1.2. Программная инженерия - эссе на заданную тему

«Любое ремесло передается не учебниками, а подзатыльниками».
Народная мудрость

Начинать изложение любой дисциплины всегда затруднительно. Один вопрос тянет за собой остальные, термины и определения звучат как пустой звук, поскольку не наполнены содержанием. Поэтому лучше пренебречь строгостью изложения и попытаться рассмотреть предмет со всех сторон понемногу. В результате может сложиться нужная первоначальная картина.

Технология и ремесло

Программирование и программная инженерия – это ремесленный и технологический подходы в разработке ПО. А поскольку результатом является программный код, то в производстве программного кода. В программировании-ремесле важны инструменты, приемы, навыки и опыт, в программной инженерии-технологии – нормативы, организация, контроль процесса труда и качества продукта.

Программная инженерия – технологическая составляющая, программирование – ремесленная составляющая в производстве программного кода

В данном случае имеет место не противопоставление, а скорее взаимное дополнение деятельностей, в которых участвует программист. Все, что направлено на сам код, является элементами ремесла. Все, что направлено на взаимодействие с другими участниками процесса, управляющей средой – инженерия.

Баня, музыка, кино и мост. Если взять аналогию с музыкальным коллективом, то исполнительское мастерство каждого музыканта – ремесло, а организация репетиционной и концертной деятельности, сыгранность, взаимоотношения и все остальное, что имеет отношение к деятельности коллектива – инженерия.

В основе программной инженерии лежит промышленный подход к разработке ПО. Основные его составляющие:

- качество кода: инструменты разработки, тестирование, шаблоны проектирования;
- структура «производственного процесса»: методологии, стандарты, свод знаний о программной инженерии (SWEBOK);
- документирование, моделирование, описание артефактов (UML);
- управление процессом: методологии, фреймворки.

Для дальнейшего изложения необходимо различать близкие по смыслу термины:

- **программное обеспечение (ПО)** - набор компьютерных программ, процедур и связанной с ними документации и данных (ISO/IEC 12207);
- **программная система (ПС)** – программное обеспечение в сочетании с аппаратным, программным и пользовательским окружением;
- **программный продукт (ПП)** – программное обеспечение как товар и услуга, включающий в себя само ПО, средства и сервисы установки, обновления, сопровождения и поддержки среды функционирования. Различают «коробочные» и заказные ПП.

Еще одним важным понятием является **жизненный цикл (ЖЦ)** – период существования ПО (ПС). Различают:

- жизненный цикл разработки ПО – проектная деятельность по разработке и развертыванию ПС;
- жизненный цикл ПС – разработка, развертывание, поддержка и сопровождение.

Программная инженерия и информатика

В каких отношениях находится программная инженерия с информатикой? Здесь уместнее использовать термин computer science (буквальный перевод «компьютерные науки»), т.к. в обиходе информатика обычно ассоциируется с массовыми знаниями в области информационных технологий. Отношения достаточно специфические. Самое главное, что научные знания в этой области не содержат **фундаментальных** или **рамочных законов,** которые имеют место в естественных науках - фундаментальных и прикладных. Например, архитектура моста может быть какой угодно, но сопромат позволяет оценить запас прочности конструкции. В программной инженерии невозможно проверить проект «на прочность» или получить его фундаментальную характеристику типа «массы» до тех пор, пока он не будет введен в эксплуатацию. Отсутствие таких законов должно компенсироваться тестированием (валидацией, верификацией) проекта на всех этапах его разработки

> Компьютерные науки не содержат **рамочных законов,** позволяющих проверить существенные свойства программного кода.

Но не все так плохо в «Соединенном королевстве». Что касается отдельных компонент компьютерной архитектуры, организации вычислительных процессов и конкретных областей приложений, то из компьютерных наук вполне могут быть полезны:

- **теория информации;**
- **теория алгоритмов –** доказательство принципиальной невозможности существования алгоритмов и, соответственно, разработки программ для решения определенного рода задач *(алгоритмическая неразрешимость)*;
- **структуры данных и алгоритмы, дискретная математика, теория графов** – типовые способы организации данных, эффективные способы работы с ними, решение частных задач путем сведения их к стандартным математическим представлениям и решениям;
- **трудоемкость и эффективность алгоритмов** – оценка эффективности алгоритмов и программ «в перспективе» в зависимости от роста объемов входных данных, масштабирование;
- **математические основы предметной области** (например, обработка сигналов, криптография) – методы и алгоритмы решения задач в конкретной предметной области, наукоемкое ПО;
- **формальные языки и трансляторы –** эффективное использование языков и средств разработки на основе понимания принципов их работы: синтаксис и семантика формальных языков, формальные модели (автоматы, стековые автоматы), описания, «зашитые» в код и мета-системы.

Знания компьютерных наук обеспечивает качество ПО, но не гарантируют успех его разработки. Периодически на программистских форумах закипает дискуссия на тему

«нужны ли программисту фундаментальные знания компьютерных наук»? Сказанное выше является основанием для взвешенной позиции в этом вопросе. Вот навскидку набор популярных мнений:

- любое решение можно «нануглить» или сверстать из готовых компонент или готовых библиотек;
- фундаментальные или рамочные знания компьютерных наук (структуры данных, трудоемкость алгоритмов, базовые алгоритмы) не важны, если в совершенстве владеть инструментами и современными средствами разработки (в дискуссиях на эту тему особой нелюбовью почему-то пользуются красно-черные деревья);
- значительная часть задач, кодируемых программистами, является типовыми, и проявлять здесь креативность, как минимум, неуместно;
- «изобретать велосипед» в виде собственного решения вместо того, чтобы использовать опробованное известное – признак консерватизма и замшелости.

В то же время опыт разработки свидетельствует:

- любую программу с заданным функционалом можно реализовать так, что она будет работать сколь угодно медленно и использовать сколь угодно много памяти;
- «тупой» неэффективный, но надежный алгоритм, удовлетворяющий по производительности для текущих размерностей данных, лучше сложного, но требующего больших затрат в реализации и доказательств работоспособности;

Программная инженерия как проектная деятельность

Баня, музыка, кино и мост. К разработке программного проекта наиболее близка по духу проектная деятельность в искусстве или кино, например, съемка фильма или постановка спектакля. В меньшей степени сходство имеется с проектной деятельностью в материальном производстве. Тем не менее, к программной инженерии применимы принципы *управления проектами* (PM – Project Management) с учетом как общих сторон, так и специфики отрасли:

Общее в проектной деятельности всех видов:

- получение продукта с гарантированным функционалом, качеством и другими свойствами, с известными сроками исполнения и в рамках отработанного технологического процесса;
- общие принципы управления проектом (менеджмент), изложенные в своде знаний по управлению проектами (PMBOK) [6-9].

Специфика программной инженерии:

- основанием программной инженерии является информатика, а не естественные науки;
- основной упор делается на дискретной, а не на непрерывной математике;
- концентрация на абстрактных/логических объектах вместо конкретных/физических артефактов;
- отсутствие «производственной» фазы в традиционном промышленном смысле «проектирование – изготовление»;

- сопровождение программного обеспечения в основном связано с продолжающейся разработкой или эволюцией, а не с традиционным физическим износом;
- нематериальный характер продукта, нулевые затраты на тиражирование (аналогия с интеллектуальным продуктом);
- отсутствие аналогов изделия в окружающем мире. На ранних стадиях разработки – создание программных прототипов как «материальных аналогов» проекта;
- отсутствие «охватывающих» объективных законов и невозможность теоретического расчета и проверки. Работоспособность проекта проверяется тестированием готового продукта или верификацией артефактов разработки;
- затраты на непосредственное производство «материальных ценностей», т.е. конструирование кода составляют 20-25% стоимости проекта;
- «материальная основа проекта» - программный код является гибкой субстанцией, допускающей в любой момент внешнее качественное усовершенствование и коренную перестройку - *рефакторинг и реинжиниринг.*

Жизненный цикл программной системы и его модели

Жизненный цикл (ЖЦ) – период существования ПО (ПС), включающий в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающий жизнь системы от установления требований к ней до прекращения ее использования. **Модель жизненного цикла –** описание структуры и ключевых идей организации жизненного цикла. Модель ЖЦ определяет характерные черты, которые объединяют родственные методологии.

Каскадная модель ЖЦ

Каскадная модель предполагает однократное последовательное выполнение всех технологических процессов (рис.1-04). Принятое решение на определенном этапе не может быть отменено по результатам последующих этапов, все «непринципиальные» ошибки исправляются тестированием..

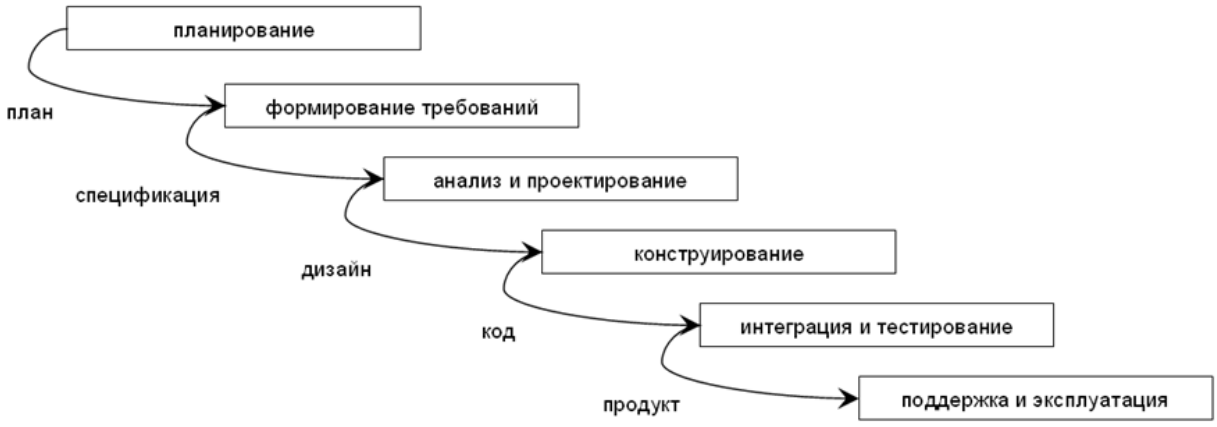

Рис.1-4. Каскадная модель ЖЦ

Несмотря на свою архаичность, каскадная модель в целом или на отдельных этапах применяется в крупных проектах, а также при передаче отдельных компонент на стороннее исполнение (аутсорсинг). Например, конструирование и тестирование (кодинг) отдельного приложения могут быть переданы после завершения этапа анализа и

проектирования системы на стороннюю разработку. Основанием для передачи и приемки является развернутое техническое задание, включающее артефакты предыдущих фаз: структуру базы данных, протоколы, спецификации графического интерфейса, сценарии, системные требования

Итеративная модель ЖЦ

Итерация – шаг разработки, завершающийся получением согласованных артефактов системы (программного прототипа, релиза, моделей и документов). Важным здесь является логическая завершенность и согласованность артефактов, создающих целостность – модель или работающий проект.

Инкремент - добавление функциональности, не сопровождаемое качественными изменениями свойств системы, чистая «дельта» функционала.

Итеративная модель предполагает, что разработка проекта идет в виде последовательности итераций. Итерация связана с понятием **эволюции** проекта. Под эволюцией понимается качественное изменение свойств проекта (модели или реализации). Итерация с точки зрения эволюции может быть двух видов:

- «чистый» инкремент: детализация требований, функциональности, архитектуры и их реализация в коде;
- эволюция требований, функциональности, архитектуры, которая сопровождается реинжинирингом кода и других артефактов, т.е. качественной перестройкой системы. При этом ревизия может касаться частностей (функциональности отдельных компонент) и общих принципов организации системы (вплоть до бизнес-требований и архитектуры).

Окончание итерации согласованными артефактами позволяет приступить к их тестированию и верификации. Это, в свою очередь, позволяет снизить неопределенность в основных характеристиках проекта (трудоемкость, сроки) и оценить возникающие риски (рис.1-5).

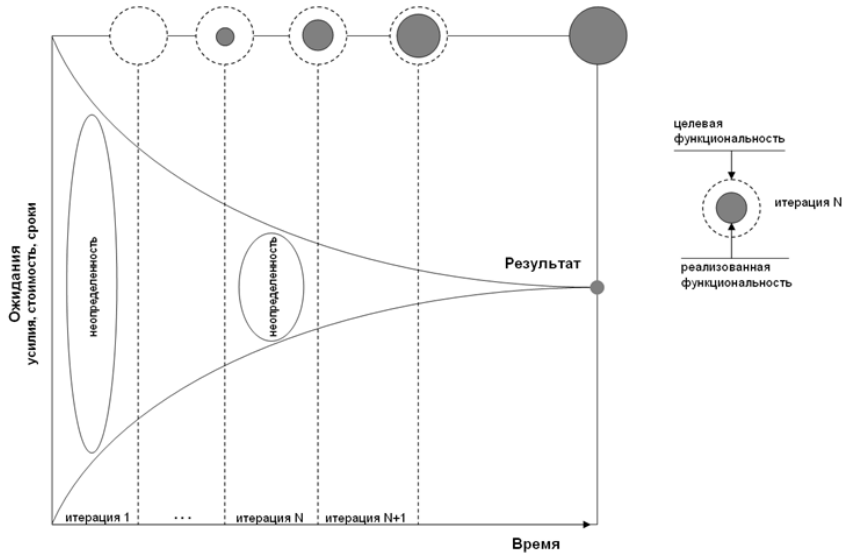

Рис.1-5. Снижение рисков и накопление функциональности в итерационной модели

Спиральная модель

Спиральная модель [1-6] является расширением итеративной: итерация является эволюционной и предполагает глубокую ревизию всех артефактов и свойств системы (рис.1-6).

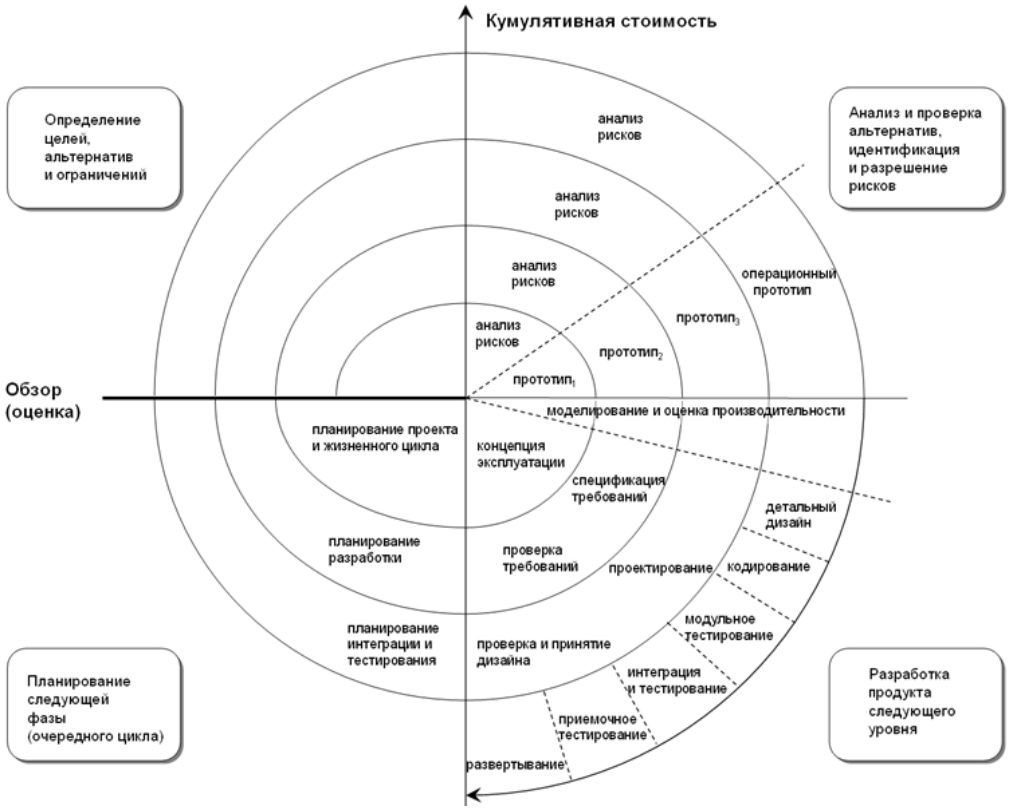

Рис.1-6. Спиральная модель ЖЦ

Итерация состоит в повторении фаз разработки «с нуля» на основе оценки результатов предыдущей итерации и рисков, таких как нереалистичные сроки и бюджет, реализация несоответствующей функциональности, разработка неправильного пользовательского интерфейса, недостаточная производительность получаемой системы и пр. [6-19].

Методологии программной инженерии

Модель жизненного цикла описывает только внешний вид жизненного цикла, не наполняя его деятельным содержанием, чем, в свою очередь, занимается методология. **Методология** (от «метод» и «логия» - учение о методах) – учение о структуре, логической организации, методах и средствах деятельности. Не стоит бояться «научности» этого слова.

Все методологии и связанные с ними артефакты: модели, стандарты, фреймворки можно позиционировать по двум качественным осям (рис.1-7):

- используемая модель жизненного цикла: каскадная – итерационная;
- степень тяжеловесности и формализации: легкие (неформальные) и тяжеловесные (формальные).

Вторая ось касается нескольких аспектов разработки:

- *степень охвата жизненного цикла*: «легковесные» и «тяжеловесные» процессы. Некоторые деятельности могут отсутствовать в методологии, если

они в малой степени присутствуют в проектах (например, в связи с масштабом проекта), пускаются на самотек или просто на оговариваются в методологии;

- *уровень формализации документов*: низкий – высокий. Используются канонические, стандартные формы артефактов (документов), либо их формат и содержание не оговаривается;
- *степень формализации взаимоотношений* исполнителей: низкая (гибкие) – высокая (традиционные). Практикуются формальные (документированные) взаимоотношения, либо стимулируются неформальные коммуникации и создание доверительной атмосферы.

Рис.1-7. Методологии программной инженерии и связанные с ней понятия

Коротко охарактеризуем артефакты, связанные с методологией:

- ГОСТы серии 19,34 – классическая водопадная модель;
- ГОСТ Р ИСО/МЭК 12207-99 - описание процессов ЖЦ программных средств (методология не оговаривается);
- структурные методы – наследие эпохи «структурного программирования», каскадный одномерный функциональный подход;
- унифицированный процесс (UP - Unified Process) и RUP (Rational Unified Process) - «тяжеловесная» итеративная, объектно-ориентированная методология, использующая UML, поддерживается фреймворком Rational Rose (см.5.1);
- Crystal Clear – легковесная гибкая методология, набор из 7 практик, ориентированных на коммуникации (а не на процессы) (рис.1-8);
- XP (Extreme Programming) - набор практик и рекомендаций в русле легковесного (гибкого) проектирования (см.5.2);

- Scrum - популярный фреймворк на основе гибкой итеративной методологии с довольно глубоким охватом процессов жизненного цикла и оригинальными решениями по планированию и управлению проектом (см.5.2);

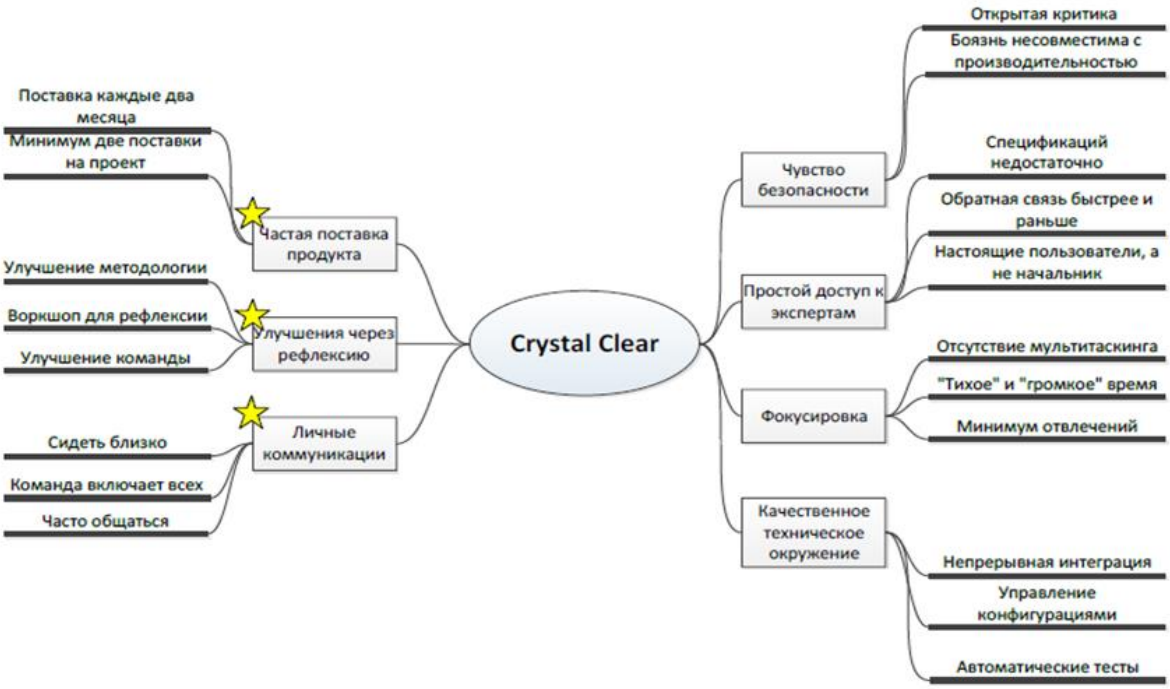

Рис.1-8. Набор практик Cristal Clear

- RAD - Rapid Application Development (быстрая разработка приложений) - разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементального прототипирования с применением инструментальных средств визуального моделирования и разработки. Большинство современных интегрированных сред разработки (IDE) по существу поддерживают RAD. «Предельный» случай RAD – верстка приложения из готовых компонент;
- AgileUP – авторская методология Скотта Амблера [5-3], попытка соединить строгость UP с гибким подходом к проектированию, снижение объемов «тяжеловесной» методологии, использование неканонических моделей и диаграмм (рис.1-9);

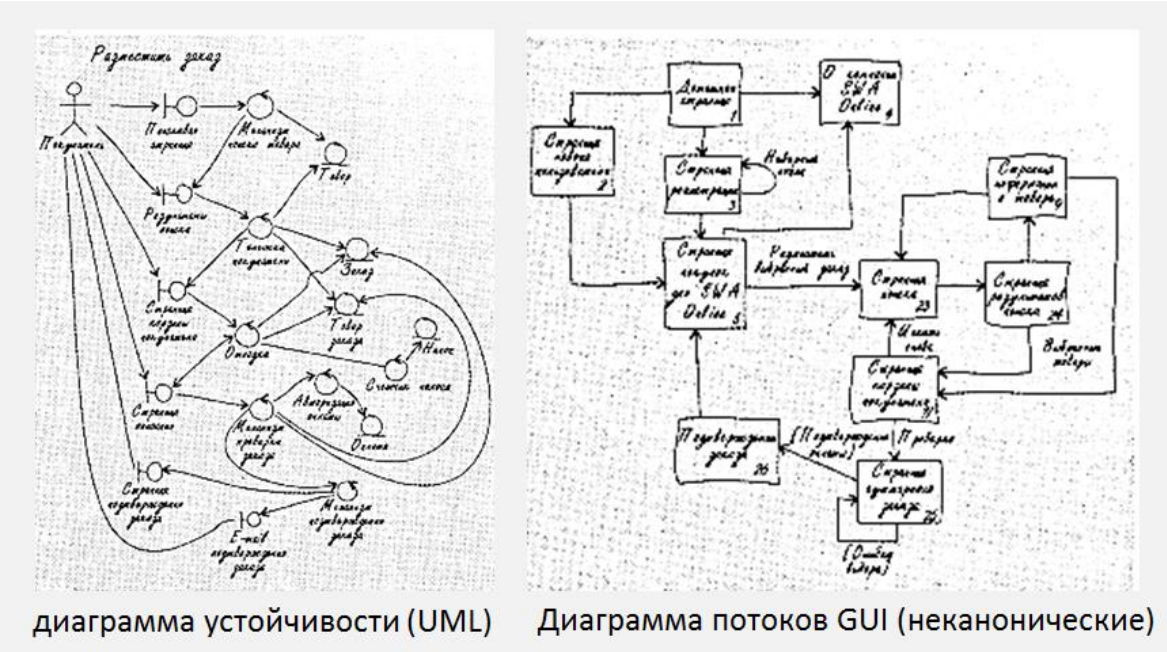

Рис.1-9. Диаграммы AgileUP

- ICONIX – облегченная версия UP с минимизацией видов моделей, артефактов и процессов (рис.1-10). Использует только 4 вида диаграмм UML: классов, устойчивости, вариантов использования и последовательности.

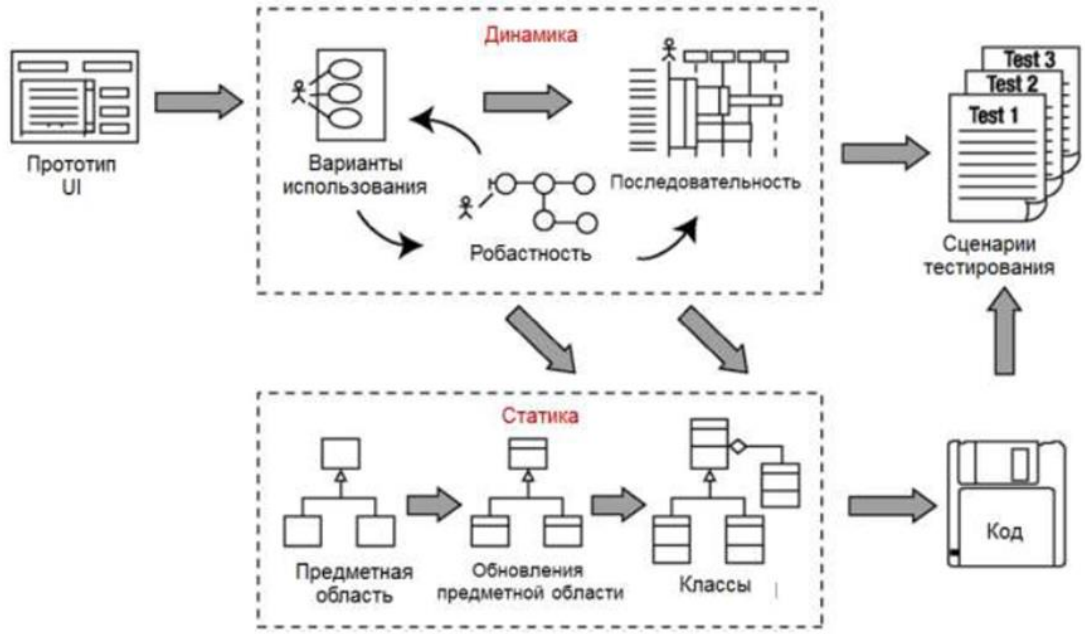

Рис.1-10. Артефакты ICONIX

Исторический экскурс: структурные модели проектирования ПО

Структурные модели представляю не только исторический интерес. Они показывают, как текущая парадигма программирования находит свое отражения во всех сопутствующих процессах и моделях проектирования. **Структурные модели** основаны на идеях технологии структурного программирования (модульность, иерархия, функциональная декомпозиция, проектирование «от функции к функции», структуры данных, нисходящее проектирование и тестирование). Они включают в себя:

- SADT (Structured Analysis and Design Technique) - модели и функциональные диаграммы;
- DFD (Data Flow Diagrams) - диаграммы потоков данных;
- ERD (Entity-Relationship Diagrams) - диаграммы «сущность-связь».

Недостатки структурных моделей следуют ограниченности функционального подхода. Это невозможность эволюции (развития), функциональная одномерность, отсутствие органической связи функциональность – данные.

Модель SADT

Базовый элемент модели - функциональный модуль (функция) – определяет процесс, в который включены данные, условия их обработки и требуемые ресурсы (рис.1-11). Соответственно, модуль-функция имеет управляющие связи четырех видов:

- вход - данные и объекты, потребляемые или изменяемые процессом;
- выход - основной результат процесса, конечный продукт;
- управление - условия, которыми регулируется процесс (стандарты, правила, время, бюджет);
- механизм – ресурсы и исполнители.

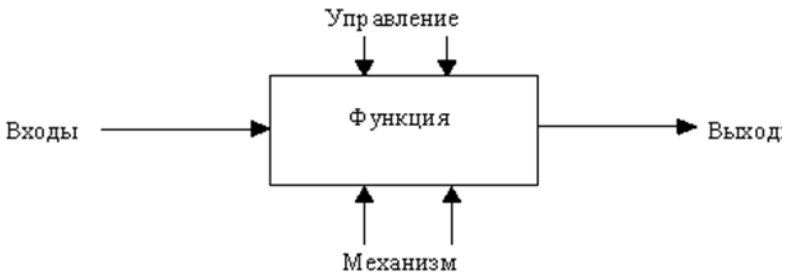

Рис.1-11. Базовый элемент SADT – функция.

Функциональные диаграммы полезны для описания бизнес-процессов, при использовании их в качестве спецификации программного кода можно провести параллели между условиями и потоком управления (вызовом функции), а также между механизмами и программным кодом со средой его исполнения (процессор/память).

Собранная из таких элементов модель позволяет отобразить структуру потоков данных и потоков управления и связать их в единое целое. На рис. 1-12 изображены тривиальные с точки зрения программиста варианты связывания модулей:

- коммуникативность по входу – набор данных с выхода модуля является входным для двух и более;
- последовательная связность – выход одного модуля является входом другого;
- процедурная связность – выходы двух модулей являются входами одного;
- функциональная связность – при исполнении кода одного модуля происходит вызов другого.

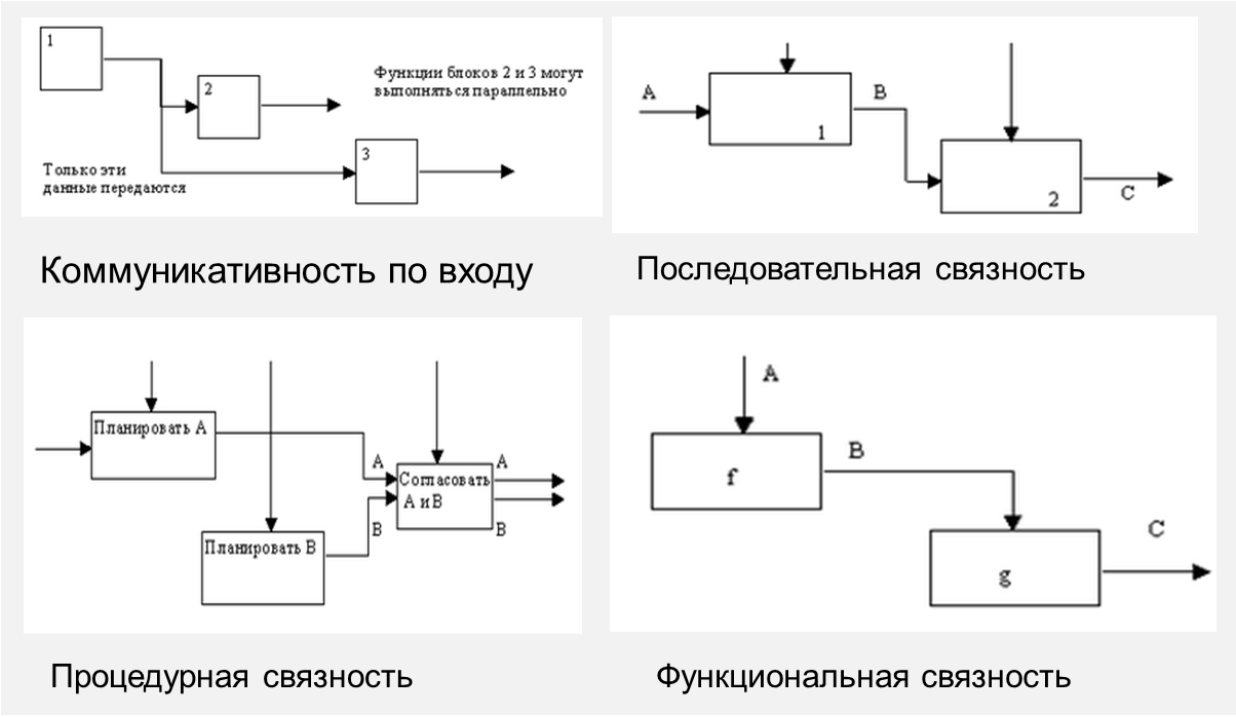

Рис.1-12. Примеры моделей SADT.

Модель потоков данных (DFD)

Основными элементами модели являются *внешние сущности, системы/подсистемы, процессы, накопители данных, потоки данных* (рис. 1-13). Как и в моделях потоков данных, используемых при описании вычислительных процессов, граф связей отражает не последовательность изменения состояний или действий «процессора», а перемещение объекта данных по технологической цепочке его накопления и обработки.

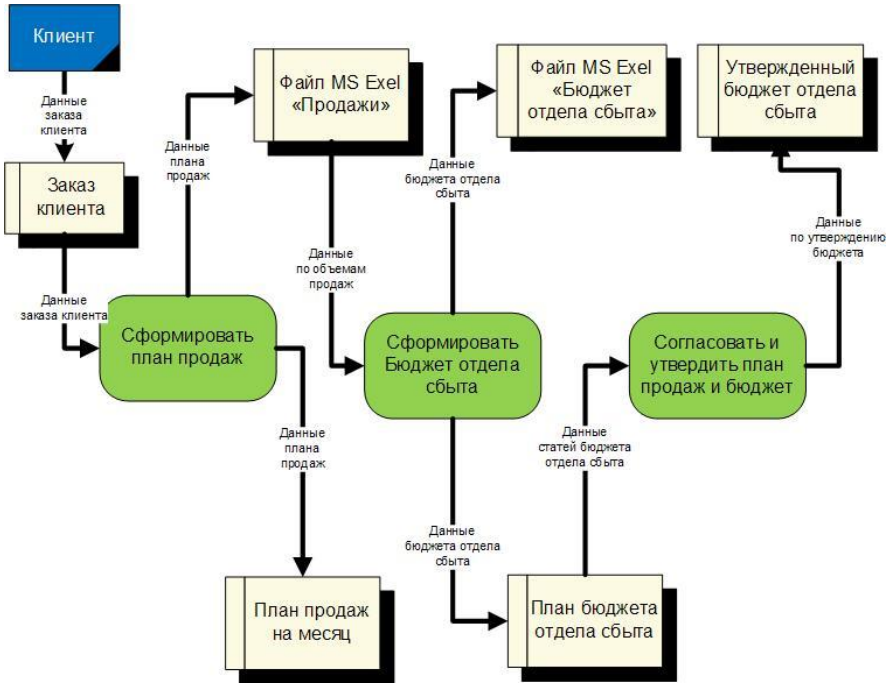

Рис.1-13. Пример модели DFD

Модели сущность-связь (ERD)

Всякий мало-мальски знакомый с сетевыми моделями баз данных, либо с диаграммами классов в UML легко узнает в этой модели первое и отдельные элементы второго (рис. 1-14).

Основные элементы (нотация) модели:

- сущность (класс, таблица);
- экземпляр сущности (объект, запись);
- атрибут (свойство объекта, поле записи): простой, составной, однозначный/многозначный, необязательны ,производный (вычисляемый);
- ключ - уникальный идентификатор объекта: простой / составной, первичный / альтернативный (несколько ключей), абсолютный (все атрибуты принадлежат сущности) / относительный (зависимая сущность);
- связи (отношения):
  - идентифицирующая (обязательная, родитель - потомок) / неидентифицирующая;
  - мощность связи (количествово экземпляров, соответствующих одному экземпляру на противоположном конце – 0..1,1,>0,*);
  - тип связи (1:1,1:N, N:1, N:N).

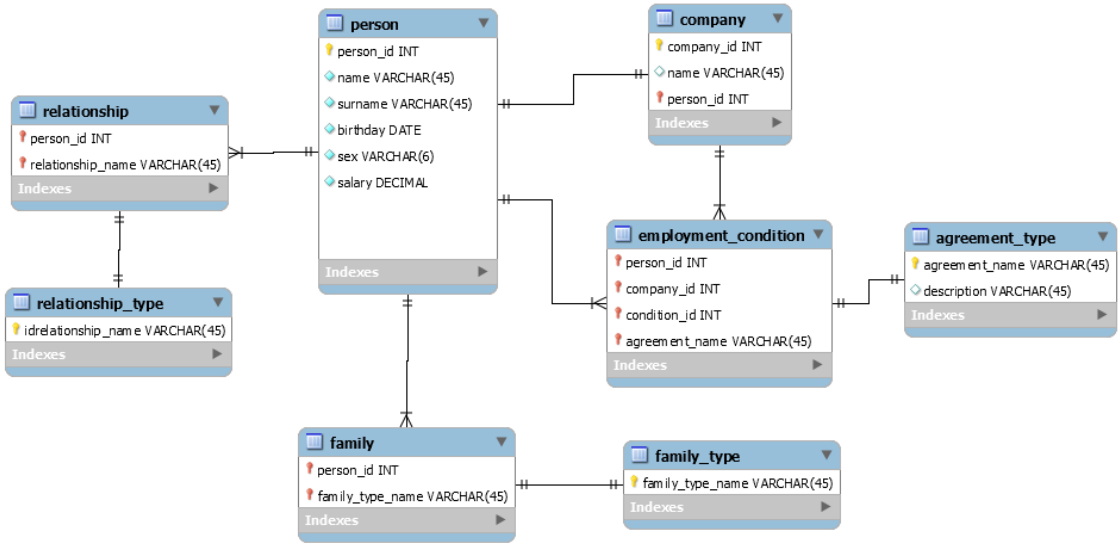

Рис.1-14. Пример модели ERD

Как любая модель, ориентированная на данные, она не отражает функциональность системы в целом и функционально-зависимые свойства связей.

Под сводом знаний SWEBOK

Основополагающим документом по программной инженерии явдяется «Свод знаний по программной инженерии» (SWEBOK - *Software Engineering Body of Knowledge*) [1-2], состоящий из 10 основных документов:

- ***Software requirements*** – программные требования;
- ***Software design*** – дизайн, проектирование (архитектура);
- ***Software construction*** – конструирование ПО;

- ***Software testing*** – тестирование;
- ***Software configuration management*** – конфигурационное управление;
- ***Software engineering management*** – управление в ПИ;
- *Software maintenance* – эксплуатация (поддержка) ПО;
- *Software engineering process* – процессы ПИ;
- *Software engineering tools and methods* – инструменты и методы;
- *Software quality* – качество ПО.

Большая часть документов свода знаний соответствуют основным технологическим процессам (дисциплинам) в разработке ПС. К сожалению, здесь применим афоризм Козьмы Пруткова: «Многие вещи нам непонятны не потому, что наши понятия слабы; но потому, что сии вещи не входят в круг наших понятий». Работа с этими документами имеет смысл только после погружения в соответствующий род деятельности, когда становится понятным содержательная сторона описанных в них предметов.