ПМ3
МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ПО

Лекция 1. ВВЕДЕНИЕ

Содержание курса. Примерный тематический план

Виды методологий программирования

Программная инженерия**,** её **с**вязь с информатикой, системотехникой и бизнес-реинжинирингом.

Определение программного обеспечения

Фаза ЖЦ ПО.

Критерии успешности проекта

Классификация, и характеристика методологий

Инженерия требований. Требования: классификация, выявление и анализ

Спецификация: классификация.
Шаблон спецификации требований (IEEE 830-1998):
общая часть (разделы 1 и 2), шаблоны раздела 3 SRS

ПРИМЕРНЫЙ ТЕМАТИЧЕСКИЙ ПЛАН (3 курс)

(ПМ3 СТАЦИОНАР: лекций – 16 часов, лабораторных– 16 часов, зачет)

1. Программная инженерия
2. Технологические подходы жизненного цикла ПО
3. Визуальное моделирование

1. Программная инженерия (КРАТКО)

1. Программная инженерия
2. Методологии проектирования ПО
3. Требования. (Свойства требований. Виды требований. Что не является требованием. Разработка требований. Спецификация требований…)
4. Информационная система

1. Программная инженерия

1. Чем программирование отличается от программной инженерии
2. Связь программной инженерии (как области практической деятельности) с информатикой, системотехникой и бизнес-реинжинирингом
3. Определение программного обеспечения, его свойства
4. Критерии успешности проекта. Классификация успешности проектов.
5. Отличие программной индустрии от других областей производства.
6. Чем определяются методологии проектирования ПО
7. Различные подходы к классификаций методологий проектирования ПО
8. Характеристика методологий: по стратегии конструирования ПО, по адаптивности процесса к окружению и полноте.
9. Перечислите отличия различные методологий проектирования ПО. Статистика использования методологий
10. Схема классификации стандартов в области информационных технологий
11. Инженерия требований. Требования. Свойства требований. Виды требований. Что не является требованием
12. Требования. Разработка требований. Спецификация требований.
13. Классификация спецификаций по уровню формализации, по способу представления
14. Характеристика процесса выявления требований. Способы выявления требований
15. Анализ требований. Приоритизация требований
16. Организация требований и способы их документирования
17. Типы документов, создаваемых в результате получения спецификации требований. Шаблоны спецификаций
18. Три уровня требований. Объективные противоречия между требованиями различных уровней
19. Организация требований и способы их документирования
20. V-модель проверки правильности требований. Основные принципы. Достоинства и ограничения
21. Стандарты, связанные с проверкой требований. Схема разработки требований.
22. Управление требованиями. Условия возможности изменений требований для разных стратегий
23. Политика управления изменениями требований
24. Управление версиями требований
25. Управление состояниями требований
26. Отслеживание состояний требований
27. Прослеживание требований (трассировка). Матрица прослеживания требований
28. Понятие информационной системы
29. "Портрет" современной информационной системы масштаба предприятия
30. "Три кита" информационной системы.

2. Технологические подходы жизненного цикла ПО (КРАТКО)

- Технологии создания программного обеспечения
- Введение в технологии программирования
- Основные группы технологических подходов
- Структурные технологии проектирования информационных систем
- Современные технологии объектно-ориентированного анализа и проектирования информационных систем

2. Технологические подходы жизненного цикла ПО

1. Понятия, используемые для представления жизненного цикла программы. Простейшее представление жизненного цикла программы. Основные группы технологических подходов жизненного цикла ПО. Классическая модель проектирования ПО: достоинства и недостатки
2. Каскадные технологические подходы жизненного цикла ПО
3. Каркасные подходы жизненного цикла ПО. Рациональный унифицированный процесс жизненного цикла ПО
4. Жизненный цикл проекта в RUP
5. Рабочий процесс RUP. Вспомогательные дисциплины RUP
6. Начальная стадия (Inception) RUP: назначение, цели, действия, артефакты
7. Уточнение (Elaboration) RUP: назначение, цели, действия, артефакты
8. Построение (Construction) RUP: назначение, цели, действия, артефакты
9. Внедрение (Transition) RUP: назначение, цели, действия, артефакты
10. Генетические подходы жизненного цикла ПО
11. Подходы на основе формальных преобразований жизненного цикла ПО
12. Гибкие (адаптивные, легкие) подходы жизненного цикла ПО. Манифест гибкой разработки ПО
13. Эволюционное прототипирование жизненного цикла ПО
14. Итеративная разработка жизненного цикла ПО
15. Различие между эволюционным и итеративным методами быстрой разработки
16. Экстремальное программирование (XP): принципы, характеристика
17. Адаптивная разработка жизненного цикла ПО. Scrum: артефакты, планирование, роли, методология
18. Постадийная (инкрементальная) разработка жизненного цикла ПО
19. Сравнительная характеристика технологических подходов к проектированию ПО
20. Сравнение методологий разработки ПО по отношению к каскадной или итеративной разработке и степени формальности
21. Модели зрелости процесса разработки (CMM, CMMI)
22. Методология объектно-ориентированного анализа и проектирования (ООАП)
23. Техники структурных диаграмм. Три базовые графические нотации структурного системного анализа для объектно-ориентированного анализа и проектирования (ООАП)
24. Диаграммы "сущность-связь"
25. Диаграммы функционального моделирования
26. Методология структурного анализа и проектирования
27. Модель IDEFo. Основные понятия:
28. Диаграммы потоков данных (Data Flow Diagrams).Графические примитивы.
29. Недостатки основных нотаций структурного системного анализа
30. Популярные графические нотации визуального моделирования конца 80-х годов

3. Визуальное моделирование (КРАТКО)

- Объектно-ориентированный язык моделирования
- Определение визуального моделирования
- Основные элементы языка UML: диаграммы
  - вариантов использования,
  - классов, кооперации,
  - последовательности,
  - состояний,
  - деятельности,
  - компонентов,
  - развертывания
- CASE-средства для построения диаграмм UML

3. Визуальное моделирование

1. Причины неудачности проектов разработки ПО. Характеристика лучших практик разработки ПО
2. Характеристика UML, как Унифицированного Языка Моделирования
3. История создания UML. Версии UML.
4. Характеристика UML, как языка спецификации, проектирования и документирования
5. Моделирование, визуальное моделирование. Основные понятия визуального моделирования
6. Поколения CASE-средств разработки визуальных моделей сложных систем
7. Объектно-ориентированное программирование (ООП) – основные понятия. Объектно-ориентированный анализ и проектирование (ООАП) – основные понятия.
8. Целесообразность использования моделей в виде диаграмм UML для различных проектов по технической сложности и сложности управления
9. Целесообразность использования моделей в виде диаграмм UML для различных проектов по типу приложений
10. Взаимосвязь нотации, методологии и инструментальных средств
11. UML, как унифицированный язык моделирования и чем он не является
12. Способы использования UML и назначение языка UML
13. Противоречивость и адекватность моделей в нотации UML
14. Канонические диаграммы языка UML 1.x и 2.x
15. Классификация моделей в языке UML
16. Статические и динамические модели языка UML
17. Рекомендации по изображению диаграмм в нотации языка UML
18. Механизмы расширения языка UML
19. Диаграмма вариантов использования (use case diagram). Графические примитивы. Типичные ошибки при разработке диаграмм вариантов использования
20. Формализация функциональных требований с помощью диаграммы ВИ. Классификация требований – модель FURPS+
21. Спецификация вариантов использования с помощью текстовых сценариев в UML
22. Шаблоны для написания сценария отдельного варианта использования
23. Принципиально разные случаи использования диаграммы UC и примеры
24. Диаграмма классов (class diagram). Графические примитивы. Назначение. Примеры.
25. Диаграмма объектов (object diagram). Графические примитивы. Назначение. Примеры.
26. Диаграмма последовательностей (sequence diagram). Графические примитивы. Назначение. Примеры.
27. Диаграмма взаимодействия (collaboration diagram). Графические примитивы. Назначение. Примеры.
28. Диаграмма состояний (statechart diagram). Графические примитивы. Назначение. Примеры.
29. Диаграмма активности или деятельностей (activity diagram). Графические примитивы. Назначение. Примеры.
30. Диаграмма развертывания (deployment diagram). Графические примитивы. Назначение. Примеры.

Методологии программирования

Методология программирования – это не способ того, как программируют, а это некий свод или совокупность идей, понятий, принципов, способов и средств, определяющая стиль написания, отладки и сопровождения программ. Она определяет то, как программист программирует, какие методы использует, что программирует, что делает и т.д.

Виды методологий программирования:

- Структурное программирование;
- Объектно-ориентированное программирование(ООП);
- Логическое программирование;
- Функциональное программирование;
- Программирование в ограничениях.

Структурное программирование – методология разработки ПО, в основе которой лежит представление программы в виде иерархической структуры блоков. Структурное программирование появилось из блок-схем. На основе блоков и работает структурное программирование. Любой классический язык (С, Фортран, Паскаль и др.) является структурным.

Объектно-ориентированное программирование (ООП) – методология, при которой основой составления программы являются объекты и классы. Неполный список объектно-ориентированных языков программирования: C# , C++, F#, Java, Delphi, Swift, Object Pascal, VB.NET, Visual DataFlex, Perl, PowerBuilder, Python, Scala, ActionScript (3.0), JavaScript, NET, Ruby, Smalltalk, Ada, Xbase++, X++, Vala, PHP, Cyclone.

Логическое программирование основано на автоматическом доказательстве теорем, а также раздел дискретной математики, изучающий принципы логического вывода информации на основе заданных фактов и правил вывода. Формально логическое программирование программированием не является, это некая математическая конструкция, предназначенная для автоматического доказательства теорем, это часть математической логики. Например, машина Тьюринга относится к логическому программированию.

Функциональное программирование – это программирование в функциях, или при помощи функций, как правило, математических. Классическим примером функционального программирования является работа в любом математическом паке, например, MathCAD, Mathematica, MatLab и Maple.

Программирование в ограничениях. Это методология программирования, когда программирование осуществляется в каких-то рамках, ограничениях, например, ограничение на максимальный размер итогового файла, по используемым функциям и т.д. Здесь отношения между переменными указаны в форме ограничений, ограничения отличаются от общих примитивов языков императивного программирования тем, что они определяют не последовательность шагов для исполнения, а свойства искомого решения. Формально скриптовое программирование является программированием в ограничениях, поскольку программист ограничен набором функций, который есть в скриптовом языке.

Методологии программирования

Наиболее популярны и известны 5 методологий программирования:

- ***Структурное программирование;***
- ***Объектно-ориентированное программирование(ООП);***
- Логическое программирование;
- Функциональное программирование;
- Программирование в ограничениях.

(Самые современные и мощные выделены).

Методологии характеризуются своим:

- философским подходом или основными принципами;
- согласованным множеством моделей методов, которые реализуют данную методологию;
- концепциями (понятиями), позволяющими более точно определить методы.

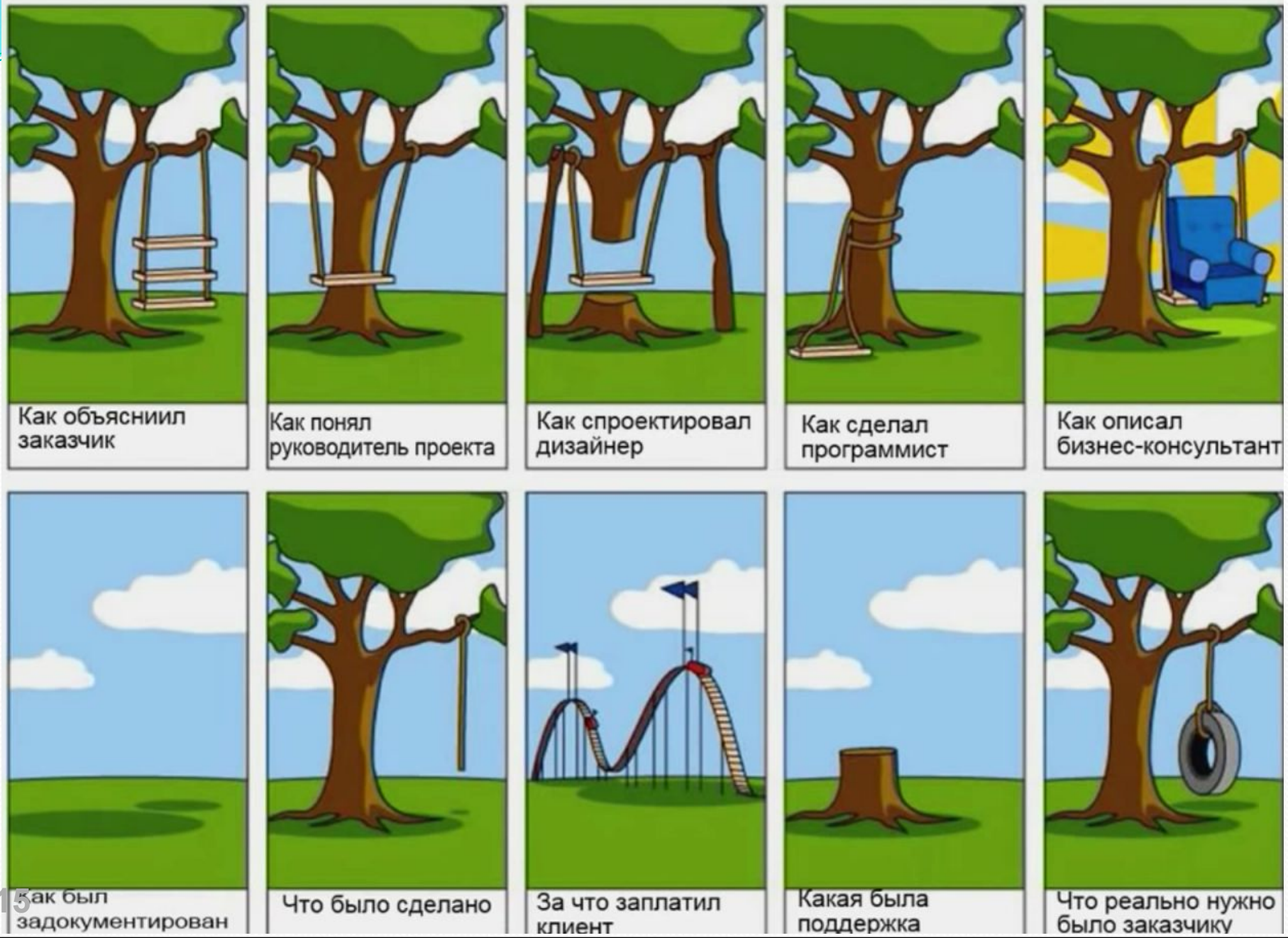
Основными самыми современными и мощными методологиями программирования являются структурное и ООП программирование.

Существуют еще другие методологии программирования, но упомянутые выше пять – наиболее популярные и известные.

Каждая методология характеризуется своим:

- философским подходом или основными принципами;
- согласованным множеством моделей методов, которые реализуют данную методологию;
- концепциями (понятиями), позволяющими более точно определить методы.

Типичный процесс создания продукта



Чем программирование отличается от программной инженерии?

- Программирование является некоторой абстрактной деятельностью и может происходить во многих различных контекстах. Можно программировать для удовольствия, для того, чтобы научиться (например, на уроках, на семинарах в университете), можно программировать в рамках научных разработок. А можно заниматься промышленным программированием.
- Как правило, это происходит в команде, и совершенно точно – для заказчика, который платит за работу деньги. При этом необходимо точно понимать, что нужно заказчику, выполнить работу в определенные сроки и результат должен быть нужного качества – того, которое удовлетворит заказчика и за которое он заплатит.
- Чтобы удовлетворить этим дополнительным требованиям, программирование "обрастает" различными дополнительными видами деятельности: разработкой требований, планированием, тестированием, конфигурационным управлением, проектным менеджментом, созданием различной документации (проектной, пользовательской и пр.).

Программная инженерия

- Разработка программного кода предваряется
  анализом (создание функциональной модели будущей системы без учета реализации, для осознания программистами требований и ожиданий заказчика);
  и *проектированием* (предварительный макет, эскиз, план системы на бумаге).
- Трудозатраты на анализ и проектирование, а также форма представления их результатов сильно варьируются от видов проектов и предпочтений разработчиков и заказчиков.
- Требуются также специальные усилия по организации процесса разработки. В общем виде это итеративно-инкрементальная модель, когда требуемая функциональность создается порциями, которые менеджеры и заказчик могут оценить, и тем самым есть возможность управления ходом разработки.
- Однако эта общая модель имеет множество модификаций и вариантов.

Программная инженерия

- Разработку системы необходимо выполнять с учетом удобств ее дальнейшего сопровождения, повторного использования и интеграции с другими системами.
- Это значит, что система разбивается на компоненты, удобные в разработке, годные для повторного использования и интеграции, а также имеющие необходимые характеристики по быстродействию.
- Для этих компонент тщательно прорабатываются интерфейсы. Сама система документируется на многих уровнях, создаются правила оформления программного кода – т. е. оставляются многочисленные семантические следы, помогающие создать и сохранить, поддерживать единую, стройную архитектуру, единообразный стиль, порядок…

Программная инженерия

- Все эти и другие дополнительные виды деятельности, выполняемые в процессе промышленного программирования и необходимые для успешного выполнения заказов называют **программной инженерией** (software engineering).
- Так мы обозначаем, во-первых, некоторую практическую деятельность, а во-вторых, специальную **область знания**. Для облегчения выполнения каждого отдельного проекта и возможности использовать разнообразный положительный опыт, достигнутый другими командами и разработчиками, этот опыт подвергается осмыслению, обобщению и надлежащему оформлению.
- Так появляются:

методы и практики (best practices) – тестирования, проектирования, работы над требованиями и пр., архитектурных шаблонов и пр.

стандарты и методологии, касающиеся всего процесса в целом (например, MSF, RUP, CMMI, Scrum).

Эти обобщения и входят в программную инженерию, как в область знания.

Необходимость в программной инженерии

- Необходимость в программной инженерии как в специальной области знаний была осознана мировым сообществом в конце 60-х годов прошлого века, более чем на 20 лет позже рождения самого программирования (если считать таковым знаменитый отчет фон Неймана "First Draft of a Report on the EDVAC", обнародованный им в 1945 году). Рождением программной инженерии является 1968 год – конференция NATO Software Engineering, г. Гармиш (ФРГ), которая целиком была посвящена рассмотрению этих вопросов.
- В сферу программной инженерии попадают все вопросы и темы, связанные с организацией и улучшением процесса разработки ПО, управлением коллективом разработчиков, разработкой и внедрением программных средств поддержки жизненного цикла разработки ПО.
- Программная инженерия использует достижения **информатики**, тесно связана с **системотехникой**, часто предваряется **бизнес-реинжинирингом**.

Информатика (computer science)

- это свод теоретических наук, основанных на математике и посвященных формальным основам вычислимости. Сюда относят математическую логику, теорию грамматик, методы построения компиляторов, математические формальные методы, используемые в верификации и модельном тестировании и т.д.

Трудно строго отделить программную инженерию от информатики, но в целом направленность этих дисциплин различна.

Программная инженерия нацелена на решение проблем производства, информатика – на разработку формальных, математизированных подходов к программированию.

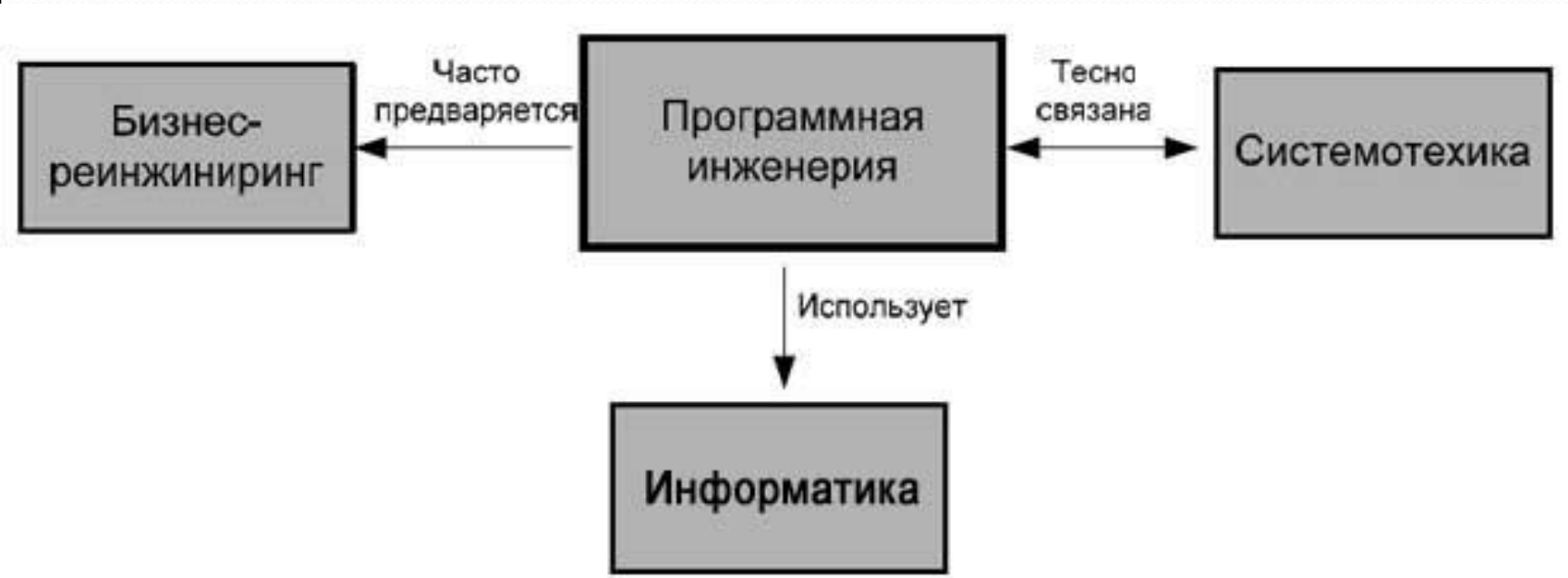
Системотехника (system engineering)

объединяет различные инженерные дисциплины по разработке всевозможных искусственных систем – энергоустановок, телекоммуникационных систем, встроенных систем реального времени и т.д. Очень часто ПО оказывается частью таких систем, выполняя задачу управления соответствующего оборудования. Такие системы называются *программно-аппаратными*, и участвуя в их создании, программисты вынуждены глубоко разбираться в особенностях соответствующей аппаратуры.

Бизнес-реинжиниринг (business reengineering)

в широком смысле обозначает модернизацию бизнеса в определенной компании, внедрение новых практик, поддерживаемых соответствующими новыми информационными системами. При этом акцент может быть как на внутреннем переустройстве компании так и на разработке нового клиентского сервиса (как правило, эти вопросы взаимосвязаны). Бизнес-реинжиниринг часто предваряет разработку и внедрение информационных систем на предприятии, так как требуется сначала навести определенный порядок в делопроизводстве, а лишь потом закрепить его информационной системой.

Связь программной инженерии (как области практической деятельности) с информатикой, системотехникой и бизнес-реинжинирингом



Определения

- Программное обеспечение
- Проектирование ПО
- Фаза проектирования ПО
- Жизненный цикл ПО
- Программный продукт

Программное обеспечение

Будем понимать под **программным обеспечением** (ПО) множество развивающихся во времени логических предписаний, с помощью которых некоторый коллектив людей управляет и использует многопроцессорную и распределенную систему вычислительных устройств.

(Определение Харальда Миллса - известного специалиста в области программной инженерии из компании IBM)

Комментарий к определению программного обеспечения

- Логические предписания – это не только сами программы, но и различная документация (например, по эксплуатации программ) и шире – определенная система отношений между людьми, использующими эти программы в рамках некоторого процесса деятельности.
- Современное ПО предназначено, как правило, для одновременной работы со многими пользователями, которые могут быть значительно удалены друг от друга в физическом пространстве. Таким образом, вычислительная среда (персональные компьютеры, сервера и т.д.), в которой ПО функционирует, оказывается распределенной.
- Задачи решаемые современным ПО, часто требуют различных вычислительных ресурсов в силу различной специализации этих задач, из-за большого объема выполняемой работы, а также из соображений безопасности. Например, появляется сервер базы данных, сервер приложений и пр. Таким образом, вычислительная среда, в которой ПО функционирует, оказывается многопроцессорной.
- ПО развивается во времени – исправляются ошибки, добавляются новые функции, выпускаются новые версии, меняется его аппаратная база.

Свойства ПО

- ПО является сложной динамической системой, включающей в себя технические, психологические и социальные аспекты.

Свойства ПО (сложность)

- Сложность программных объектов существенно зависит от их размеров. Как правило, бОльшее ПО (бОльшее количество пользователей, больший объем обрабатываемых данных, более жесткие требования по быстродействию и пр.) с аналогичной функциональностью – это другое ПО. Классическая наука строила простые модели сложных явлений, и это удавалось, так как сложность не была характеристической чертой рассматриваемых явлений.

(Сравнение программирования именно с наукой, а не с театром, кинематографом, спортом и другими областями человеческой деятельности, оправдано, поскольку оно возникло, главным образом, из математики, а первые его плоды – программы – предназначались для использования при научных расчетах. Кроме того, большинство программистов имеют естественнонаучное, математическое или техническое образование. Таким образом, парадигмы научного мышления широко используются при программировании – явно или неявно.)

Свойства ПО (согласованность)

ПО основывается не на объективных посылках (подобно тому, как различные системы в классической науке основываются на постулатах и аксиомах), а должно быть согласовано с большим количеством интерфейсов, с которыми впоследствии оно должно взаимодействовать. Эти интерфейсы плохо поддаются стандартизации, поскольку основываются на многочисленных и плохо формализуемых человеческих соглашениях.

Свойства ПО (изменяемость)

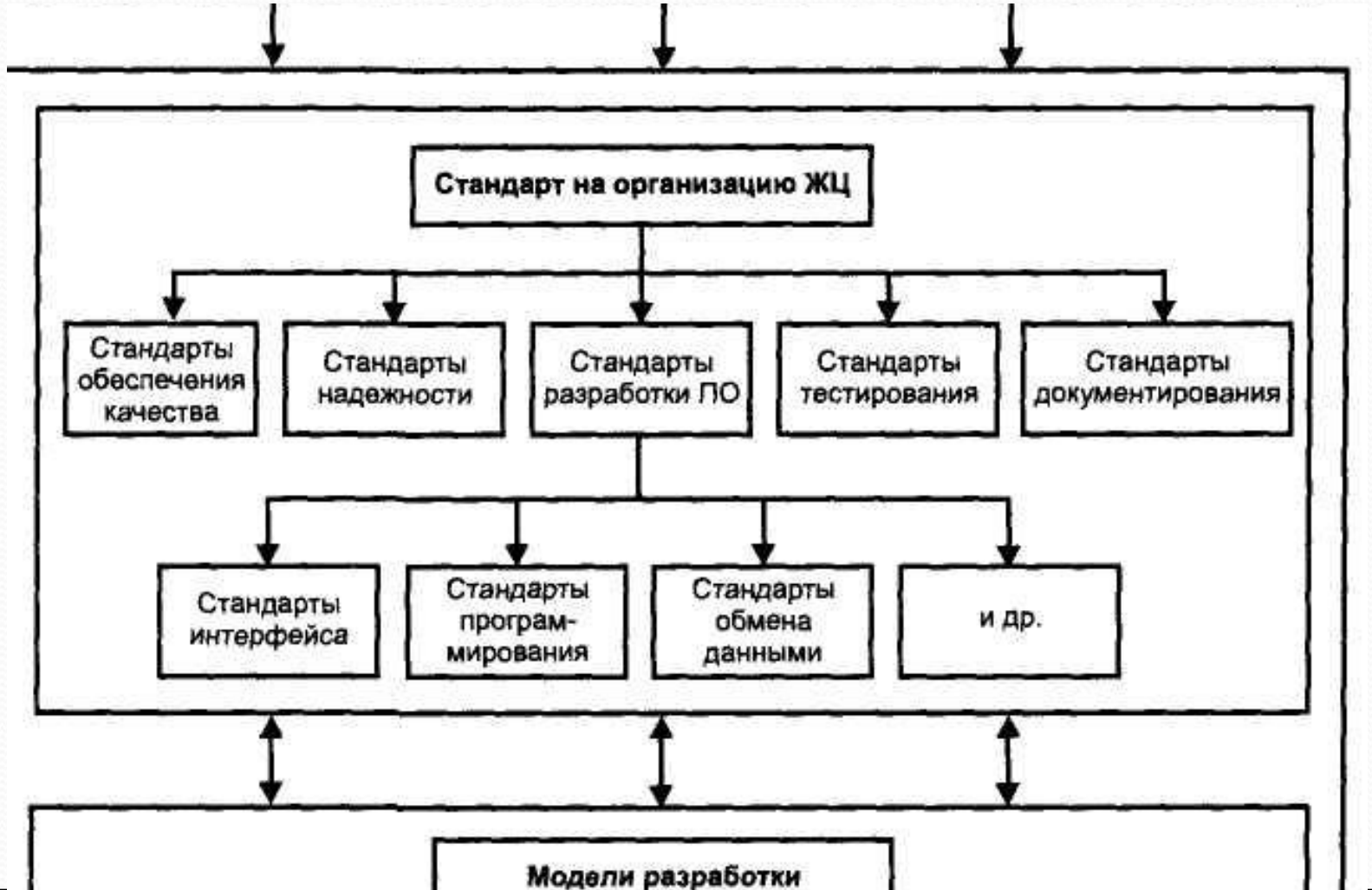
ПО легко изменить и, как следствие, требования к нему постоянно меняются в процессе разработки. Это создает много дополнительных трудностей при его разработке и эволюции.

Свойства ПО (нематериальность)

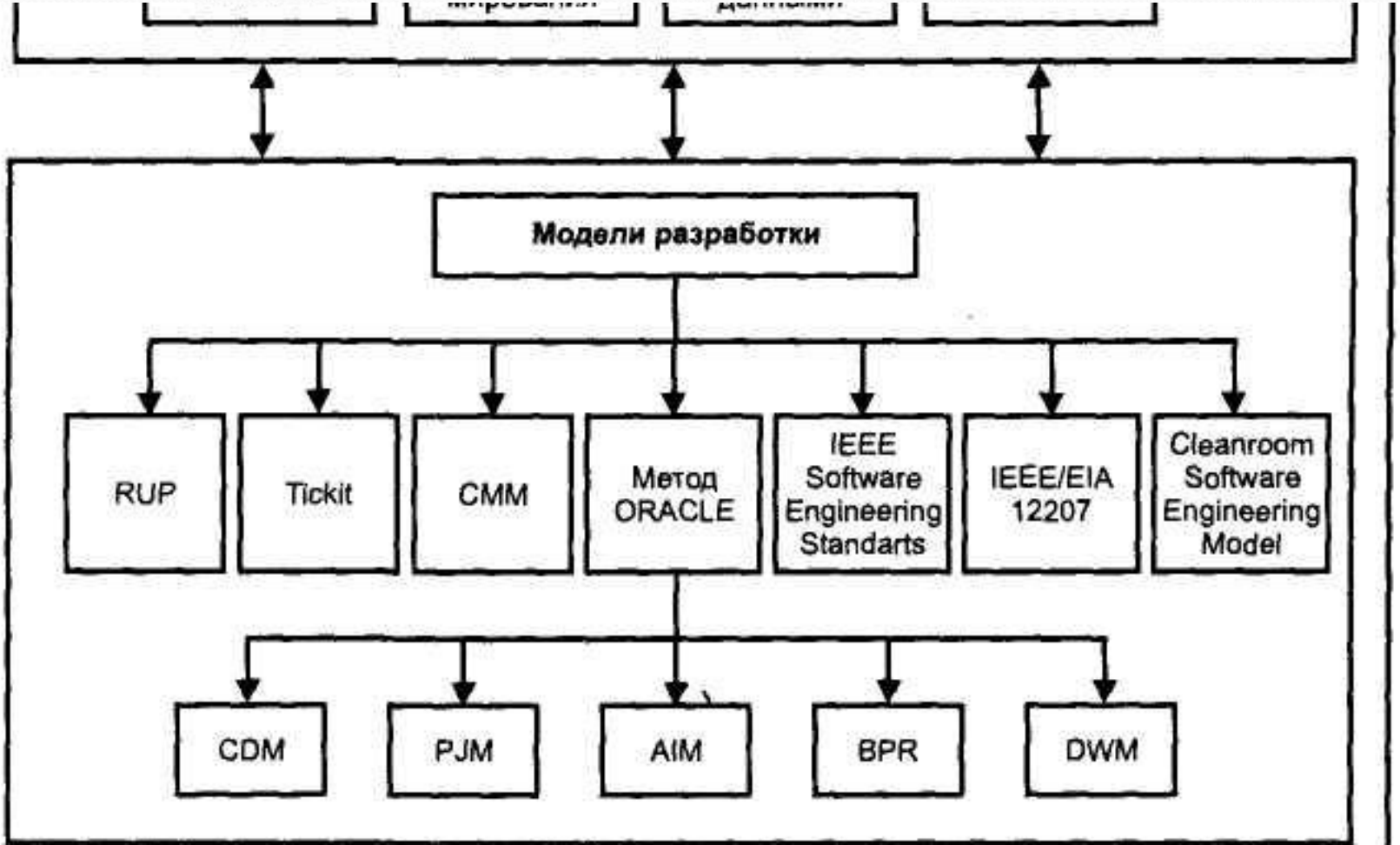
ПО невозможно увидеть, оно виртуально. Поэтому, например, трудно воспользоваться технологиями, основанными на предварительном создании чертежей, успешно используемыми в других промышленных областях (например, в строительстве, машиностроении). Там на чертежах в схематичном виде воспроизводятся геометрические формы создаваемых объектов. Когда объект создан, эти формы можно увидеть.

стандартов в области информационных технологий

стандартов в области информационных технологий



стандартов в области информационных технологий



Фаза (phase)

это определенный этап процесса, имеющий начало, конец и выходной результат. Например, фаза проверки осуществимости проекта, сдачи проекта и т.д.

Фазы следуют друг за другом в линейном порядке, характеризуются предоставлением отчетности заказчику и, часто, выплатой денег за выполненную часть работы.

Основные фазы ЖЦ ПО (пример)

1Анализ и планирование

2Проектирование

3Разработка

4Тестирование

5Документирование

6Эксплуатация / сопровождение

Критерии успешности проекта

- Качество
- Время
- Бюджет

По оценкам The Standish Group, в 2013 году во всем мире на проекты разработки и внедрения программных продуктов было потрачено $750 млрд.

Классификация успешности проектов

- **Успешные проекты (Successfull)** – проект сделан в рамках тройного ограничения, т.е. все цели проекта достигнуты в плановый срок и бюджет.
- **Провальные проекты (Failed)** – проекты, остановленные без получения результата, то есть, по сути, деньги потрачены зря.
- **Спорные проекты (Challenged)** – те проекты, которые были завершены либо с превышением сроков, либо стоили дороже, чем планировалось, либо достигли лишь части целей.
- Статистика успешности ИТ-проектов приведена далее:

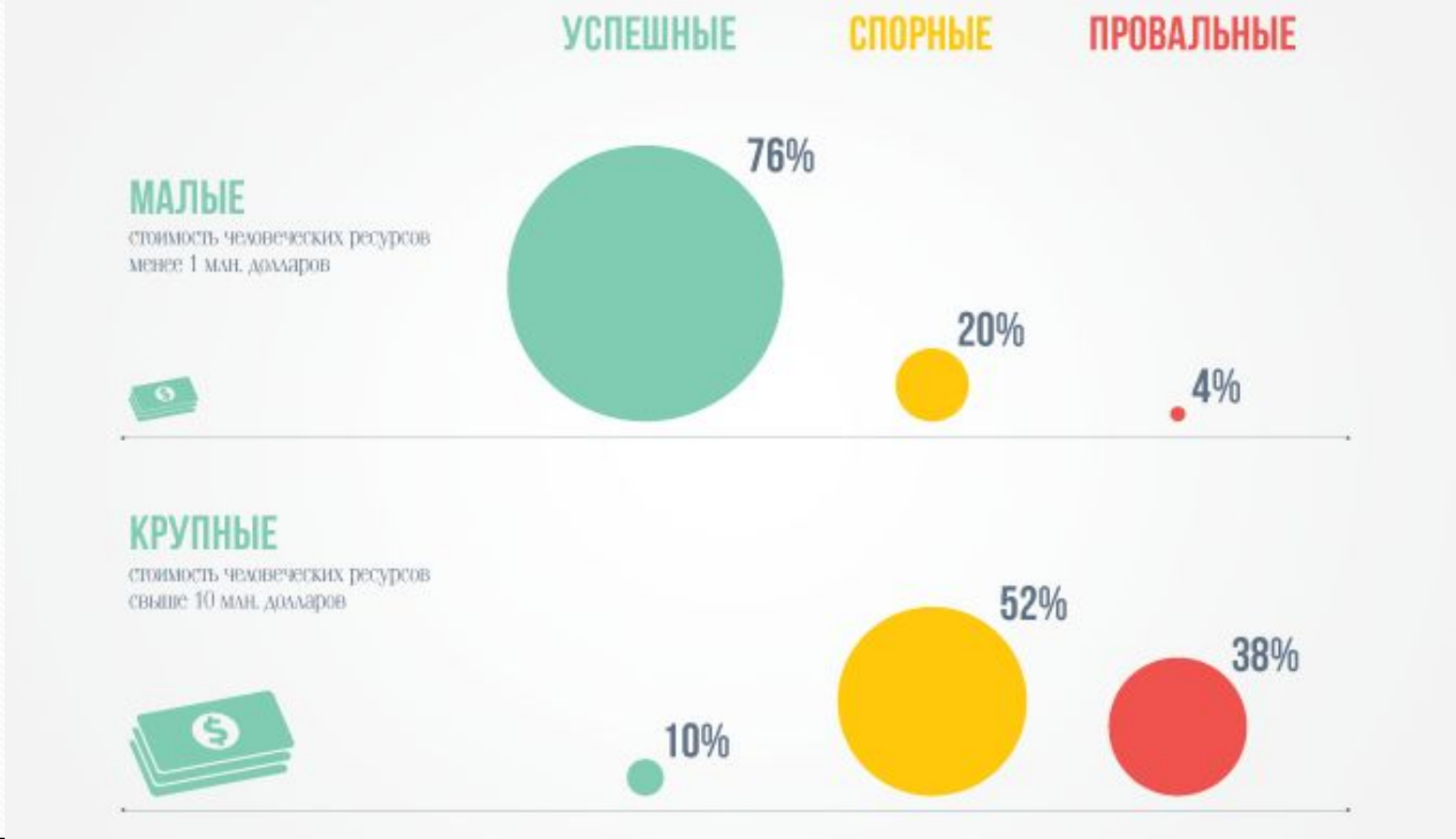
В 2013 году «лидером» по количеству неудачных проектов стали Соединенные Штаты, но Европа «отстала» не намного. По оценкам The Standish Group на так называемые «провальные» проекты потрачено $120 млрд. Что касается «спорных» софтверных проектов, то The Standish Group оценивает перерасход бюджета по ним примерно в $80 млрд., итого $200 млрд. потраченны без видимой экономической выгоды.

	2004	2006	2008	2010	2012	2013
УСПЕШНЫЕ	29%	35%	32%	37%	39%	36%
ПРОВАЛЬНЫЕ	18%	19%	24%	21%	18%	16%
СПОРНЫЕ	53%	46%	44%	42%	43%	48%

Вероятность успеха:

для больших проектов (стоимость человеческих ресурсов в проекте оказалась свыше $10 млн.) составляет всего 10%.

для малых проектов (стоимость человеческих ресурсов была менее $1 млн.) она в 7,5 раз выше.



- В 2012 (2014) году, по данным Всемирного банка (World Bank), ВВП (**валовый внутренний продукт**) России составил почти $2014,8 (1013,9) млрд.,
- т.е., затраты на софт в мире – более трети от ВВП России, которая находилась на восьмом месте в рейтинге по этому показателю.
- Если взять всю мировую экономику, то в 2012 году ее объем составил $72 440 млрд.,
- т.е. затраты на программное обеспечение достигли примерно 1% от мирового валового продукта.
- Получается, что при наличии огромного числа отраслей и сфер деятельности доля расходов на софтвер превысила 1% всей мировой экономики.

Успешность программного проекта

Программная индустрия существенно отличается от других областей производства:

- Очень высокая сложность системы
- Менее предсказуемый результат
- Хуже поддаются планированию
- До сих пор в большей степени творчество, чем ремесло

Что влияет на успешность проекта?

- Решаемая задача
- Заказчик
- Со стороны разработчика
  - Команда разработки
  - Инфраструктура
  - ***Выбранная методология проектирования ПО***

Методологии разработки ПО

- **Методология** — это система принципов, а также совокупность идей, понятий, методов, способов и средств, определяющих стиль разработки программного обеспечения.
- Методология — это реализация стандарта. Сами стандарты лишь говорят о том, что должно быть, оставляя свободу выбора и адаптации.
- Конкретные вещи реализуется через выбранную методологию. Именно она определяет, как будет выполняться разработка. Существует много успешных методологий создания программного обеспечения. Выбор конкретной методологии зависит от размера команды, от специфики и сложности проекта, от стабильности и зрелости процессов в компании и от личных качеств сотрудников.

Методологии проектирования ПО определяются

- Составом и последовательностью работ
- Ролью участников проекта
- Составом и шаблонами документов
- Организацией и управлением требованиями
- Порядком контроля и проверки качества
- Способом взаимодействия участников

Классификация методологий

Методологии представляют собой ядро теории управления разработкой программного обеспечения. К существующей классификации в зависимости от используемой в ней модели жизненного цикла (водопадные и итерационные методологии) добавилась более общая классификация на прогнозируемые и адаптивные методологии.

- **Прогнозируемые методологии** фокусируются на детальном планировании будущего. Известны запланированные задачи и ресурсы на весь срок проекта. Команда с трудом реагирует на возможные изменения. План оптимизирован исходя из состава работ и существующих требований. Изменение требований может привести к существенному изменению плана, а также дизайна проекта. Часто создается специальный комитет по «управлению изменениями», чтобы в проекте учитывались только самые важные требования.
- **Адаптивные методологии** нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. Когда меняются требования, команда разработчиков тоже меняется. Команда, участвующая в адаптивной разработке, с трудом может предсказать будущее проекта. Существует точный план лишь на ближайшее время. Более удаленные во времени планы существуют лишь как декларации о целях проекта, ожидаемых затратах и результатах.

Известные методологии проектирования ПО

Agile software development
Agile Unified Process (AUP)
Behavior Driven development (BDD)
Big Design Up Front (BDUF)
Constructionist design methodology (CDM)
Design-driven development (D3)
Design Driven Testing (DDT)
Domain-Driven Design (DDD)
Dynamic Systems Development Method (DSDM)
Evolutionary Model
Extreme Programming (XP)
Feature Driven Development
Iterative and incremental Development
Kaizen
Kanban
Lean soft development
Microsoft Solutions Framework (MSF)
Model-driven architecture (MDA)
Open Unified Process
Rapid application development (RAD)
Scrum
Rational Unified Process (RUP)
Software Craftsmanship
Spiral model
Structured Systems Analysis and Design Method (SSADM)
Team Software Process (TSP)
Test-driven development (TDD)
Unified Process (UP)
V-Model
Waterfall model
Wheel and spoke model

Характеристика методологий

- Стратегии конструирования
- Адаптивность процесса
- Этапы и связи
- Формулировка требований

Стратегии конструирования ПО

- Однократные

Определены все требования

Один цикл конструирования

Промежуточных версий нет

- Инкрементные (иногда инкрементно-итеративные)

Определены все требования

Множество циклов конструирования

Промежуточные версии могут распространяться

- Эволюционные (иногда эволюционно-итеративные)

Определены не все требования

Множество циклов конструирования

Промежуточные версии могут распространяться

Адаптивность процесса к окружению

- Тяжеловесные (прогнозирующие):
  - Фиксированные требования
  - Большая команда
  - Разная квалификация разработчиков
- Адаптивные (облегченные):
  - Постоянно меняющиеся требования
  - Маленькая команда
  - Высококвалифицированные разработчики

Охарактеризуем методологии

- Классическая (водопадная) модель
  - Общепринятая линейная модель
  - Классическая итерационная
  - Каскадная модель
  - Строгая каскадная модель
- Прототипирование (макетирование)
- Инкрементная модель
- Быстрая разработка приложений (RAD)
- Спиральная модель
- Rational Unified Process (RUP)
- Microsoft Solutions Framework (MSF)
- Экстремальное программирование (XP)
- Scrum

Характеристика методологий

	Стратегии	Адаптивность	Полнота
Классическая	Однокр.	Прогн.	+
Прототипирование	Эвол.	Прогн.	-
Спиральная	Эвол.	Прогн.	+
Инкрементная	Инкр.	Прогн.	+
RAD	Инкр.	Прогн.	+
RUP	Инкр./Эвол.	Прогн.	+
MSF	Однокр./Эвол.	Прогн./Адапт.	+
XP	Эвол.	Адапт.	+
Scrum	Эвол.	Адапт.	+

Как выбрать методологию?

Выбор зависит от:

- Решаемых задач (из какой она области, как она сформулирована…)
- Сроком реализации (гибкие подходят для коротких, классические – не подходят)
- Команды разработчиков
  - Размер команды разработчиков
  - Опыт команды разработчиков
  - Сработанность команды разработчиков
  - Местонахождение разработчиков

  {Влияет на общение, например, оффшорная компания – часть команды в Америке, России, Китае: одни спят, одни работают.

  Влияет и в какой стране: разная разработка в России, Индии (национальный аспект культурный: если поставить из высшей касты в подчинение кого-то из нижней касты, работать система не будет), Китае (невыполнение задания разработчиком может повлечь уголовную ответственность))
  - Механизмы взаимодействия в команде разработчиков
- Заказчика
  - Требований заказчика
  - Механизмов взаимодействия с заказчиком (можно поговорить, или только списаться…)
- И т.д.

Чем отличаются различные методологии проектирования?

- Этапы
  Список этапов
  Последовательность этапов
  Связи между этапами
  Состав этапов
  Объемы (длительность) этапов
- Требования
  Формулировка требований
  Корректировка требований
- Команда
  Размер
  Роли участников проекта
  Способами взаимодействия участников
- Артефакты
  Состав и содержание
  Время получения артефактов (например, версий)
  Объем и состав документации
- Заказчик
  Квалификация заказчика
  Степень участия заказчика
- Порядок контроля и проверки качества

Статистика использования методологий

- В 2009г. опросили более 1000 различных ИТ разработчиков (какую методологию проектирования они используют?)
- Результат опроса (по количеству проектов):
  35% - гибкие (Scrum, XP и др)
  30% - не используют никаких
  21% - разные варианты итеративных (эволюционных) процессов (RUP, спиральные и др.)
  13% - различные модификации классической (водопадной) модели

По статистике Standish Group выбор модели жизненного цикла влияет на успех проекта.

Среди проектов с каскадным ЖЦ

- успешны 26%,
- ущербны 59%,
- аннулированы 15%.

Для проектов с итерационным ЖЦ соответствующие показатели составили:

- 45%,
- 43%
- и 12%.

Характеристика методологий

- Стратегии конструирования
- Адаптивность процесса
- Этапы и связи
- **Формулировка требований** *(В чем проблема?)*

В чем проблема?

«Самой сложной задачей при создании
программной системы является точное
определение того, что требуется создать…
Ни одна задача не приносит такого же вреда
Конечной системе в случае ошибки.
И нет ни одной задачи настолько же
сложной в исправлении последствия»
Фредерик Брук

(Американский учёный в области теории вычислительных систем, автор книги «Мифический человеко-месяц».

Управлял разработкой OS/360 в IBM.

Награждён Премией Тьюринга в 1999 году.)

«Мифический человеко-месяц или Как создаются программные системы»

— книга Фредерика Брукса — книга Фредерика Брукса об управлении проектами — книга Фредерика Брукса об управлении проектами в области разработки программного обеспечения, центральной темой которой стало то, что привнесение в проект новых сил на поздних стадиях разработки лишь отодвигает срок сдачи проекта. Эта идея стала известна под названием «закон Брукса».

Проблемы определения требований

- Разработка требований – самая сложная часть проектирования ПО
- Требования постоянно меняются
- Требования могут быть
  - Неясны
  - Двусмысленны
  - Противоречивы
- Спецификации могут быть неполны
- Пользователи, излагающие требования, непредставительны

План раздела «Инженерия требований»

- Определение требований
- Разработка требований
  - Выявление требований
  - Анализ требований
- Документирование и организация (структуризация) требований
- Изменение требований
- Планирование и управление требованиями

Инженерия требований.

Разработка требований

- Выявление требований
  - Исследование
  - Интервью
  - Семинар
  - Создание прототипа
  - Use Case
- Анализ требований
  - Уточнение требований
  - Структуризация
  - Установка приоритетов

Документирование и организация требований

- Состав и распределение работ
- Спецификация требований
- Концепция эксплуатации
- Начальный план разработки ПО
- Критерии принятия работ

Управление требованиями

- Изменение требований
  - Предложение изменений
  - Анализ изменений
  - Принятие решений
  - Обновление требований
  - Обновление планов
- Контроль версий требований
  - Управление состояниями требований
  - Прослеживаемость требований

Требования

- Требование - это условие или возможность, которой должна соответствовать система.

- Требование по *IEEE* 1990 :
- условия или возможности, необходимые пользователю для решения проблем или достижения целей;
- условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;
- документированное представление условий или возможностей для пунктов 1 и 2.

Свойства требований

- *Корректность*
- *Ясность, недвусмысленность*
- *Полнота и непротиворечивость*
- *Приоритезация*
- *Необходимый уровень детализации*
- *Отслеживаемость*
- *Прослеживаемость*
- *Тестируемость и проверяемость*
- *Модифицируемость*

Свойства требований

- *Корректность* — требования должны описывать то, что нужно заказчику.
- *Ясность, недвусмысленность* — однозначность понимания требований заказчиком и разработчиками. Часто этого трудно достичь, поскольку конечная формализация требований, выполненная с точки зрения потребностей дальнейшей разработки, трудна для восприятия заказчиком или специалистом предметной области, которые должны проинспектировать правильность формализации.
- *Полнота* — требования должны охватывать все аспекты описываемой системы.
- *Непротиворечивость* — не должно быть 2-х требований удовлетворение которым вызовет противоречие.
- *Приоритезация.* Требования формируются в отдельные наборы, отличающиеся приоритетом. Некоторые требования более важные. Не могут все требования иметь одинаковую категорию важности.

Свойства требований

- *Необходимый уровень детализации*. Требования должны обладать ясно осознаваемым уровнем детализации, стилем описания, способом формализации: либо это описание свойств предметной области, для которой предназначается ПО, либо это техническое задание, которое прилагается к контракту, либо это проектная спецификация, которая должна быть уточнена в дальнейшем, при детальном проектировании. Либо это еще что-нибудь. Важно также ясно видеть и понимать тех, для кого данное описание требований предназначено, иначе не избежать недопонимания и последующих за этим трудностей. Ведь в разработке ПО задействовано много различных специалистов – инженеров, программистов, тестировщиков, представителей заказчика, возможно, будущих пользователей – и все они имеют разное образование, профессиональные навыки и специализацию, часто говорят на разных языках. Здесь также важно, чтобы требования были максимально абстрактны и независимы от реализации.

Свойства требований

- *Прослеживаемость* — важно видеть то или иное требование в различных моделях, документах, наконец, в коде системы. А то часто возникают вопросы типа – "Кто знает, почему мы решили, что такой-то модуль должен работать следующим образом ....?". Прослеживаемость функциональных требований достигается путем их дробления на отдельные, элементарные требования, присвоение им идентификаторов и создание трассировочной модели, которая в идеале должна протягиваться до программного кода. Хочется например, знать, где нужно изменить код, если данное требование изменилось. На практике полная формальная прослеживаемость труднодостижима, поскольку логика и структура реализации системы могут сильно не совпадать с таковыми для модели требований. В итоге одно требование оказывается сильно "размазано" по коду, а тот или иной участок кода может влиять на много требований. Но стремиться к прослеживаемости необходимо, разумно совмещая формальные и неформальные подходы.

Возможность найти соответствие в строчках спецификации и строчках порождаемых артефактов (документации, тестах, программном коде и др.)

Свойства требований

- *Тестируемость и проверяемость* — необходимо, чтобы существовали способы оттестировать и проверить данное требование. Причем, важны оба аспекта, поскольку часто проверить-то заказчик может, а вот тестировать данное требование очень трудно или невозможно в виду ограниченности доступа (например, по соображениям безопасности) к окружению системы для команды разработчика. Итак, необходимы процедуры проверки –выполнение тестов, проведение инспекций, проведение формальной верификации части требований и пр. Нужно также определять "планку" качества (чем выше качество, тем оно дороже стоит!), а также критерии полноты проверок, чтобы выполняющие их и руководители проекта четко осознавали, что именно проверено, а что еще нет.

Например, требование программа должна работать быстро не проверяемо (оно должно быть количественно).

- *Модифицируемость*. Определяет процедуры внесения изменений в требования.

Виды требований

- *Функциональные* (ЧТО система должна уметь делать)
  - Бизнес-требования
  - Пользовательские требования
- *Нефункциональные* (КАК система должна быть сделана)
  - Ограничения
  - Требования к качеству

Виды требований

- *Функциональные* требования являются детальным описанием поведения и сервисов системы, ее функционала. Они определяют то, ЧТО система должна уметь делать.
- *Нефункциональные* требования не являются описанием функций системы. Этот вид требований описывает такие характеристики системы, как надежность, особенности поставки (наличие инсталлятора, документации), определенный уровень качества (например, для новой Java-машины это будет означать, что она удовлетворяет набору тестов, поддерживаемому компанией Sun). Сюда же могут относиться требования на средства и процесс разработки системы, требования к переносимости, соответствию стандартам и т.д. Требования этого вида часто относятся ко всей системе в целом. На практике, особенно начинающие специалисты, часто забывают про некоторые важные нефункциональные требования.

Функциональные требования

- Бизнес-требования

  Формулируются заказчиком

  Описывают цели, которые требуется достичь с помощью данной системы
- Требования пользователей

  Какие задачи можно решить с помощью системы
- Собственно функциональные требования

  Определяется функциональность, которую необходимо реализовать

Нефункциональные требования

- Требования к характеристикам качества
  - Требования к надежности
  - Требования к совместимости
  - Требования к эффективности
  - Требования к гибкости
  - Требования к эргономике
- Ограничения
  - Соответствие стандартам и правилам системы
  - Бюджет
  - Предопределенные архитектурные решения

Эргоно́мика

- (от др.-греч. (от др.-греч. ἔργον — работа и νόμος— «закон») — в традиционном понимании — наука о приспособлении должностных обязанностей, рабочих мест, предметов и объектов труда, а также компьютерных программ для наиболее безопасного и эффективного труда работника, исходя из физических и психических особенностей человеческого организма.

Ограничения

- Мы сделаем проект
  - Быстро
  - Качественно
  - Недорого
- Выберите 2 из 3-х

Что не является требованием?

- Детали архитектуры
- Детали реализации
- Сведения о планировании
- Сведения о тестировании
- Проектная информация
  - Инфраструктура разработки (системы контроля версий)
  - Процесс разработки
  - Команда разработки

Разработка требований

- Выявление требований
- Анализ требований

- Результат – спецификация требований

Выявление требований (кто?)

- Заинтересованные лица
  - Заказчики
  - Менеджеры
  - Пользователи
    - Операторы
    - Менеджеры
- Разработчики
- Служба поддержки
- Другие лица

- ВАЖНО: заказчик $\neq$ пользователь

Выявление требований

Формирование требований очень сложный процесс

- О размере документа. Аппаратные и программные требования к одному из смартфонов составляют более 7000 страниц (цена ошибки очень велика)
- При планировании необходимо определить:
  - Цели выявления требований
  - Стратегии и процессы выявления требований (интервью, опросы…)
  - Что будет результатом усилий по выявления требований (нужно определиться: спецификация, несколько документов…)
  - Оценка календарного плана и ресурсов выявления требований
  - Риски, связанные с выявления требований

Как можно не «утонуть» в этом документе и разрешить вопросы полноты и др.?

Выявление требований (способы)

- Способы выявления требований

  Исследования предметной области (рынка…)

  Интервью

  Семинар

  Создание прототипов

  Создание вариантов использования (Use Case)

  (показывают функциональные системы с точки зрения имеющихся прецедентов и задействованных лиц). Могут общим языком взаимодействия.

Выявление требований

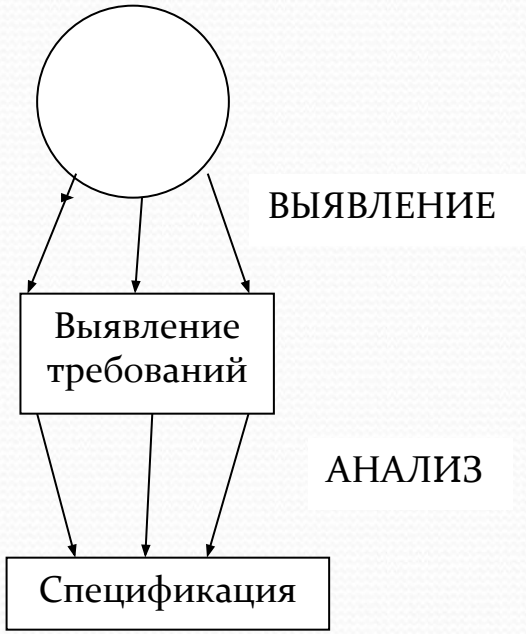
- Проблемы определения требований
  - Ожидания пользователей
  - Умение оценить противоречивые требования
  - Недостаточные требования
  - Умение понять требования пользователей

- Проблемы
  - Формулирования требований
  - Терминологии (например, клиент)
  - Неявных допущений (контекст – правила, система всегда многопользовательская)
  - Предвзятые решения

Разработка требований

- Выявление требований

  Расходящийся процесс, цель которого собрать как можно больше данных
- Анализ требований - сходящийся процесс:
  - Уточняет данные
  - Структурирует информацию
  - Устанавливает приоритеты
- Результат анализа – спецификация требований



О спецификациях

Спецификация - достаточно точное и достаточно полное описание задачи, которое человеку, участвующему в решении, написать, понять и прочесть легче, чем программу решения этой задачи на доступном ему языке программирования.

Более коротко спецификацию можно определить как подробное описание некоторой работы, подлежащей выполнению.

Средства спецификации - любые средства получения или построения таких описаний.

Язык спецификации - рационально оформленный и синтаксически организованный набор таких средств.

В спецификациях программ имеет смысл выделить две существенно отличающиеся части.

Функциональные спецификации, описывающие функцию программы.

Эксплуатационные спецификации, описывающие скорость работы программы, используемые ресурсы, характеристики аппаратуры, специальные требования к надежности и безопасности.

О спецификациях

- Существует точка зрения об относительности понятия спецификации. Имея дело с одной и той же задачей, данные люди в данном окружении (включающем их знания, языки и т. п.) будут склонны считать спецификацией одно, а другие люди в другой обстановке - совсем другое.
- Например "x := x + i" будет лучшей спецификацией задачи , чем "увеличить x на единицу" для человека, знакомого с оператором присваивания и его обозначением.

Классификация спецификаций по уровню формализации

- Словесные спецификации, обработка которых может осуществляться обычным текстовым редактором.
- Модельные (структурированные) спецификации, которые предполагают построение схем, диаграмм, других информационных структур.
- Формальные спецификации, получаемые строгим формальным способом с использованием математических формализмов, которые обеспечивают полное определение семантики.

Классификация спецификаций по способу представления

Существует два основных способа представления спецификаций:

- Текстовое представление, которое подходит для словесных и формальных спецификаций.
- Представление с помощью информационных объектов. Как правило, таким способом представляются модельные спецификации.

Шаблоны спецификаций

- Существуют различные государственные, отраслевые и корпоративные стандарты
- Наиболее распространенные в России:
  - IEEE 830-1998 «Recommended Practice for Software Requirements Specifications» (описывает, как спецификацию формировать и представлять с помощью разных шаблонов)
  - ГОСТ 34.602–89 «Техническое задание на создание автоматизированной системы»
- Шаблон не должен являться догмой (если это не требование заказчика)
- Следует при необходимости модифицировать шаблоны в соответствии с природой и потребностями проекта

- Полезный документ: IEEE Guide for Developing **System Requirements Specications (SRS)** (описывает, как необходимо создавать спецификацию требований)

Шаблон спецификации требований (IEEE 830-1998).

1. Введение
1.1.Назначение
1.2. Область действия
1.3. Определения, акронимы и сокращения
1.4. Публикации
1.5. Краткий обзор

2. Общее описание
2.1. Перспектива изделия
2.2. Функция изделия
2.3. Характеристики пользователей
2.4. Ограничения
2.5. Допущения и зависимости
2.6. Разделение требований

3. Специфические требования
3.1. Внешние интерфейсы
3.2. Функции системы
3.3. Требования к рабочим характеристикам
3.4. Логические требования к базе данных
3.5. Проектные ограничения
3.6. Атрибуты системы программного обеспечения
(нефункциональные требования)

Шаблон спецификации требований (IEEE 830-1998). Общая часть (разделы 1 и 2).

1. Введение
1.1.Назначение
1.2. Область действия
1.3. Определения, акронимы и сокращения
1.4. Публикации
1.5. Краткий обзор

2. Общее описание
2.1. Перспектива изделия
2.2. Функция изделия
2.3. Характеристики пользователей
2.4. Ограничения
2.5. Допущения и зависимости
2.6. Разделение требований

Шаблон спецификации требований (IEEE 830-1998). Специфические требования (раздел 3).

3. Специфические требования
3.1. Внешние интерфейсы
3.2. Функции системы
3.3. Требования к рабочим характеристикам
3.4. Логические требования к базе данных
3.5. Проектные ограничения
3.6. Атрибуты системы программного обеспечения
(нефункциональные требования)

Шаблон раздела 3 SRS (по режимам. V1)

3.1 Требования к внешним интерфейсам
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Функциональные требования
3.2.1 Режим 1
3.2.1.1 Функциональное требование 1.1
…
3.2.1.n Функциональное требование 1.n
3.2.2 Режим 2
…
3.2.m Режим m
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования

Например, режимы для ПО управления самолетом: 1) штатный режим (самолет летит), 2) диагностический, 3) тренажер.

Шаблон раздела 3 SRS (по режимам. V2)

3.1 Функциональные требования
3.1.1 Режим 1
3.1.1.1 Внешние интерфейсы
3.1.1.1.1Интерфейсы пользователя
3.1.1.1.2 Аппаратные интерфейсы
3.1.1.1.3 Интерфейсы программного обеспечения
3.1.1.1.4 Интерфейсы связи
3.1.1.2 Функциональные требования
3.1.1.2.1 Функциональные требования 1
…
3.1.1.2.n Функциональные требования n
3.1.2 Режим 2
…
3.1 m Режим m
3.2 Проектные ограничения
3.3 Атрибуты системы программного обеспечения
3.4 Другие требования

Шаблон раздела 3 SRS (по классам пользователей)

3.1 Внешние интерфейсы
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Функциональные требования
3.2.1 Класс пользователей 1 (Менеджер/Продавец - консультант)
3.2.2 Класс пользователей 2 (Начальствующий штат сотрудников)
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования

Например, классы пользователей для ПО управления самолетом:
1)пилот,
2) стюардесса,
3) механик.

Шаблон раздела 3 SRS (по объектам)

3.1 Внешние интерфейсы
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Классы/объекты
3.2.1 Класс/объект 1
3.2.1.1 Атрибуты
3.2.1.1.1 Атрибут 1
…
3.2.1.1.n Атрибут n
3.2.1.2 Функции
3.2.1.2.1 Функциональные требования 1.1
…
3.2.1.2.m Функциональные требования 1.m
3.2.1.3 Сообщения
3.2.2 Класс/объект 1
…
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования

Наиболее применим для информационных систем.
Например, система состоит из модулей (кабина самолета, хвост), у них есть свойства.

Шаблон раздела 3 SRS (по свойствам)

3.1 Внешние интерфейсы
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Свойства системы
3.2.1 Свойство системы 1
3.2.1.1 Назначение свойства
3.2.1.2 Последовательность стимулов/откликов
3.2.1.3.1 ФТ 1
…
3.2.1.3.n ФТ n
3.2.2 Свойство системы 2
…
3.2.m Свойство системы m
…
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования

Например, уклонение от боевой ракеты

Шаблон раздела 3 SRS (по функциональной иерархии)

3.1 Внешние интерфейсы
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Функциональные требования
3.2.1 Информационные потоки
3.2.1.1 Схема потока данных 1
…
3.2.1.n Схема потока данных n
3.2.2 Описание процессов
3.2.2.1 Процесс 1
…
3.2.2.m Процесс m
3.2.3 Спецификации структуры данных
3.2.3.1 Структура 1
…
3.2.3.r Структура r
3.2.4 Словарь данных
3.2.4.1 Элемент данных 1
…
3.2.4.t Элемент данных t
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования

Шаблон раздела 3 SRS (множественная организация)

3.1 Внешние интерфейсы
3.1.1 Интерфейсы пользователя
3.1.2 Аппаратные интерфейсы
3.1.3 Интерфейсы программного обеспечения
3.1.4 Интерфейсы связи
3.2 Функциональные требования
3.2.1 Класс пользователей 1
3.2.1.1 Свойство 1.1
3.2.1.1.1 Последовательность стимулов/откликов
3.2.1.1.2 Ассоциативные ФТ
3.2.1.n Свойство 1.n
3.2.2 Класс пользователей 2
3.2.3 Класс пользователей m
3.3 Требования к рабочим характеристикам
3.4 Проектные ограничения
3.5 Атрибуты системы программного обеспечения
3.6 Другие требования