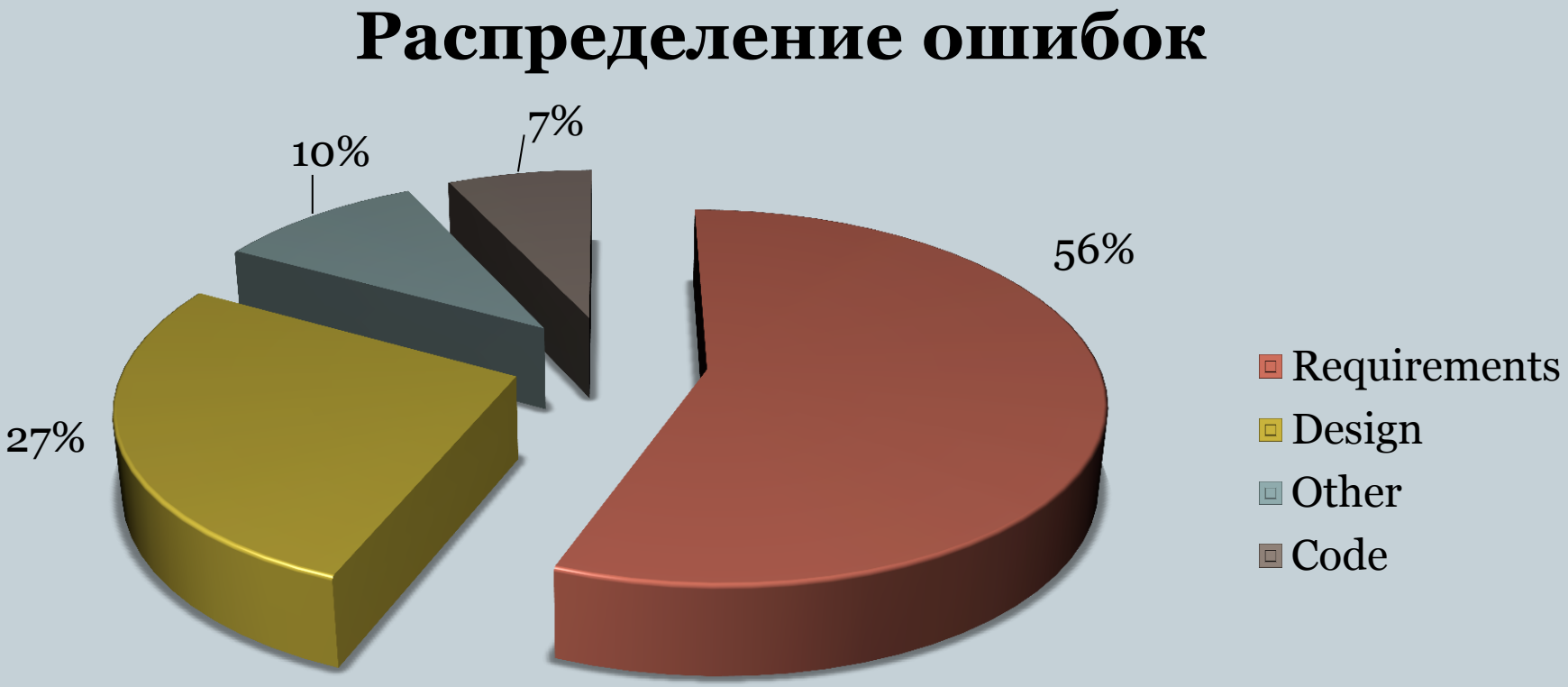
Тестирование требований

PHD. MARYNA DIDKOVSKA
TEST MANAGER
VIDEO INTERNET TECHNOLOGIES LTD.
MD@VIT.UA

Важность


Распределение ошибок
10%
7%
56%
27%
Requirements
Design
Other
Code

Относительная стоимость исправления багов

Phase in Which Found	Cost Ratio
Requirements	1
Design	3-6
Coding	10
Unit/Integration Testing	15-40
System/Acceptance Testing	30-70
Production	40-1000

Требования. Определения

- Требования – это спецификация того, что должно быть реализовано.

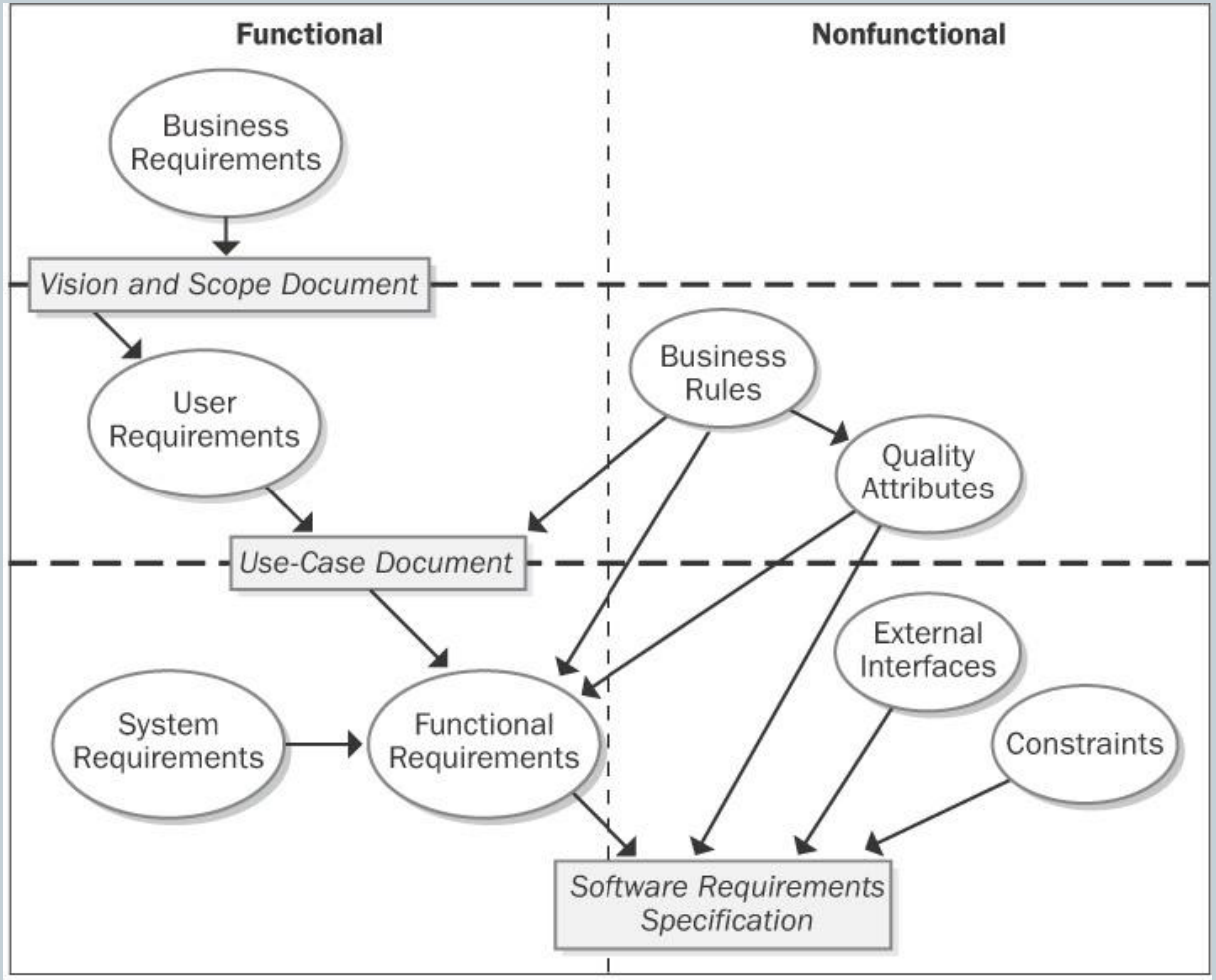
- Требования описывают то, что необходимо реализовать, без детализации технической стороны решения

Что, а не *как*

Почему проекты не всегда успешны?

- Требования и спецификации неполны
- Требования и спецификации слишком часто изменяются.
- При составлении требований мнение пользователей слабо учитывалось.

Уровни и типы



Характеристики требований

- Недвусмысленность
- Полнота
- Правильность
- Осуществимость
- Необходимость
- Приоритезированность
- Проверяемость

Характеристики спецификации

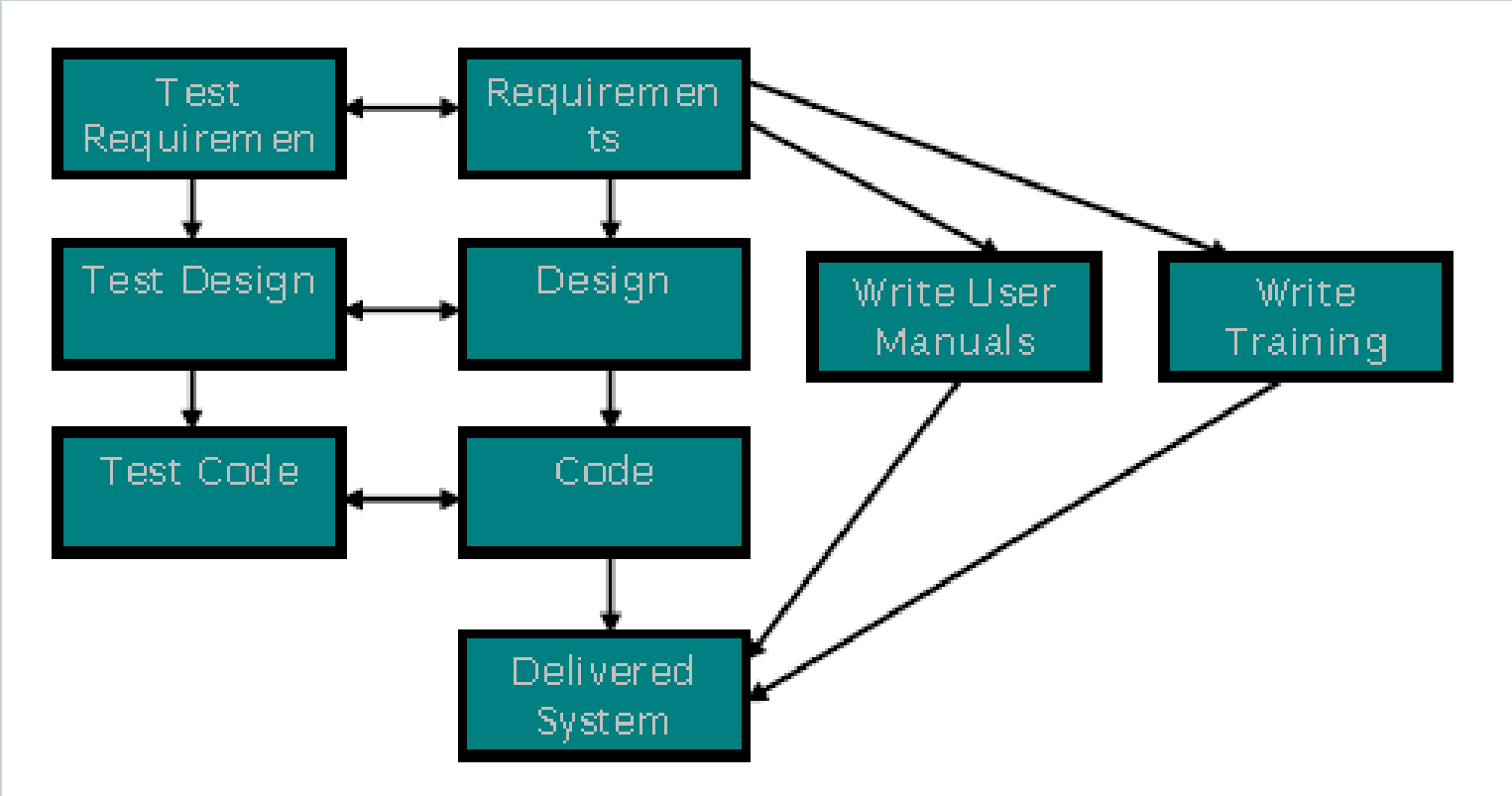
- Полнота
- Последовательность
- Модифицируемость
- Прослеживаемость

Прослеживаемость требований

User Requirement	Functional Requirement	Design Element	Code Module	Test Case
UC-28	catalog.query.sort	Class catalog	catalog.sort()	search.7 search.8
UC-29	catalog.query.import	Class catalog	catalog.import() catalog.validate()	search.12 search.13 search.14

Changes are not so dangerous
when you can manage them

Влияние требований



Процесс тестирования требований

1. Validate requirements against objectives
2. Apply scenarios against requirements
3. Perform initial ambiguity review
4. Perform domain expert reviews
5. Create cause-effect graph
6. Logical consistency check by RBT Tool
7. Review of test cases by specification writers
8. Review of test cases by users
9. Review of test cases by developers
10. Walk test cases through design
11. Walk test cases through code
12. Execute test cases against code

Валидация требований

1. Экспертиза спецификации
2. Тестирование требований
3. Определение критериев сдачи продукта

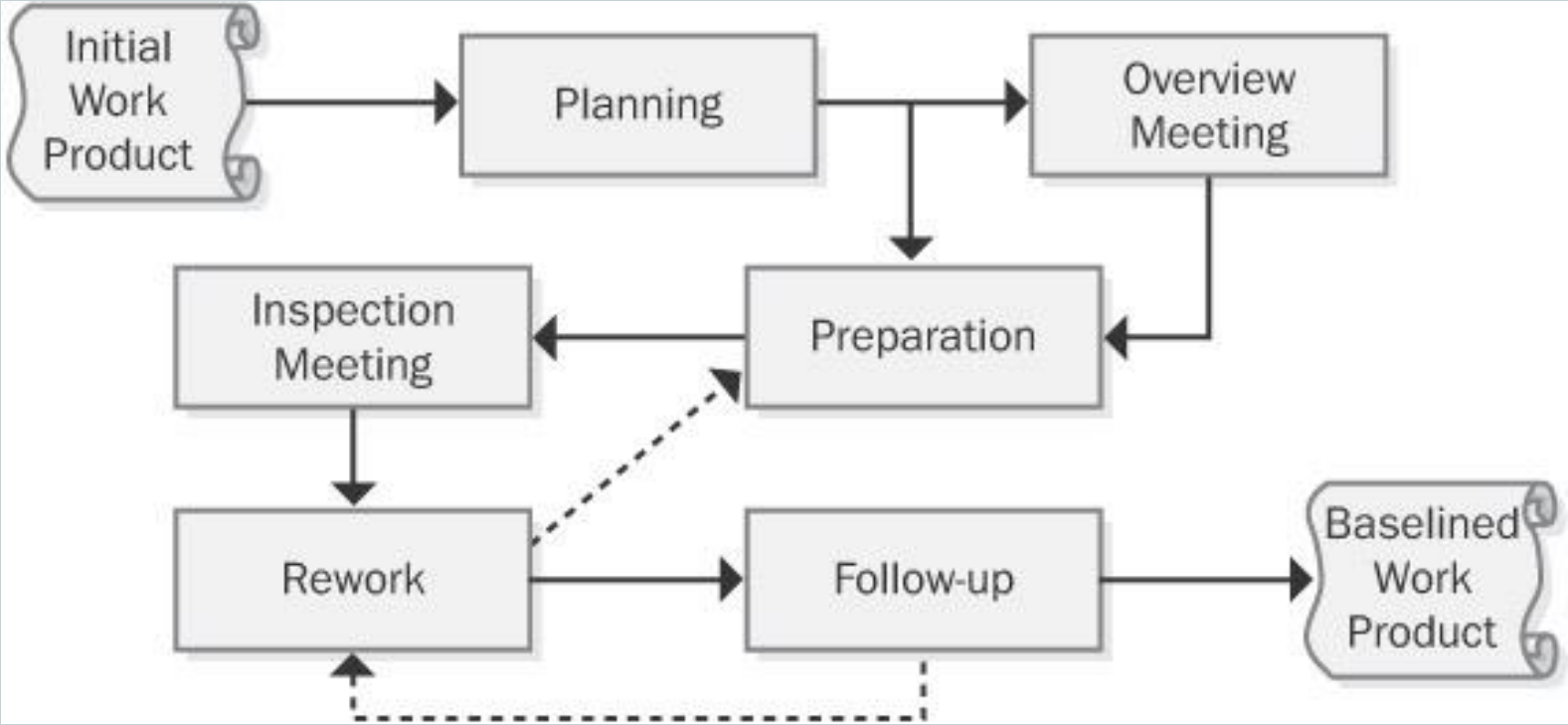
Review: Участники

- Автор работы
- Автор(ы) предшествующих документов
- Те, кто впоследствии будут использовать данный документ в своей работе
- Те, кто отвечают за уже существующие продукты, которые должны будут взаимодействовать с проектируемым

Review: Роли

- Author
- Moderator
- Reader
- Recorder

Review: Этапы



Review

- Entry criteria
- Exit criteria

Ambiguity Review

- это проверка требований, чтобы убедиться в том, что они написаны ясным, четким и недвусмысленным образом.

- осуществляется до анализа требований экспертом в предметной области.

Ambiguity review check list

- The dangling Else
- Ambiguity of reference
- Scope of action
- Omissions
- Ambiguous logical operators
- Negation
- Ambiguous statements
- Random organization
- Built-in assumptions
- Ambiguous precedence relationships
- Implicit cases
- Etc.
- I.E. versus E.G.
- Temporal ambiguity
- Boundary Ambiguity

Неоднозначные термины

- acceptable, adequate
- as much as practicable
- at least, at a minimum, not more than, not to exceed
- between
- depends on
- efficient
- fast, rapid
- flexible

Неоднозначные термины

- improved, better, faster, superior
- including, including but not limited to, and so on, etc., such as
- maximize, minimize, optimize
- normally, ideally
- optionally
- reasonable, when necessary, where appropriate

Неоднозначные термины

- robust
- seamless, transparent, graceful
- several
- shouldn't
- state-of-the-art
- sufficient
- support, enable
- user-friendly, simple, easy

Ambiguity Review : Example

V.1 The expert system should be able to recognize and identify potential critical situations.

It is not Unambiguous, Complete, Correct, Feasible

- *What does it mean "potential critical situations"?*
- *How can this be done?*
- *How can the system recognize and identify critical situations?*

It's unverifiable.

Ambiguity Review : Example

V.2 The expert system should keep a history of the values that it is reading at specific time intervals. If the values has risen considerably without any obvious reason during the last ten measurements, the expert system will inform it.

- What values should be gathered by the system?
- How deep should be the history?
- What does “specific time interval” mean exactly?
- What is “risen considerably”?
- What is “obvious reason”?
- Will? The word “will” identifies an ambiguity called a “Dangling Else”. The sentence with “will” in it tells us what is normally expected to happen
- It? Whom exactly and in what form?

Ambiguity Review : Example

V.3 The expert system should keep monthly history of the pressure and temperature, that are read every 60 seconds. The values and time of the measurement should to be stored in DB. If pressure has risen on more then 10 pts and variation of temperature is less then 3 pts, then the IPC system should display to the operator message “Warning: critical rising of pressure” and produce alarm signal. The warning message should be displayed until an operator presses button “close” (Sketch Warning.CritialPressure)

Problems in requirements

- The most of the configuration settings of IPS system shall be easily upgradeable in future versions

- **It is ambiguous**: How could we determine that it is "easily"
- **It's incomplete**:
  - What do "upgradeable" mean? How (in what way) should it be done?
  - Which exactly settings do "the most of configuration settings" include?
  - In what number of version this functionality will be necessary?
- **It's unverifiable**.

Problems in requirements

- The IPC system should not allow an operator functions, the consequences of which might be disastrous.

- **It's incomplete and unverifiable**:
- What does "disastrous" mean?
- "may be" is very fuzzy, "may be" but "may be not". How can be verified the functions the results of which might be disastrous?
- What does "*should not allow*" mean? Someone can assume that buttons for some functions will be disabled, another - that an operator will not see such functions, another one that the IPC system should produce alarm message.

Some more bugs

- Simplify the installation as much as possible by including dependencies in our installer where it is possible and makes sense.
- Follow standard Linux installations concepts where possible.
- Provide a suitable installation package for each support operating system.

Тестирование требований: Базовые подходы

1. Context free questions
2. Writing test cases
3. Models, Diagrams
   - Use-Case Diagram
   - Data Flow Diagram
   - Entity Relationships Diagram
   - State-Transition Diagram
   - Activity diagram
   - Class diagram
   - Decision Tables and Decision Trees

Определите критерии приема продукта

- Спросите пользователей, как они будут определять, отвечает ли ли продукт их потребностям и подходит ли для использования.
- Основывайте приемо-сдаточное тестирование на сценариях использования

Симптомы проблем с требованиями

- Перерасход бюджета
- Срыв сроков
- Продукт не удовлетворяет заказчика
- Постоянные переписывания кода
- Демотивация команды
- Упущенная рыночная ниша
- Неприятие продукта рынком

Преимущества качественных требований

- Уменьшение затрат на переделки
- Меньше ненужного кода
- Более быстрая разработка
- Уменьшение хаоса в проекте
- Довольные заказчики и команда

Спасибо!

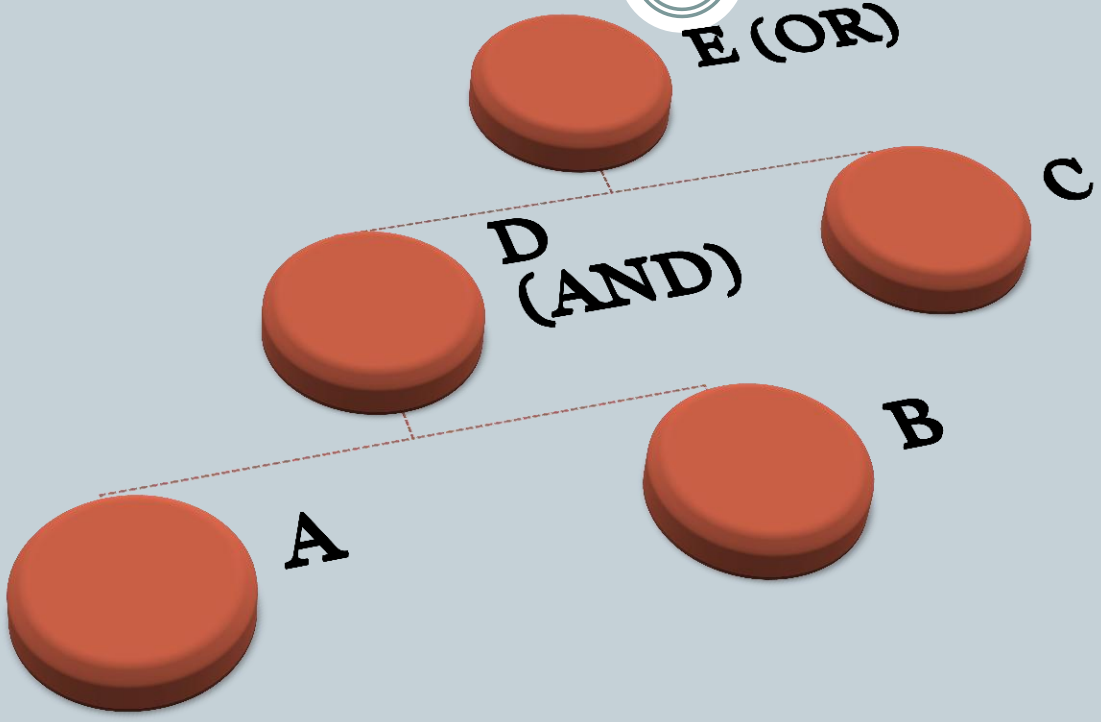
Вопросы?

Additional materials

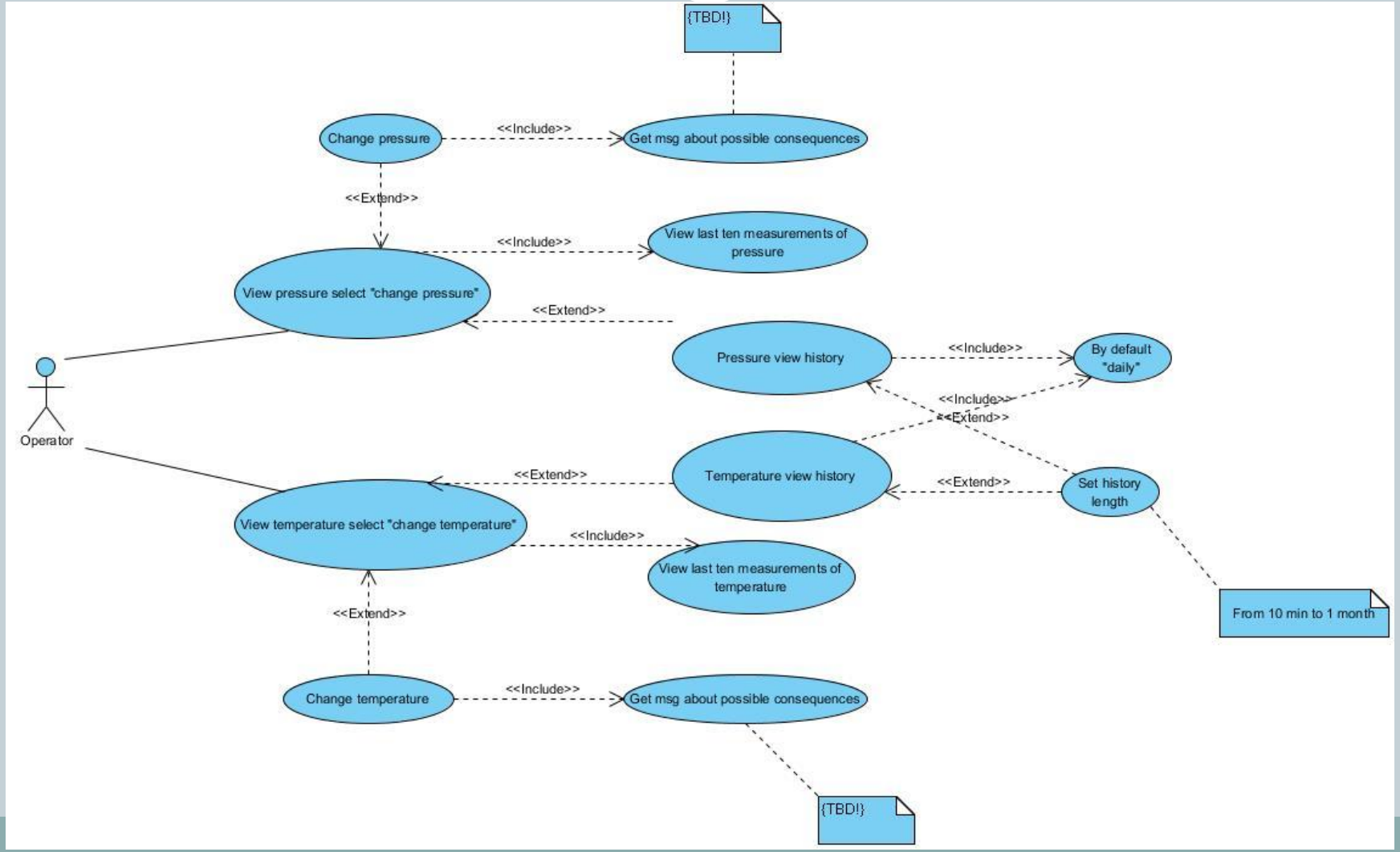
Cause-Effect graph

- The error message shall appear when the measurement on operator demand is not done, and there are problems with network or invalid sensor ID
- **Cause**
  - **A -** when the measurement on operator demand is not done
  - **B -** there are problems with network
  - **C -** or invalid sensor ID
- **Effect**
  - **E -** The error message shall appear

Cause-Effect graph



IPC Use-Case Example



Check List Example for Review (focused on Correctness)

- Do any requirements conflict with or duplicate other requirements?
- Is each requirement written in clear, concise, and unambiguous language?
- Is each requirement verifiable by testing, demonstration, review, or analysis?
- Is each requirement in scope for the project?
- Is each requirement free from content and grammatical errors?
- Can all the requirements be implemented within known constraints?
- Are all specified error messages unique and meaningful?