторы, отладчики, средства документирования и библиотеки, а также класса “*комплексных средств конструирования*” – интегрированных сред и различных платформ, что, безусловно, не претендует на истину в последней инстанции и является одной из возможных точек зрения.

1.4 Инструменты тестирования (Software Testing Tools)

- Генераторы тестов (test generators). Эти инструменты помогают в разработке сценариев тестирования.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

- Средства выполнения тестов (test execution frameworks). Эти средства обеспечивают среду исполнения тестовых сценариев в контролируемом окружении, позволяющем отслеживать поведение объекта, подвергаемого тестированию.

- Инструменты оценки тестов (test evaluation tools). Эти инструменты поддерживают оценку результатов выполнения тестов, помогая определить в какой степени и где именно обнаруженное поведение <тестируемого объекта> соответствует ожидаемому поведению.

- Средства управления тестами (test management tools). Эти средства обеспечивают поддержку всех аспектов процесса тестирования программного обеспечения (выполняя, в какой-то степени, функции “IDE для тестирования”, *прим. автора*).

- Инструменты анализа производительности (performance analysis tools). Эти инструменты используются для количественной оценки и анализа производительности программного обеспечения, являющегося специализированным видом тестирования, цель которого – в оценки поведения программ в части производительности, в отличие от тестирования <корректности> функционального поведения.

Последний класс инструментов тестирования явно подчеркивает, по мнению автора, недостаточность предложенной классификации, упуская, например, инструменты функционального тестирования, средства тестирования безопасности, инструменты тестирования пользовательского интерфейса, инструменты нагрузочного тестирования и др., соответствующие, различным целям тестирования, представленным в секции 2.2 области знаний SWEBOK “Тестирование”, и естественно задающим “подвиды” возможного класса “*специализированных* или *целевых инструментов тестирования*”, к которым, в частности, относится тестирование производительности.

1.5 Инструменты сопровождения (Software Maintenance Tools)

Эта тема охватывает инструменты, особенно важные для обеспечения сопровождения существующего программного обеспечения, подверженного модификациям. SWEBOK идентифицирует две категории таких инструментов:

- Инструменты облегчения понимания (comprehension tools). Эти инструменты помогают человеку в понимании программ. Примерами могут служить различные средства визуализации.

- Инструменты реинжиниринга (reengineering tools). Эти инструменты поддерживают деятельность по реинжинирингу, описанную в области знаний SWEBOK “Software Maintenance”.

Средства “обратного” инжиниринга (reverse engineering) помогают в процессе восстановления для существующего программного обеспечения таких артефактов, как спецификация и описание дизайна (архитектуры), которые, в дальнейшем, могут быть трансформированы для генерации нового продукта на основе функциональности существующего.

Последнее замечание, по мнению автора, в сочетании с типичной функциональностью современных средств проектирования, поддерживающих анализ исходного кода (в случае объектно-ориентированных систем) и его визуализацию (в том числе, поведенческую, например, в виде диаграмм UML Sequence), позволяет объединить упомянутые категории инструментов в единый класс “инструментов реинжиниринга”. В то же время, деятельность по сопровождению и поддержке, в частности, касающаяся сбоев и исправления обнаруженных ошибок в программном обеспечении, требует, в определенной степени, отнесения к этой теме и средств конфигурационного управления, рассматриваемых ниже (например, в части обработки запросов на изменения).

1.6 Инструменты конфигурационного управления (Software Configuration Management Tools)

Инструменты конфигурационного управления делятся на три категории:

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

- Инструменты отслеживания (tracking) дефектов, расширений и проблем.

- Инструменты управления версиями.

- Инструменты сборки и выпуска. Эти инструменты предназначены для управления задачами сборки и выпуска продуктов, а также включают средства инсталляции.

Дополнительная информация по данной теме представлена в области знаний SWEBOK “Конфигурационное управление”.

1.7 Инструменты управления инженерной деятельностью (Software Engineering Management Tools)

Средства управления деятельностью по программной инженерии делятся на три категории:

- Инструменты планирования и отслеживания проектов. Эти средства используются календарного планирования работ, количественной оценки усилий и стоимостных ожиданий, связанных с проектами.

- Инструменты управления рисками. Эти средства используются для идентификации, оценки ожиданий и мониторинга рисков.

- Инструменты количественной оценки. Эти инструменты ведения измерений помогают в выполнении работ, связанных с программой количественной оценки, проводимой в отношении проектов программного обеспечения.

Функциональные аспекты управления инженерной деятельностью достаточно детально представлены в области знаний SWEBOK “Управление программной инженерией” (Software Engineering Management).

1.8 Инструменты поддержки процессов (Software Engineering Process Tools)

В описании этой темы в текущей версии SWEBOK наблюдается противоречие между кратким делением на категории инструментов и их более детальным определением. Скорее всего, такая несогласованность связана, в первую очередь, с отсутствием достигнутого консенсуса в этой области. Базируясь на обеих классификациях, упомянутых в SWEBOK, автор хотел бы отметить несколько типов инструментов из “смежных” областей, имеющих особое значение в поддержке процессов программной инженерии:

- Инструменты моделирования, позволяющие, в частности, описать и модель процессов, как таковую.

- Инструменты управления проектами.

- Инструменты конфигурационного управления, поддерживающие работу с актуальными версиями всего комплекса артефактов проекта и, что не менее важно, позволяющие задать поведенческие характеристики (в упрощенном понимании - workflow) и атрибуты этих артефактов в форме элементов конфигураций.

- Ролевые платформы разработки программного обеспечения, охватывающие все стадии жизненного цикла и, на сегодняшний день, являющиеся развитием интегрированных средств разработки и CASE-инструментов в направлении поддержки “смежной” функциональности – управления требованиями, работ по конфигурационному управлению с поддержкой управления изменениями, тестирования и оценки качества.

Первые три вида инструментов в такой классификации позволяют описать применяемые процессы программной инженерии. Четвертый класс – “супер-интегрированные среды разработки”, называемые сегодня ролевыми платформами разработки, обеспечивают поддержку заданных процессов, описанных, например, в виде соответствующих правил на уровне глубоко интегрированных в такие среды инструментов конфигурационного управления.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

1.9 Инструменты обеспечения качества (Software Quality Tools)

Средства обеспечения качества делятся на две категории:

- Инструменты инспектирования. Эти средства используются для поддержки обзора (review) и аудита.

- Инструменты (статического) анализа. Эти средства используются для анализа программных артефактов, данных, потоков работ и зависимостей. Такие инструменты предназначены для проверки определенных свойств или артефактов, в целом, на соответствие <заданным характеристикам>.

1.10 Дополнительные аспекты инструментального обеспечения (Miscellaneous Tool Issues)

Эта тема охватывает вопросы, касающиеся всех классов инструментов. Создателями SWEBOK идентифицированы три категории таких аспектов:

- Техники интеграции инструментов. Эти техники важны для естественного использования сочетания различных инструментов. Типичные виды интеграции инструментов включают платформы, представление, процессы, данные и управление.

- Мета-инструменты. Такие средства генерируют другие инструменты. Например, классическим примером мета-инструмента является компилятор компиляторов.

- Оценка инструментов. Данная тема представляется достаточно важной в силу постоянной эволюции инструментов программной инженерии.

2. Методы программной инженерии (Software Engineering Methods)

Данная секция (подобласть) разделена на три темы: *эвристические методы* (heuristic methods), касающиеся неформализованных подходов; *формальные методы* (formal methods), обоснованные математически; *методы прототипирования* (prototyping methods), базирующиеся на различных формах прототипирования. Эти три темы не являются изолированными <друг от друга>, скорее они выделены исходя из их значимости и на основе определенных достаточно явных индивидуальных особенностей. Например, объектно-ориентированный подход может включать формальные техники и использовать прототипирование для проверки и аттестации. Так же как и инструменты, методы программной инженерии постоянно эволюционируют. Именно поэтому, в описании данной области знаний авторы SWEBOK постарались избежать, насколько это возможно, упоминания любых конкретных методологий.

2.1 Эвристические методы (Heuristic Methods)

Эта тема содержит четыре категории методов: *структурные*, *ориентированные на данные*, *объектно-ориентированные* и *ориентированные на область применения*.

- Структурные методы (structured methods). При таком подходе системы строится с функциональной точки зрения, начиная с высокоуровневого понимания поведения системы с постепенным уточнением низко-уровневых деталей. (такой подход, иногда, также называют “проектированием сверху-вниз”, *прим. автора*)

- Методы, ориентированные на данные (data-oriented methods). Отправной точкой такого подхода являются структуры данных, которыми манипулирует создаваемое программное обеспечение. Функции в этом случае являются вторичными.

- Объектно-ориентированные методы (object-oriented methods). Система при таком подходе рассматривается как коллекция объектов, а не функций.

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

- Методы, ориентированные на конкретную область применения (domain-specific methods). Такие специализированные методы разрабатываются с учетом специфики решаемых задач, например, систем реального времени, безопасности <жизнедеятельности> (safety) и защищенности <от несанкционированного доступа> (security).

2.2 Формальные методы (Formal Methods)

Эта тема касается математических (строгих, *прим. автора*) методов программной инженерии.

К сожалению, по мнению автора, SWEBOK не дает здесь какого-либо определения формальных методов, поэтому, хотелось бы привести в данном контексте характеристику, данную им одним из классиков программной инженерии – Ианом Соммервиллем [Соммервилл, 2002, стр. 188]: “Термин *формальные методы* подразумевает ряд операций, в состав которых входит создание формальной спецификации системы, анализ и доказательство спецификаций, реализация системы на основе преобразования формальной спецификации в программы и верификация программ. Все эти действия зависят от формальной спецификации программного обеспечения. Формальная спецификация – это системная спецификация, записанная на языке, словарь, синтаксис и семантика которого определены формально. Необходимость формального определения языка предполагает, что этот язык основывается на математических концепциях. Здесь используется область математики, которая называется дискретной математикой и основывается на алгебре, теории множеств и алгебре логики.”

Эти методы можно классифицировать в виде следующих категорий:

- Языки и нотации специфицирования (specification languages and notations). Языки спецификаций могут быть ориентированы на модель, свойства и поведение. По мнению автора, ярким примером такого рода методов являются формальные методы описания требований, интерес к которым периодически возникает на протяжении всей истории программной инженерии.

- Уточнение (refinement). Данные подходы связаны с уточнением (трансформацией) превращения спецификаций в конечный результат, максимально близкий желаемому. В качестве результата применения таких методов рассматривается конечный - исполнимый программный продукт.

- Подтверждение/доказательство (verification/proving properties). Этот подход основывается на строгом доказательстве точности <любых> характеристик <исходных предпосылок и получаемого продукта> с использованием теорем и проверки точности моделей.

По мнению автора, история программной инженерии показала, что в области разработки прикладных систем, обоснованность (в частности, в силу трудоемкости) применения формальных методов не подтверждается на практике, за исключением случаев “скрытого” (неявного для разработчиков) применения определенных формальных методов на уровне внутренней реализации конкретных инструментов программной инженерии, например, в средствах моделирования и проектирования. Иан Соммервилл дает такую характеристику формальным методам [Соммервилл, 2002, стр. 188]: “Традиционные технические дисциплины … обычно легко адаптируют математические методы. Однако инженерия программного обеспечения не идет таким путем. Хотя прошло более 25 лет исследований по использованию математических методов в процессе создания ПО, воздействие этих методов все же ограничено. Так называемые формальны методы … широко не используются. Многие компании, разрабатывающие ПО, не считают экономически выгодным применение этих методов в процессе разработки.”

2.3 Методы прототипирования (Prototyping Methods)

Эта тема охватывает методы, основанные на прототипировании программного обеспечения. Они разделены на три категории:

- Стили прототипирования. Идентифицированы следующие подходы, касающиеся стилей прототипирования – создание временно используемых прототипов (throwaway), эволюционное прототипирование (в подавляющем большинстве случаев предполагает

mailto:sorlik@borland.ru
http://sorlik.blogspot.com

превращение прототипа в конечный продукт, *прим. автора*) и разработка исполняемых спецификаций (часто основывается как на формальных методах, *прим. автора*).

- Цели прототипирования. Примерами таких целей служат требования, архитектурный дизайн или пользовательский интерфейс.

- Техники оценки/исследования (evaluation) <результатов> прототипирования. Эти аспекты касаются того, как именно будут использованы результаты создания прототипа (например, будет ли он трансформирован в продукт, создается он для оценки нагрузочных способностей и других аспектов масштабируемости и т.п., *прим. автора*).

mailto:sorlik@borland.ru
http://sorlik.blogspot.com