# Вопрос №3.22

*Требования к ПО. Анализ требований. Управление требованиями. Оценка качества требований. Трассировка требований.*

# 1 Требования к ПО

Для разных видов ПО требования имеют разную природу (см. таблицу). Сокращения: BA — Business Analysis, RM — Requirements Management.

| | Много клиентов | Несколько клиентов | Один клиент |
|---|---|---|---|
| Продукты | **Коробочные продукты**<br>За BA и RM борются отделы маркетинга и продаж. Что на самом деле хочет заказчик, никто не знает и никого это не интересует. | **Решения**<br>BA делает заказчик, но объяснить, что он хочет, толком не может. Vendor вместе с заказчиком составляют требования, потом их обрабатывает маркетинг. | **Заказное ПО**<br>BA делает заказчик, а RM — vendor. Считается, что заказчик прав, так как знает предметную область. |
| Сервисы | **Массовые сервисы**<br>Всем занимается маркетинг. | **Настройка продуктов**<br>См. первую строку. | **Consulting**<br>Всем занимается Vendor. Заказчик ничего не знает и его никто не спрашивает. |

Как видно из таблицы, самая сложная ситуация со второй колонкой. Именно для такого вида ПО обычно нужно управление требованиями.

# 2 Анализ требований

По тому, что описывают требования, их обычно делят на *функциональные* (описывают функции системы), *нефункциональные* (описывают другие свойства) и *специальные* (никто не знает, что это такое). По уровню абстракции требования делят на требования *бизнес уровня* (требования в терминах решаемой задачи), *пользовательского уровня* (в терминах интерфейса с пользователем) и *уровня приложения* (в программных терминах).

Требования бизнес уровня оформляются в виде документа Vision/Scope. Фукнциональные требования пользовательского уровня оформляются как Use Case document. Наконец, требования уровня приложения оформляются как Requirement Specification (RS) — по нашему ТЗ.

| | Фукнциональные | Нефункциональные |
|---|---|---|
| Бизнесс | Business Needs | Regulations |
| Пользовательский | Use Cases | QoS, Business Rules |
| Приложения | Func. req. spec. | Protocols, constraints |

QoS (Quality of Service) — это требования ко времени выполнения конкретных операций.

На самом раннем этапе (в документе Scope) должны быть определены:

- Контекст, в котором работает система и интерфейсы (протоколы) взаимодействия с этим контекстом. Кто реализует эти протоколы?
- Лицензии.
- Какие компьютеры у заказчика?

Полезные артефакты для составления требований:

- описание аналогичных систем,
- характеристики пользователей,
- data dictionary,
- XP user stories,
- vision document,
- контекстная диаграмма.

Зачем нужны требования?

- На их основе подписывается контракт.
- Они позволяют разбить проект на фазы и версии.
- Приоритеты требований позволяют определить приоритеты задач и построить план работ.

# 3 Управление требованиями

Относительно обсуждения с заказчиком требования бывают:

- **determined** (ничего не попишешь!), например, требования, связанные с окружением, операционной системой;
- **known** (все согласны);
- **to clarify** (подлежит выяснению).

Атрибуты требований: status, budget, priority, dependencies, scope (текстовое описание), code (id).

Примеры систем управления требованиями: Borland Caliber RM, IBM Requisite Pro.

Функции систем управления требованиями:

- редактор,
- вычисление метрик,
- трассировка,
- срезы (выборки по запросу),
- автоматическая генерация прототипа.

Прототипы:

- визуальные
  - интерактивные (модели экранов со ссылками друг на друга)
  - неинтерактивные (просто модели экранов)
- функциональные
  - горизонтальные (реализуют всю функциональность поверхностно)
  - вертикальные (реализуют одну функцию, но до конца)
  - эволюционные (потом превращаются в реальную систему)

Существует несколько моделей жизненного цикла требований. Например, модель RDEM:

1. Captured
2. Specified
3. Planned
4. Realized

# 4 Оценка качества требований

Качества хорошего требования:

- тестируемость,
- реализуемость,
- однозначность,

- осмысленность (обоснованность).

Качества хорошего набора требований:

- полнота,
- непротиворечивость,
- отсутствие циклических зависимостей,
- отсутствие дубликатов/пересечений,
- модифицируемость.

# 5 Трассировка требований

Что делать, если заказчик хочет изменить требования во время работы над проектом?

*Контроль изменений* (Requirement Changes Tracking). Необходимо оценить масштаб изменения, область изменения, вписывается ли данное изменение в архитектуру системы. На основе этих данных сделать вывод о затратах на реализацию этого изменения. Кроме того, необходимо выяснить приоритет (есть ли у заказчика действительно потребность в этом изменении и насколько она обоснованна?)

Если требования часто меняются, необходима *трассировка требований*. Для этого требования объединяются в специальные структуры:

- *Дерево требований* — дерево, в котором требования-дети являются уточнениями требования-родителя. Такое дерево построить довольно легко, однако в нем нет всех зависимостей и неизвестно, что произойдет, если какое-то требование изменится.
- *Граф требований* — граф, в котором вершины соответствуют требованиям, а дуги — зависимостям между ними. Такой граф очень полезен, по нему можно отследить, какие другие требования затронет изменение определенного требования. Хорошо, если этот граф обладает высокой кластеризацией (состоит из небольших сильносвязных компонент, которые между собой слабо связаны). Новые требования, которые создают много новых связей в этом графе — плохие требования (дорогие).
- Еще лучше построить две таблицы: *матрицу трассировки* $R \times R \to \{0, 1\}$, которая отражает зависимости между требованиями, и *матрицу функциональных компонент* $R \times F \to \{0, 1\}$, которая показывает, в каких функциональных компонентах системы (модулях) реализуются те или иные требования. Тогда при изменении одного требования мы сможем не только проследить, какие другие требования это затронет, но и какие модули придется менять.