# Программная инженерия

## Конструирование программного обеспечения (Software Construction)

Глава базируется на IEEE Guide to the Software Engineering Body of Knowledge - SWEBOK®, 2004. Содержит перевод описания области знаний SWEBOK® “Software Construction”, с комментариями и замечаниями.



Русский перевод SWEBOK 2004 с замечаниями и комментариями подготовлены Сергеем Орликом при участии Юрия Булуя. Дополнительные главы написаны Сергеем Орликом. Текст расширений SWEBOK отмечен ццветом, отличным от перевода оригинального текста.



**Официальный сайт “Основ программной инженерии” (по SWEBOK) - http://swebok.sorlik.ru**

# Программная инженерия

## Конструирование программного обеспечения (Software Construction)



Термин конструирование программного обеспечения (software construction) описывает детальное создание рабочей программной системы посредством комбинации кодирования, верификации (проверки), модульного тестирования (unit testing), интеграционного тестирования и отладки.

Данная область знаний связана с другими областями. Наиболее сильная связь существует с проектированием (Software Design) и тестированием (Software Testing). Причиной этого является то, что сам по себе процесс конструирования программного обеспечения затрагивает важные аспекты деятельности по проектированию и тестированию. Кроме того, конструирование отталкивается от результатов проектирования, а тестирование (в любой своей форме) предполагает работу с результатами конструирования. Достаточно сложно определить границы между проектированием, конструированием и тестированием, так как все они связаны в единый комплекс процессов жизненного цикла и, в зависимости от выбранной модели жизненного цикла и применяемых методов (методологии), такое разделение может выглядеть по разному.

Хотя ряд операций по проектированию детального дизайна может происходить до стадии конструирования, большой объем такого рода проектных работ происходит параллельно с конструированием или как его часть. Это есть суть связи с областью знаний “Проектирование программного обеспечения”.

В свою очередь, на протяжении всей деятельности по конструированию, инженеры используют модульное и интеграционное тестирование. Таким образом связана данная область знаний с “Тестированием программного обеспечения”.

В процессе конструирования обычно создается большая часть активов программного проекта - конфигурационных элементов (configuration items). Поэтому в реальных проектах просто невозможно рассматривать деятельность по конструированию в отрыве от области знаний “Конфигурационного управления” (Software Configuration Management).

Так как конструирование невозможно без использования соответствующего инструментария и, вероятно, данная деятельность является наиболее инструментально-насыщенной, важную роль в конструировании играет область знаний “Инструменты и методы программной инженерии” (Software Engineering Tools and Methods).

Безусловно, вопросы обеспечения качества значимы для всех областей знаний и этапов жизненного цикла. В то же время, код является основным результирующим элементом программного проекта. Таким образом, явно напрашивается и присутствует связь обсуждаемых вопросов с областью знаний “Качество программного обеспечения” (Software Quality).

Из связанных дисциплин программной инженерии (Related Disciplines of Software Engineering) наиболее тесная и естественная связь данной области знаний существует с компьютерными науками (computer scince). Именно в них, обычно, рассматриваются вопросы построения и использования алгоритмов и практик кодирования. Наконец, конструирование касается и управления проектами (project management), причем, в той степени, насколько деятельность по управлению конструированием важна для достижения результатов конструирования.

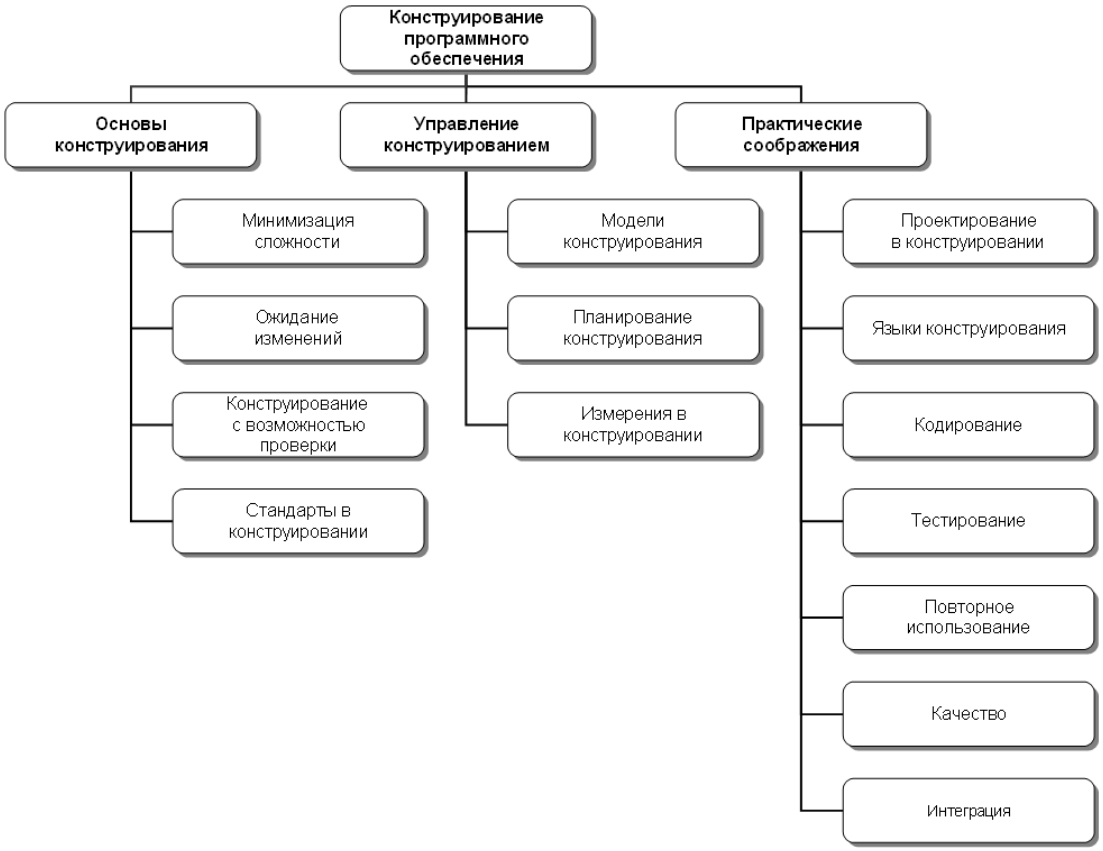


Рисунок 4. Область знаний “Конструирование программного обеспечения” [SWEBOK, 2004, с.4-2, рис. 1]

## 1. Основы конструирования (Software Construction Fundamentals)

Фундаментальные основы конструирования программного обеспечения включают:

- Минимизация сложности
- Ожидание изменений
- Конструирование с возможностью проверки
- Стандарты в конструировании

Первые три концепции применяются не только к конструированию, но и проектированию, и лежат в основе современных методологий управления жизненным циклом программных систем.

### *1.1 Минимизация сложности (Minimizing Complexity)*

Основной причиной того, почему люди используют компьютеры в бизнес-целях, являются ограниченные возможности людей в обработке и хранении сложных структур и больших объемов информации, в частности, на протяжении длительного периода времени. Это соображение является одной из основных движущих сил в конструировании программного обеспечения: минимизация

сложности. Потребность в уменьшении сложности влияет на все аспекты конструирования и особенно критична для процессов верификации (проверки) и тестирования результатов конструирования, т.е. самих программных систем.

Уменьшение сложности в конструировании программного обеспечения достигается при уделении особого внимания созданию простого и легко читаемого кода, пусть и в ущерб стремлению сделать его идеальным (например, с точки зрения гибкости или следования тем или иным представлениям о красоте, утончённости кода, ловкости тех или иных приемов, позволяющих его сократить в ущерб размерам и т.п.). Это не значит, что должно ущемляться применение тех или иных развитых языковых возможностей используемых средств программирования. Это подразумевает “лишь” придание большей значимости читаемости кода, простоте тестирования, приемлемому уровню производительности и удовлетворению заданных критериев, вместо постоянного совершенствования кода, не оглядываясь на сроки, функциональность и другие характеристики и ограничения проекта.

Минимизация сложности достигается, в частности, следованием стандартам (обсуждаются в теме 1.4 “Стандарты в конструировании”), использованием ряда специфических техник (освещаются в 3.3 “Кодирование”) и поддержкой практик, направленных на обеспечение качества в конструировании (3.5 “Качество конструирования”).

*1.2 Ожидание изменений (Anticipating Changes)*

Большинство программных систем изменяются с течением времени. Причин этому – множество. Ожидание изменений является одной из движущих сил конструирования программного обеспечения. Программное обеспечение не является изолированным от внешнего окружения (как системного, так и с точки зрения области деятельности, для автоматизации задач и проблем которого оно применяется). Более того, программные системы являются частью изменяющейся среды и должны меняться вместе с ней, а, иногда, и быть источником изменений самой среды.

Ожидание изменений поддерживается рядом техник, представленных в теме 3.3 “Кодирование”.

*1.3 Конструирование с возможностью проверки (Constructing for Verification)*

“Конструирование для проверки” (а именно такой смысл заложен в оригинальное название данной темы) предполагает, что построение программных систем должно вестись таким образом, чтобы сама программная система помогала вести поиск причин сбоев, будучи прозрачной для применения различных методов проверки (и, кстати, внесения необходимых изменений), как на стадии независимого тестирования (например, инженерами-тестировщиками), так и в процессе операционной деятельности - эксплуатации, когда особенно важна возможность быстрого обнаружения и исправления возникающих ошибок.

Среди техник, направленных на достижение такого результата конструирования:

- обзор, оценка кода (code review)
- модульное тестирование (unit-testing)
- структурирование кода для и совместно с применениям автоматизированных средств тестирования (automated testing)
- ограниченное применение сложных или тяжелых для понимания языковых структур

*1.4 Стандарты в конструировании (Standards in Constructing)*

Стандарты, которые напрямую применяются при конструировании, включают:

- коммуникационные методы (например, стандарты форматов документов и <оформления> содержания)
- языки программирования и соответствующие стили кодирования (например, Java Language Specification, являющийся частью стандартной документации JDK – Java Development Kit и Java Style Guide, предлагающий общий стиль кодирования для языка программирования Java)
- платформы (например, стандарты программных интерфейсов для вызовов функций операционной среды, такие как прикладные программные интерфейсы платформы Windows -

Win32 API, Application Programming Interface или .NET Framework SDK, Software Development Kit)

- инструменты (не в терминах сред разработки, но возможных средств конструирования – например, UML как один из стандартов для определения нотаций для диаграмм, представляющих структура кода и его элементов или некоторых аспектов поведения кода)

*Использование внешних стандартов*. Конструирование зависит от внешних стандартов, связанных с языками программирования, используемым инструментальным обеспечением, техническими интерфейсами и взаимным влиянием Конструирования программного обеспечения и других областей знаний программной инженерии (в том числе, связанных дисциплин, например, управления проектами). Стандарты создаются разными источниками, например, консорциумом OMG – Object Management Group (в частности. Стандарты CORBA, UML, MDA, …), международными организациями по стандартизации такими, как ISO/IEC, IEEE, TMF, …, производителями платформ, операционных сред и т.д. (например, Microsoft, Sun Microsystems, CISCO, NOKIA, …), производителями инструментов, систем управления базами данных ит.п. (Borland, IBM, Microsoft, Sun, Oracle, ...). Понимание этого факта позволяет определить достаточный и полный набор стандартов, применяемых в проектной команде или организации в целом.

*Использование внутренних стандартов*. Определенные стандарты, соглашения и процедуры могут быть также созданы внутри организации или даже проектной команды. Эти стандарты поддерживают координацию между определенными видами деятельности, группами операций, минимизируют сложность (в том числе при взаимодействии членов проектной группы и за ее пределами), могут быть связаны с вопросами ожидания и обработки изменений, рисков и вопросами конструирования для проверки и дальнейшего тестирования. В сочетании со внешними стандартами, внутренние стандарты призваны определить общие правила игры для всех членов проектной команды, договорившись о терминах, процедурах и других значимых соглашениях, вне зависимости от степени формализации процессов конструирования, в частности, и процессов жизненного цикла, в общем случае.

## 2. Управление конструированием (Managing Construction)

### *2.1 Модели конструирования (Construction Models)*

Модели конструирования определяют комплекс операций, включающих последовательность, результаты (например, исходный код и соответствующие unit-тесты) и другие аспекты, связанные с общим жизненным циклом разработки. В большинстве случаев, модели конструирования определяются используемым стандартом жизненного цикла, применяемыми методологиями и практиками. Некоторые стандарты жизненного цикла, по своей природе, являются ориентированными на конструирование – например, экстремальное программирование (XP- eXtreme Programming). Некоторые рассматривают конструирование в неразрывной связи с проектированием (в части моделирования), например, RUP (Rational Unified Process).

Создано множество моделей разработки программного обеспечения. Ряд из них в большей степени сфокусирован на конструировании программного обеспечения, как таковом.

Некоторые модели являются более линейными с точки зрения конструирования ПО. К ним относятся, например, водопадная (waterfall) и поэтапная (staged-delivery) модели жизненного цикла (моделям жизненного цикла посвящена специальная глава, написанная Сергеем Орликом как важное расширение SWEBOK). Эти модели рассматривают конструирование как деятельность, которая начинает проводиться только после завершения определенных обязательных к выполнению (prerequisite) работ, включающих детальное определение требований, подробный дизайн и детальное планирование. Более линейные подходы стараются подчеркнуть действия, предваряющие конструирование (т.е. требования и дизайн) и создать более четкое разделение между такими различными типами деятельности. В таких моделях основным содержанием конструирования может быть кодирование.

Другие модели более итеративны, к ним относятся – эволюционное прототипирование, экстремальное программирование и Scrum. Эти подходы сходятся к рассмотрению конструирования как деятельности, которая ведется одновременно с другими видами работ по созданию программного обеспечения и пересекаясь с ними (видимо, здесь имеется в виду взаимозависимость

и влияние друг на друга), включая определение требований, проектирование и планирование. Эти подходы смешивают проектирование, кодирование и тестирование, часто рассматривая их комбинацию как конструирование.

Соответственно, что именно подразумевается под “конструированием” зависит в определенной степени от используемой модели жизненного цикла.

*2.2 Планирование конструирования (Construction Planning)*

Выбор метода (методологии) конструирования является ключевым аспектом для планирования конструкторской деятельности. Такой выбор значим для всей конструкторской деятельности, а также необходимых условий её осуществления, определяя порядок соответствующих операций и уровень выполнения заданных условий перед тем как начнется конструирование или составляющие его действия. Например, модульное тестирование в ряде методов является частью работ, после написания соответствующего функционального кода, в то время, как ряд гибких (agile) практик, например, XP (кстати, первыми начавшие использовать такие методы верификации кода), требуют написания Unit-тестов до того, как пишется соответствующий код, требующий тестирования.

Используемый подход к конструированию влияет на возможность уменьшения (в идеале - минимизации) сложности, готовности к изменениям и конструировании с возможностью проверки.

Планирование конструкторской деятельности определяет порядок, в котором создаются компоненты и другие активы данной области знаний (фазы деятельности), проводятся работы по обеспечению качества получаемого программного обеспечения, распределяются* задачи и соответствующие ресурсы, в том числе, определяются назначения/отображения работ конкретным инженерам-программистам, членам проектной группы. Все это, конечно, происходит, следуя правилам, определяемым используемым методом (методологией, практиками и т.п.).

*Заметьте – не распределяют, а *распределяются*, подразумевая процесс, приводящий к обеспечению явной связи между задачей и ресурсами. В нечетко регламентированных (это ни в коем случае не ругательство, это определение – ведь существует же понятие нечёткая логика, неструктурированные базы данных, например, в отношении нереляционных систем и т.п.) и неформальных методах, таких, как XP, члены проектной группы сами принимают на себя ответственность по решению определенных задач, а “владение” кодом является совместным на основе сотрудничества, как одного из ключевых принципов работы проектной команды.

*2.3 Измерения в конструировании (Construction Measurement)*

Большая часть результатов, да и самой деятельности по конструированию программного обеспечения, может быть измерена, в том числе - количественно. Это и исходный разработанный код, и модифицированный объем кода, и степень повторного использования, и многие другие характеристики. Эти измерения, или как их еще принято называть – результаты *аудита кода* и *метрики кода*, несут большую пользу как для оценки рисков и качества (приводящих к соответствующим операциям по снижению рисков и повышению качества), а также, для управления конструированием и программными проектами, в целом. О каком планировании может идти речь, если мы не способны предсказать (например, на основе оценки результатов предыдущих проектов) ни длительность работ, ни стоимость отдельных задач, ни вероятность возникновения дефектов против заданных параметров приемлемого качества?

Код является одним из наиболее четко детерминированных активов проекта (постепенно такими становятся и модели, строящиеся на основе структур метаданных, и тесно связанные с кодом - например, UML). Код является и самим носителям требуемой функциональности. Соответственно, применение измерений в отношении кода становится тем инструментом, который влияет и на управление и на сам код.

Последнее время, большое внимание многие профессиональные разработчики, то есть инженеры-конструкторы программного обеспечения, уделяют *рефакторинг* кода, как методы его реструктурирования, призванные без изменения содержания (то есть функциональности и функциональной целостности) обеспечить решение задач минимизации сложности, готовности к изменениям (гибкости), прозрачности документирования и многих других актуальных аспектов

конструирования. Но, к сожалению, многие забывают о необходимости *мотивированности* изменений, даже на уровне рефакторинга. Применение измерений, в частности, метрик, позволяет определить необходимость внесения таких изменений, проведения рефакторинга. И не потому что “так, наверное, будет всё же лучше, красивше...”, а потому, что в иерархии наследования из 10 поколений классов – 9 являются абстрактными (“из любви к искусству”), а на 10-м (в силу превышений сроков проекта, после того, как долго и “в кайф” создавали архитектурно-красивый код) “повешено” 20 (!) бизнес-методов. Вот это – *действительно* обоснованная причина для рефакторинга, который, даже с применением самых совершенных инструментальных средств, вместе с обдумыванием необходимости рефакторинга, а потом, иногда, и борьбой с его последствиями из-за несогласованности членов команды (а это уже жизнь, даже в самом идеальном коллективе, где люди понимают друг друга с полуслова) часто является просто тратой времени. Если применяется рефакторинг, но не применяются метрики – в подавляющем большинстве случаев, это отрицательно влияет на проект. И таких примеров работ, требующих применения измерений, но, к несчастью, игнорирующих их, можно приводить достаточно долго. Вы, наверняка, сами можете рассказать на эту тему много примеров из жизни.

Почему “авторизованный перевод” этой темы SWEBOK оказался столь эмоционален? Практика автора показывает, что в погоне за “красотой”, разработчики слишком часто забывают о цели их работы и воспринимают деятельность по управлению проектами (не буду спорить, иногда лишь формальную, для “прикрытия” всего, что только можно …) со стороны менеджеров и, тем более, любой аудит – как однозначную помеху “полёту мысли”. Зря. Если все именно так обстоит в вашем случае – проект, с большой вероятностью, а иногда и просто, “если не случится чудо”, можно считать пусть и не провальным (“фуух... не закрыли....”), то, наверняка, выйдущим за рамки тех или иных ограничений, будь то сроки, деньги, качество или, наконец, время, которое разработчик мог провести с семьей. И не сидеть ночью, дописывая функциональность или отлавливая очередную ошибку за несколько дней до передачи проекта в эксплуатацию. Все эти вопросы преследуют одну единственную цель – *предсказуемость работ и результата*. И измерения – важный инструмент достижения этой цели.

Область знаний “Software Engineering Process” уделяет специальное внимание вопросам измерений при создании программного обеспечения.

## 3. Практические соображения (Practical Considerations)

Конструирование – деятельность, в рамках которой программное обеспечение приводится к соглашению с произвольными (иногда - хаотическими) ограничениями реального мира, которые требуют от программного обеспечения точного следования им. Приближаясь к ограничениям реального мира, конструирование (в большей степени, чем любая другая область знаний) ведется на основе практических соображений и техник.

### *3.1 Проектирование в конструировании (Construction Design)*

Некоторые проекты предполагают больший объем работ по проектированию именно на стадии конструирования; другие проекты явно выделяют проектную деятельность в форме фазы дизайна. Вне зависимости от четкости выделения деятельности по проектированию, как таковой, практически всегда на стадии конструирования приходится заниматься и вопросами детального дизайна системы. Такие проектные работы имеют стремление к следованию устойчивым ограничениям, навязываемым конкретными проблемами, решение которых должно быть обеспечено использованием конструируемой программной системы.

Детали деятельности по проектированию на стадии конструирования в основном те же самые, что и описанные в области знаний “Проектирование программного обеспечения” (Software Design). Отличие заключается в большем внимании деталям.

### *3.2 Языки конструирования (Construction Languages)*

Языки конструирования включают все формы коммуникаций, с помощью которых человек может задать решение проблемы, выполняемое на компьютере.

Простейший тип языков конструирования – *конфигурационный язык (configuration language)*, позволяющий задавать параметры выполнения программной системы.

*Инструментальный язык (toolkit language)* – язык конструирования из повторно-используемых элементов; обычно строится как сценарный язык (script), выполняемый в соответствующей среде.

*Язык программирования (programming language)* – наиболее гибкий тип языков конструирования. Содержит минимальный объем информации о конкретных областях приложения и процессе разработки, требуя больше всего (по сравнению с другими типами языков конструирования) усилий на изучение и наработку опыта для эффективного применения при решении конкретных задач.

Существует три основных вида нотаций используемых при определении языков программирования:

- лингвистическая
- формальная
- визуальная

*Лингвистические нотации* характеризуются, в частности, использованием строк текста, содержащих специализированные “слова”, представляющие сложные программные конструкции, и комбинируемые в шаблоны, напоминающие предложения, построенные в соответствии с определенным синтаксисом. В случае корректного использования таких нотаций, каждая получаемая строка обладает строгой смысловой нагрузкой (семантикой), обеспечивающей интуитивное понимание того, что будет происходить когда будет выполняться программное обеспечение, построенное с использованием такого языка конструирования.

*Формальные нотации* являются менее интуитивными, чем лингвистические, и часто базируются на точных формальных (математических) определениях. Формальные нотации конструкций и формальные методы являются ядром практически всех форм системного программирования, точнее – поведения систем во времени. Такие нотации обеспечивают наибольшую готовность получаемого кода к тестированию, что намного важнее, чем просто возможность отображения на обычный человеческий язык. Формальные конструкции также используют точный метод определения комбинаций применяемых символов, что позволяет избежать неоднозначностей, присущих конструкциям естественных языков.

*Визуальные нотации* наименее связаны с текстово-ориентированными подходами, предполагая непосредственную интерпретацию визуальных конструкций в процессе исполнения описываемой логики. При этом логика в визуальных нотациях задается расположением соответствующих визуальных сущностей, ответственных за те или иные операции и структуры данных.

Использование визуальных конструкций ограничено сложностью визуального представления сложных выражений и утверждений только за счет перемещения визуальных сущностей на диаграмме (визуальном представлении). Однако, визуальная нотация может играть роль достаточно мощного инструмента, когда применяется в тех задачах программирования, где необходимо построение пользовательского интерфейса для программ, чья логика. Детализированное поведение определено ранее.

Сегодняшние работы (и их состояние) в области архитектур и приложений, управляемых моделями, в первую очередь - OMG MDA (Model-Driven Architecture www.omg.org/mda)/UML (Unified Modeling Language www.omg.org/uml), Microsoft DSL (Domain-Specific Language), направлены на то, чтобы использовать ту или иную визуальную нотацию, базирующуюся на мета-моделях, в качестве инструмента, применяемого и для определения функциональной логики системы. Можно ли в чистом виде считать эти нотации именно визуальными - вопрос спорный. Но в терминах, предлагаемых SWEBOK, они относятся именно к визуальным нотациям, так как предполагают однозначную интерпретацию визуального представления в виде текста и наоборот. Кроме того, исторически эти нотации определялись изначально как нотации визуального представления функциональности и уже в дальнейшем эти визуальные представления были отражены на цровне соответствующих мета-моделей (хотя это в большей степени верно для UML, чем DSL, но DSL можно рассматривать и как аналог UML, предполагающий бОльшую свободу применений и интегрированность с конкретной платформой - Microsoft). Другая область стандартов, направленных на применение визуальных нотаций для описания функциональности – Business Process Management Notation (BPMN - www.omg.org/bpmn) и связанный с ней язык Business Process Execution Language, построенный на

базе XML. Таким образом, область обоснованного применения визуальных нотаций для конструирования программных систем качественно расшириться и, не исключено, мы станем свидетелями de-facto формирования новой категории нотаций, соглашений и смешанных типов языковых средств, предназначенных для конструирования программного обеспечения как естественного продолжения проектирования.

*3.3 Кодирование (Coding)*

Практика конструирования программного обеспечения показывает активное применение следующих соображений и техник:

- техники создания легко понимаемого исходного кода на основе использования соглашений об именовании, форматирования и структурирования кода;
- использование классов, перечисляемых типов, переменных, именованных констант и других выразительных сущностей;
- организация исходного текста (в выражения, шаблоны, классы, пакеты/модули и другие структуры)
- использование структур управления;
- обработка ошибочных условий и исключительных ситуаций
- предотвращение возможных брешей в безопасности (например, переполнение буфера или выход за пределы индекса в массиве)
- использование ресурсов на основе применения механизмов исключения (из рассмотрения) и порядка доступа к параллельно используемым ресурсам (например, на основе блокировки данных, использования потоков и их синхронизации и т.п.)
- документирование кода
- тонкая “настройка” кода
- рефакторинг (не упоминается SWEBOK)

*3.4 Тестирование в конструировании (Construction Testing)*

При конструировании используются две формы тестирования, проводимого инженерами, непосредственно создающими исходный код:

- модульное тестирование (unit testing)
- интеграционное тестирование (integration testing)

Главная цель тестирования в конструировании уменьшить временной разрыв между моментом проявления ошибок, имеющихся в коде, и моментом их обнаружения. Во многих случаях, тестирование в конструировании производится после того, как код написан. В ряде случаев, тесты (что отмечалось ранее, на примере XP) пишутся до того, как создается код.

Тестировании в конструировании обычно включает подмножество видов тестирования, описанных в области знаний “Тестирование программного обеспечения” (Software Testing). Например, тестирование в конструировании обычно не включает системного тестирования, нагрузочного тестирования, usability-тестирования (оценки прозрачности использования) и других видов тестовой деятельности.

IEEE опубликовал два стандарта, посвященных данной теме:

- IEEE Std 829-1998: “IEEE Standard for Software Test Documentation”
- IEEE Std 1008-1987: “IEEE Standard for Software Unit Testing”

Напрямую с данной темой связаны под-темы SWEBOK 2.1.1 “Unit Testing” и 2.1.2 “Integration Testing”.

*3.5 Повторное использование (Reuse)*

Во введении в стандарт IEEE Std. 1517-99 “IEEE Standard for Information Technology – Software Lifecycle Process – Reuse Processes” даётся следующее понимание повторному использованию в программном обеспечении: “Реализация повторного использования программного обеспечения подразумевает и влечёт за собой нечто большее, чем просто создание и использование библиотек активов. Оно требует формализации практики повторного использования на основе интеграции процессов и деятельности по повторному использованию в сам жизненный цикл программного обеспечения.” В то же время, повторное использование достаточно важно и непосредственно при

конструировании программных систем, что подчеркивается включением этой темы в обсуждаемую область знаний конструирования ПО.

Задачи, связанные с повторным использованием в процессе конструирования и тестирования, включают:

- выбор единиц (units), баз данных тестовых процедур, данных <полученных в результате> тестов и самих тестов, подлежащих повторному использованию
- оценку потенциала повторного использования кода и тестов
- отслеживание информации и создание отчетности по повторному использованию в новом коде, тестовых процедурах и данных, полученных в результате тестов.

*3.6 Качество конструирования (Construction Quality)*

Существует ряд техник, предназначенных для обеспечения качества кода, выполняемых по мере его конструирования. Основные техники обеспечения качества, используемые в процессе конструирования, включают:

- модульное (unit) и интеграционное (integration) тестирование
- разработка с первичностью тестов (test-first development - тесты пишутся до конструирования кода)
- пошаговое кодирование (деятельность по конструированию кода разбивается на мелкие шаги, только после тестирования результатов которых производится переход к следующему шагу кодирования; известен также как итеративное кодирование с тестированием)
- использование процедур утверждений (assertion)
- отладка (в привычном понимании - debugging)
- технические обзоры и оценки (review)
- статический анализ

Выбор и использование конкретных техник часто диктуется стандартами (внутренними и внешними), используемыми проектной командой, а также зависят от опыта и подготовленности специалистов, занимающихся конструированием кода.

Деятельность по обеспечению качества в конструировании отличается от других операций по обеспечению качества. Основное отличие заключается в фокусе на программном (исходном) коде и других артефактах (активах), тесно связанных с кодом, в частности, детальных моделях.

*3.7 Интеграция (Integration)*

Одна из ключевых деятельностей, осуществляемых в процессе конструирования, - интеграция отдельно сконструированных операций (процедур), классов, компонентов и подсистем (модулей). В дополнение к этому, некоторые программные системы нуждаются в специальной интеграции с другим программным и аппаратным обеспечением.

Кроме упомянутых аспектов интеграции, к обсуждаемым интеграционным вопросам конструирования относятся:

- планирование последовательности, в которой интегрируются компоненты;
- обеспечение поддержки создания промежуточных версий программного обеспечения;
- задание “глубины” тестирования (в частности, на основе критериев “приемлемого” качества) и других работ по обеспечению качества интегрируемых в дальнейшем компонент;
- наконец, определение этапных точек проекта, когда будут тестироваться промежуточные версии конструируемой программной системы.