03 Quality Assurance

PowerPoint by Wanida Khamprapai

Introduction

- Software verification in general would mean verification of all the processes and phases to ensure that whatever is done till now and during the phase is good enough to build (develop) a quality application.
- This would typically mean a review of requirements, design, and code such that the code which is constructed will meet the expectations.
- Thorough review of requirements and design are also important because the cost of fixing defects related to requirements and design is very high if not detected and fixed in the same phase.

Requirement analysis

Requirement is a capability needed by a user to solve a problem or achieve an objective. It must be met or possessed by a system or system component. It could be to satisfy a contract, standard, specification, or other formally imposed document.

Having a complete and clear understanding of requirements is very important. An incomplete and wrong understanding of requirements can lead to an erroneous system.

Specification is a document that specifies, ideally in a complete, precise, and in verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied. [IEEE 610].

Requirement analysis

Challenges of requirements

There are many issues/challenges in the requirements gathered, as described below, which can lead to incomplete or wrong understanding.

Requirement not clearly defined

It is practically impossible to describe each requirement in detail. Business users also expect that people involved in development and testing have some level of understanding of the domain and standard operating practices within the industry.

Communication issues

There are three basic communication issues that can result in incomplete/wrong understanding of the requirements

- Omissions – Sender misses to communicate or receiver misses to listen even if sender communicates.
- Untold assumptions – The provider of the requirements and the receiver of the requirements both make some wrong assumptions even if some aspects are not discussed. These assumptions, if not right, may result in bugs.
- Misunderstanding – Everyone has a different level of knowledge and background and is likely to misunderstand what is provided.

Criticality, Priority not provided

A clear understanding of the criticality or priority of the requirements helps in such situations.

Requirement analysis

Requirement verification, Ambiguity reviews

The primary objective of the Requirement review process is to evaluate correctness, completeness, accuracy, consistency, testability and clarity of requirements.

Requirement Ambiguities: A requirement is considered ambiguous if it can create misunderstanding or different understanding among different people.

Requirement analysis

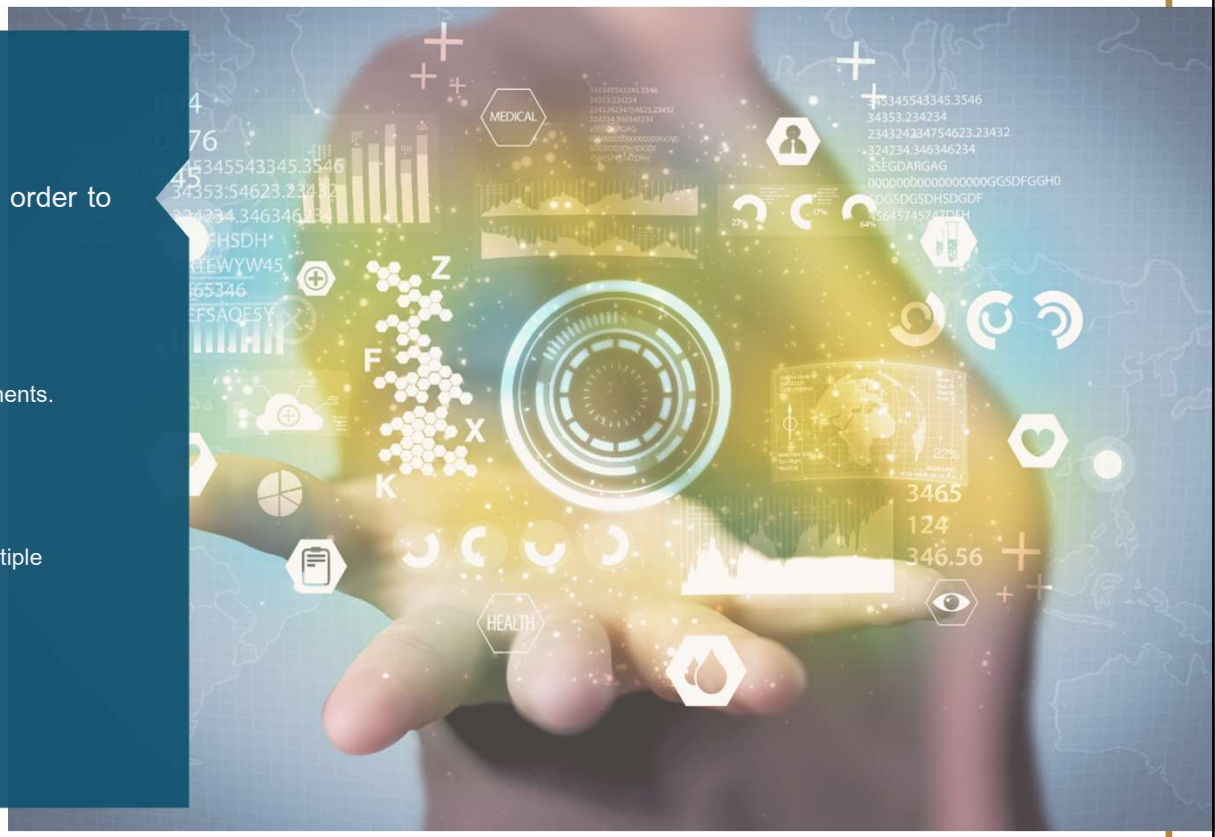
Understanding what is causing the ambiguity is necessary in order to remove the ambiguity and write requirements that are clear.

Sloppiness

Absence of due or proper care or attention while providing requirements.

Linguistic Ambiguity

a quality of language that makes speech or written text open to multiple interpretations.



Requirement analysis

Understanding what is causing the ambiguity is necessary in order to remove the ambiguity and write requirements that are clear. (Cont.)

Dangling Else

The requirements which uses any of MUST BE, WILL BE, IS ONE OF, SHOULD BE, COULD BE, CAN BE, SHALL will result in ambiguity if there is no clarity on what should be done if this is not true (Else part is not clear)

Ambiguity of reference

Many times, some statements are used which refer to some details provided earlier in the document. If the reference is not explicit, the understanding can be wrong.

Omissions

Applications generally produce some output (effects) based on some type of inputs (causes). However, it is likely that one of the causes or effects may be missing.

Requirement analysis

Many times, it may not be easy to ensure coverage and find all ambiguities. Following guidelines can help for the same:

Ensure requirement coverage by understanding the following aspects. You my need to ask many questions to get complete clarity.

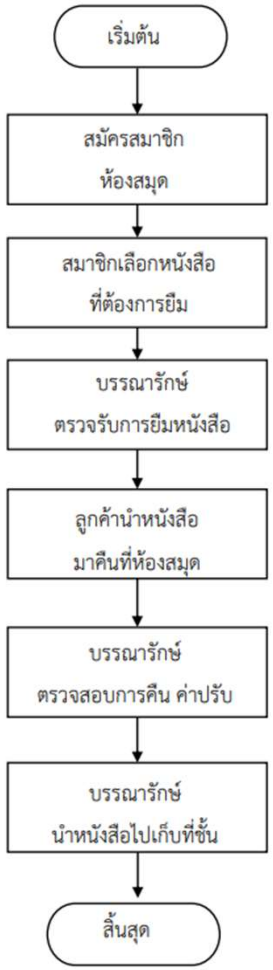
Itemize the requirements: Divide complex requirements into multiple simple requirements. This will not only help in identifying and clarifying ambiguities but will also help in translating them into solutions and test cases.

Verify the gathered requirements with the customer

- through review meetings
- Summarize and present the requirements as understood by you to the user/stakeholder
- Carry out a reverse walk through to eliminate bad or wrong understanding
- Clarify any unclear requirements or conflicts

Requirement

analysis



คำอธิบายขั้นตอนการทำงานระบบ

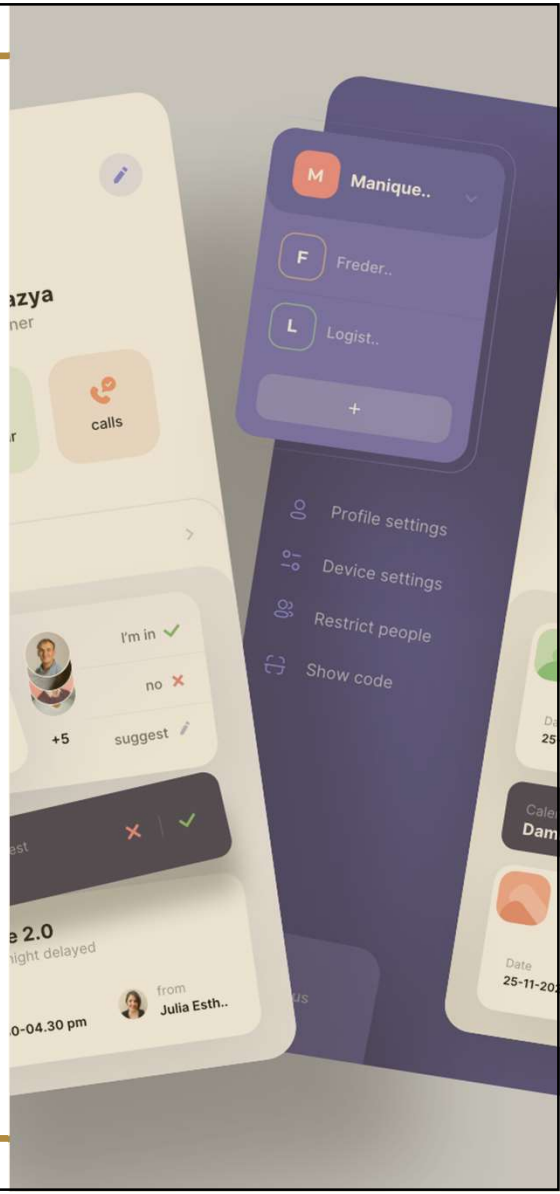
1) นักเรียนสมัครสมาชิกห้องสมุดเพื่อทำการยืมหนังสือ
2) เมื่อเป็นสมาชิกแล้ว จะสามารถเลือกหนังสือที่ต้องการยืม นำให้บรรณารักษ์ การยืมแต่ละครั้งจะยืมได้จำนวน 7 วัน หากเกินกำหนดจะคิดค่าปรับวันละ 5 บาท/เล่ม
3) บรรณารักษ์ทำการตรวจรับการยืมหนังสือเพื่อออกใบยืมหนังสือให้สมาชิก
4) เมื่อถึงวันกำหนดคืน สมาชิกจะนำหนังสือมาคืนให้กับบรรณารักษ์
5) บรรณารักษ์ทำการตรวจรับการคืนหนังสือพร้อมคำนวณหากมีค่าปรับเพิ่มเติม
6) บรรณารักษ์นำหนังสือกับไปที่ชั้นหนังสือตามเดิม

Design Review

Reviews of functional and architectural design, component design, database design, user interface design, and any other such design documents can be prevented defects/errors in software development and implementation.

In general, design is examined to make sure it

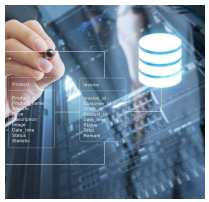
- Implements all the explicit and implicit requirements
- Provides a complete picture of the software addressing data, functional, and behavioral domains from an implementation perspective
- It is logically partitioned into subsystems and modules
- Contains distinct representations of data, architecture, interfaces, and components that
- Leads to appropriate data structures
- Leads to components that exhibit independent functional characteristics
- Leads to interfaces that reduce the complexity between components and external subsystems
- Is readable, and understandable, and is represented using notations that effectively communicate its meaning – database design and user interface design.



Design Review

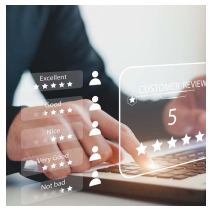
Database design reviews

The quality of the data that is entered and stored in the system has a significant impact on its quality. Data quality can be classified into



Data consistency ความสอดคล้องของข้อมูล

Ensuring that data is represented “semantically the same way” within and across tables in a database system.



Data completeness ความสมบูรณ์ของข้อมูล

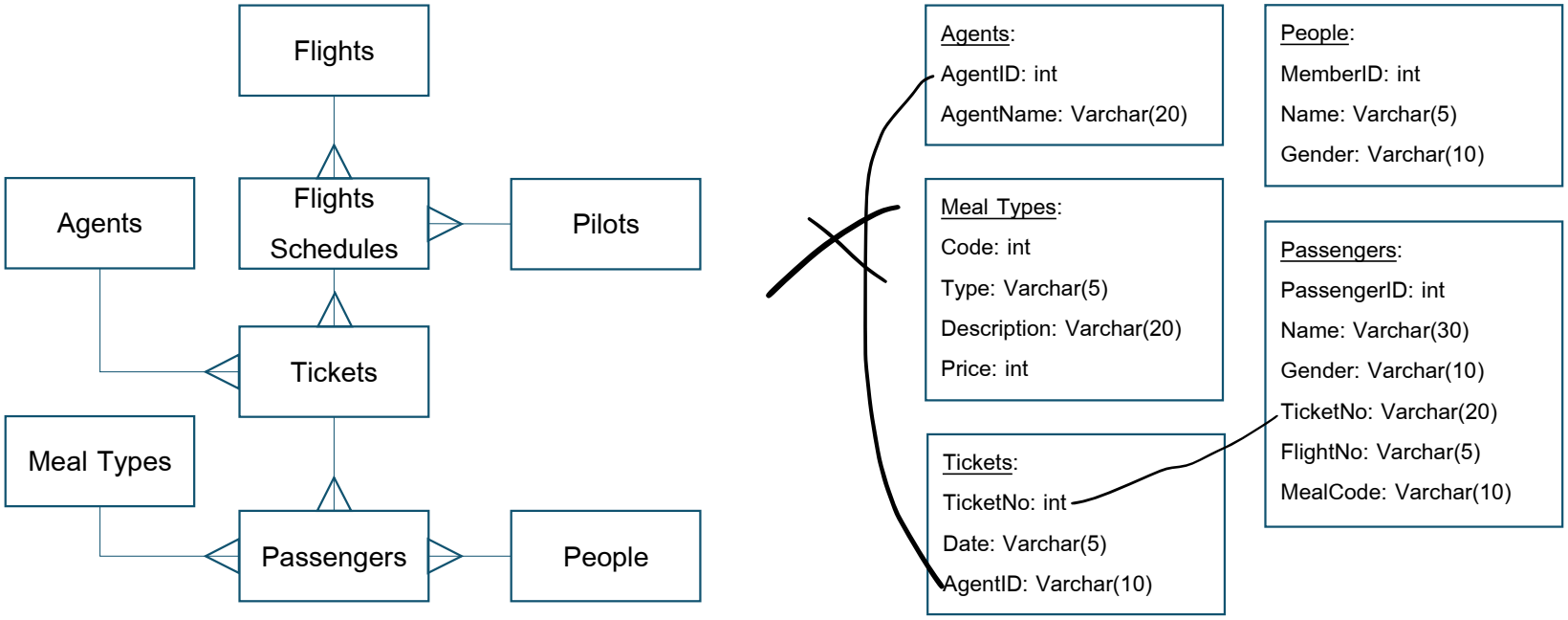
Ensuring that there is no missing or inaccessible data because of inadvertently storing nulls, and blanks and there is no partial data due to truncations.

Data correctness ความถูกต้องของข้อมูล

Ensuring that corrupt or wrong data is not stored; as well as dirty data is not being shown to the user.

Design Review

Database design reviews



Design Review

User interface design reviews

The user interface is the point at which human users interact with a computer, website or application. The goal of effective UI is to make the user's experience easy and intuitive, requiring minimum effort on the user's part to receive the maximum desired outcome.

Every user of the application should feel good while using the application and hence User Interface of the application should be

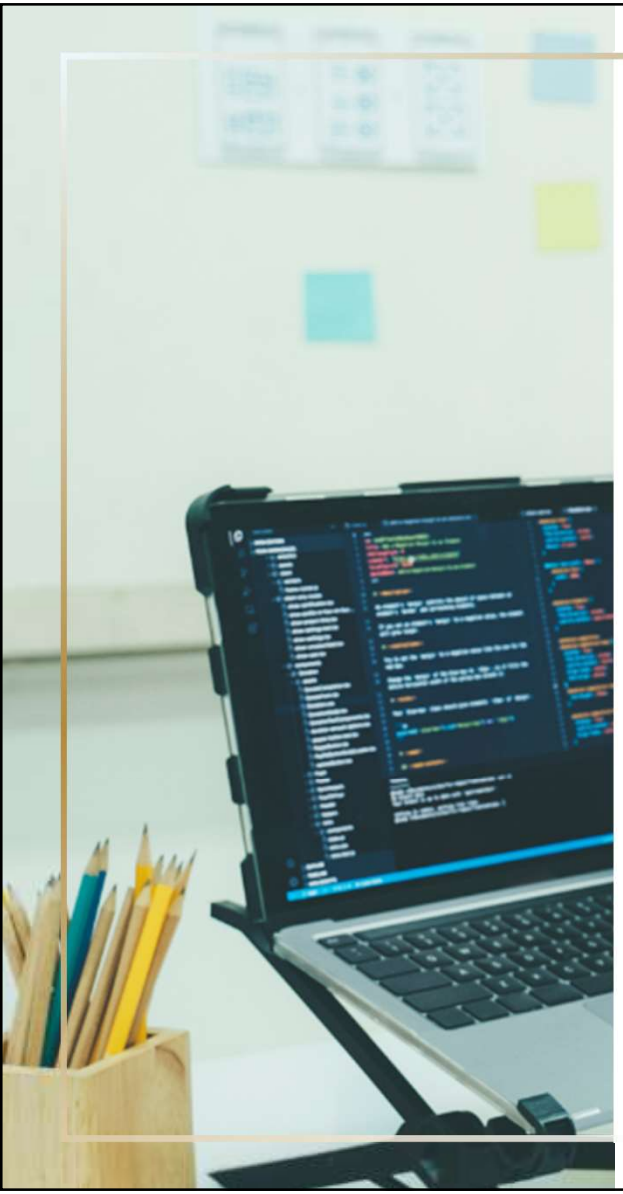
- Simple, readable and easy to understand,
- easy to navigate
- aesthetically rich
- Make optimal use of screen size and resolution

Some important aspects of the user interface designs include

- Position of Title, Menu options, and standard buttons
- Alignment of fields/ objects.
- Height and width of text boxes or any other controls
- Font color and size of all labels, entered text, error messages
- Provision to distinguish frequently used options
- Mandatory input fields can be distinguished

Design Review

User interface design reviews



Code Review

Primary objective of code review is to ensure the correctness, understandability, and maintainability of the code developed.

```
#include<stdio.h>
#include <math.h>
int cal_amt (char ttype, int booking_amt) {
    float gst_rt=.05;
    bill_amt = 0;
    if (ttype = “S”)
        if (booking_amt > 1000)
            discount = booking_amt * .10;
        else if (booking_amt >= 500)
            discount = booking_amt * .05;
        bill_amt=booking_amt - discount * gst_rt;
    return bill_amt;
}
```

```
int main()  {
    char ttype;
    int booking_amt;
    printf("Enter Traveller Type:");
    scanf("%c", &Ttype);
    printf("Booking Amount:");
    scanf("%f", &booking_amt);
    final_amt = cal_amt (booking_amt, ttype);
    printf("Final Amount = %f",final_amt);
    return 0;
}
```

Code analysis

Developers make some general mistakes resulting in some standard errors as described below.

Data declaration errors

- Have all variables been explicitly declared? Meaningful? Is the case sensitive?
- Are variables/Pointers properly initialized in declaration sections?
- Are variables assigned with the correct length, type, and storage class? E.g. Should a variable be declared as a string instead of an array of characters?
- Are there variables declared which are never used in the program?

Comparison errors

- Are there any comparisons between variables having inconsistent data types?
- Ensure comparison operators are used correctly - </<=, >/>=
- Ensure that there is no precision problem when fractional or floating point values are compared.

Data reference errors

- Is a variable referenced whose value is unset or uninitialized? (e.g. Two variables with similar names and the wrong variable is used)
- Usage of array subscripts – Are they integers and values within the boundary? In Some languages, it starts from 0, and in others, it starts from 1
- Is there any variable where a constant can be used?
- Is there any variable that is assigned a value of a different type? E.g. Assigning a floating point number to an integer variable can cause an issue
- Is data structure defined identically in all functions where the same data structure is referenced?
- Is the usage of similar-looking operators (=/==, &/&&) proper?

Code analysis

Developers make some general mistakes resulting in some standard errors as described below (Cont.).

Computation errors

- Is there any computation using variables having inconsistent data types?
- Is the target variable (on the left side of the = sign of an expression) of smaller size than the right-hand side expression?
- Is there any possibility of division by 0?
- Is there any possibility of a loss of precision – e.g. in integer arithmetic?
- Is there a possibility where computed variables assume value outside the range? E.g. If the variable represents% ensure it remains between 0% and 100%
- In a compound/complex expression, check whether the order of evaluation and operator precedence is correct

Control Flow errors

- Are the statement blocks within a control structure proper? (Are Begin and End of blocks correctly used?/ Parentheses correctly used?)
- Will the loop eventually terminate?
- Is the loop executed at least once? If not, is it really, correct?
- If the program contains multi-way branch statements (e.g. switch...case), can the index variable exceed the total possible numbers? If so, is it handled properly?
- Delimitation of parts in if then else

Interface errors

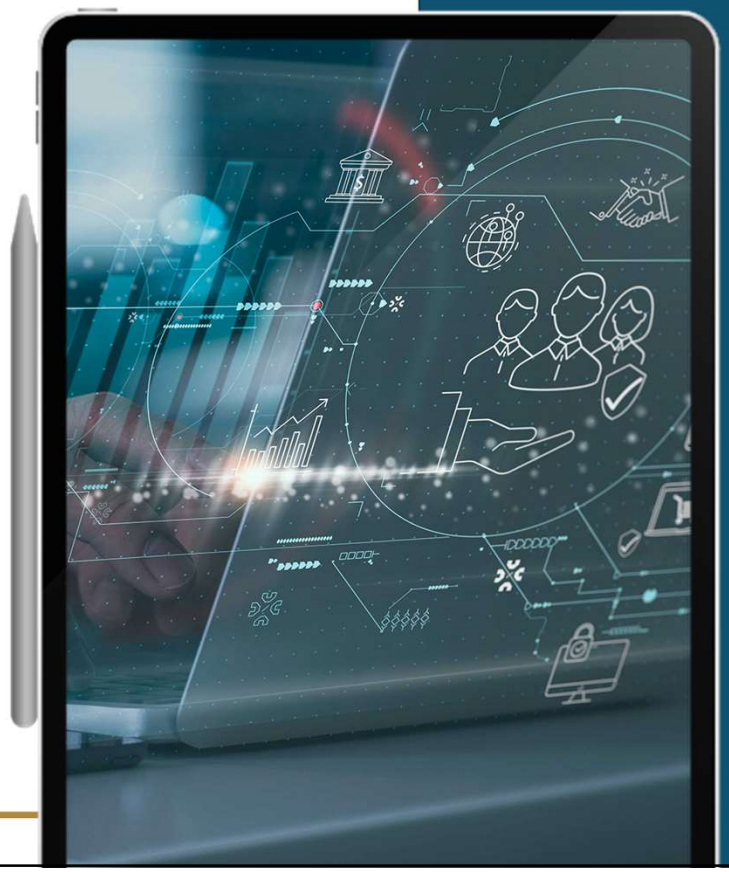
- Does the number of parameters received by the module equal the number of arguments sent by calling modules?
- Also is the order correct?
- Are the sizes and type matching?

Static testing techniques

Testing of a component or system at specification or implementation level without execution of that software.

It has been observed that effective static testing can find 30-70% of defects. It is also important because it can find some types of errors which cannot be found by dynamic testing by executing the program such as

- Wrong understanding/ Missing Requirements
- Design Defects
- Non-maintainability of code
- Inconsistent interface specifications
- Dead codes
- Unused variables
- Uninitialized variables
- Standard violations
- Infinite loops



Static testing techniques

Informal

In informal review, the document designer place the contents in front of viewers, and everyone gives their view; therefore, bugs are acknowledged in the early stage.

Walkthrough

Generally, the walkthrough review is used to performed by a skilled person or expert to verify the bugs. Therefore, there might not be problem in the development or testing phase.

Peer review

In peer review, we can check one another's documents to find and resolve the bugs, which is generally done in a team.

Inspection

The inspection is essentially verifying the document by the higher authority, for example, the verification of SRS [software requirement specifications] document.

Questions & Answers