

## ACCEPTANCE TESTING

### (UAT Execution & Reports submission)

Date	14 NOVEMBER 2022
Team ID	PNT2022TMID50000
Project Name	Real-time River water quality monitoring and control system
Maximum Marks	4 Mark

### Introduction

Effectively documenting incidents during the testing process is the key to improving software or processes before a system is released. Sometimes, the testers themselves document issues they encounter; but more often, a UAT coordinator verifies, consolidates, and classifies reported issues before assigning them to the appropriate group to address. Then, that IT coordinator again validates and prioritizes the technical issues before handing them off to an IT developer to investigate further and resolve.

During the course of UAT, it is inevitable that issues will be discovered. It is shocking how often documented issues contain insufficient data to facilitate a quick and thorough investigation.

### Deliverables of UAT

Every interviewer very quickly stated that UAT is to assure quality. Project managers also stated that it can double as a training exercise for business users as well as ensuring that the requirements set match the functionality that is desired from the system. People managers expressed that one of the most important deliverables is the decision to go forward with the update or new system; the "green" or "red" light. Individual contributors expressed that UAT and inclusion goes hand in hand. That the testers feel included in the development and actually have a say in what works and what doesn't. Individual contributors stated that they felt that UAT has been done enough when the tests they are running are all success full but that it is a gut- feeling or intuition that says when they are content with the testing. They alsostated many perks of UAT such as: learning the new system, cooperationbetween departments, learning something new, feeling valued by the company and inclusion in decision making. Project Managers statedthat the organization at large sometimes acted asthough it had forgotten the purpose of UAT - to assure quality and usability of a release.



## Data Mining

This section represents the actions pillar of the research. Here results based on empirical insights from the system log files are presented. Results from the qualitative review of the testers use of test management tools are also presented in this section.

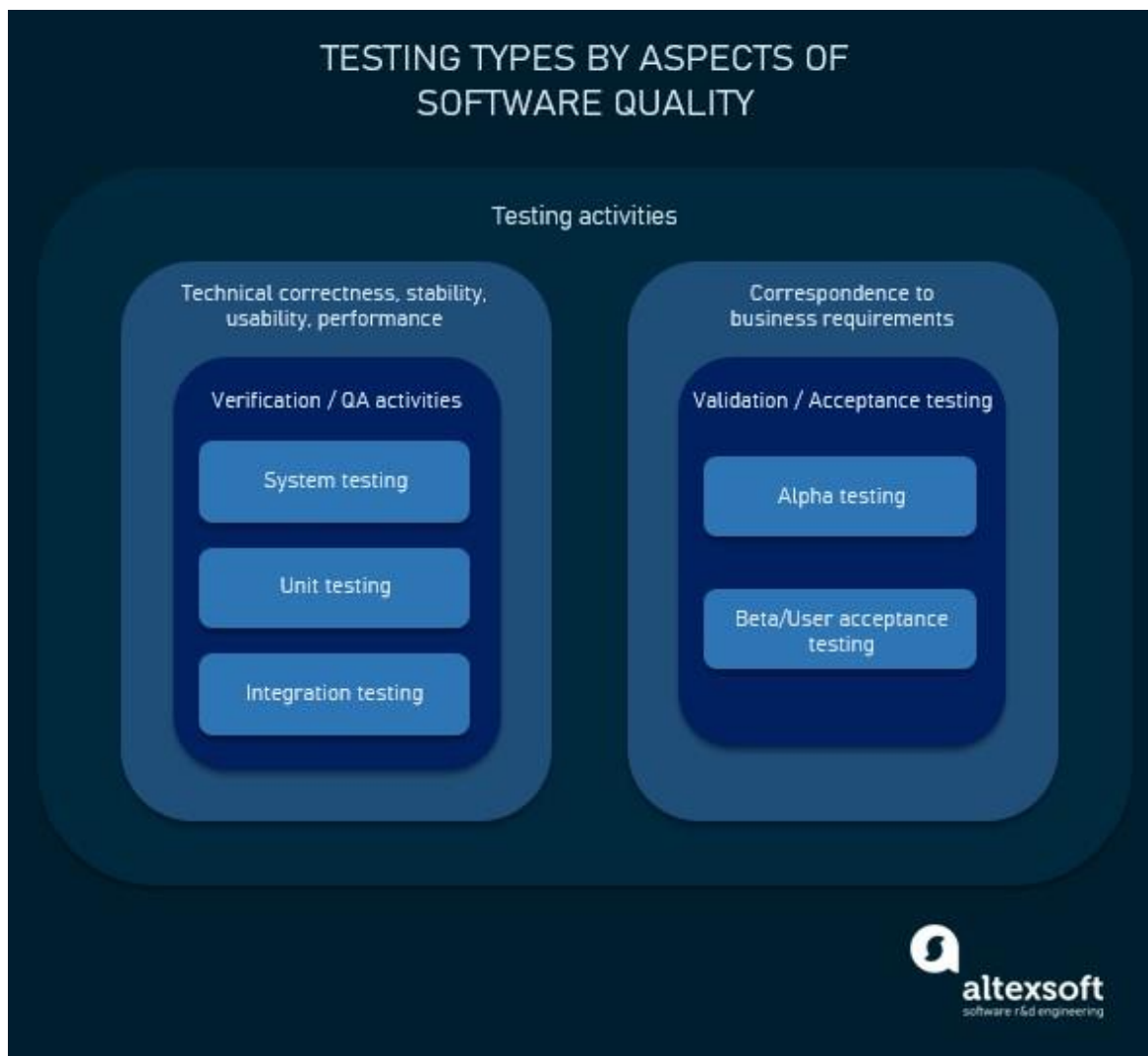
## Time spent on Testing

Because access was granted to the SUTs application logs it was possible to track exactly how much time users spent on testing functionality in the system. In the blow table 4.3.1 average times are detailed some of the users from the SUT. Table 4.3.1: Overview of the operative time release, i.e. one execution of the test plan.

Role	Operation Time
Employee 1	3 hours 21 min
Employee 2	1 hour 32 min
Employee 3	0 hours 49 min
Employee 4	1 hour 45 min
Employee 5	3 hours 04 min

## Test Quality

From the production logs of the SUT a Markov-chain with 68 states (one for each application feature that was left after filtering out non-relevant states) was created. Due to the fact that the SUT was a regular release of an existing system, and not a newly adopted software, a transition matrix could be made on a per-tester level for both the production system logs, as well as the test system logs. Variability due to changes in logging were taken into account by qualitatively examining the log files. As transition matrices for both TEST and PROD had been computed, a similarity score could be computed to directly and in bulk estimate the quality of the testing.

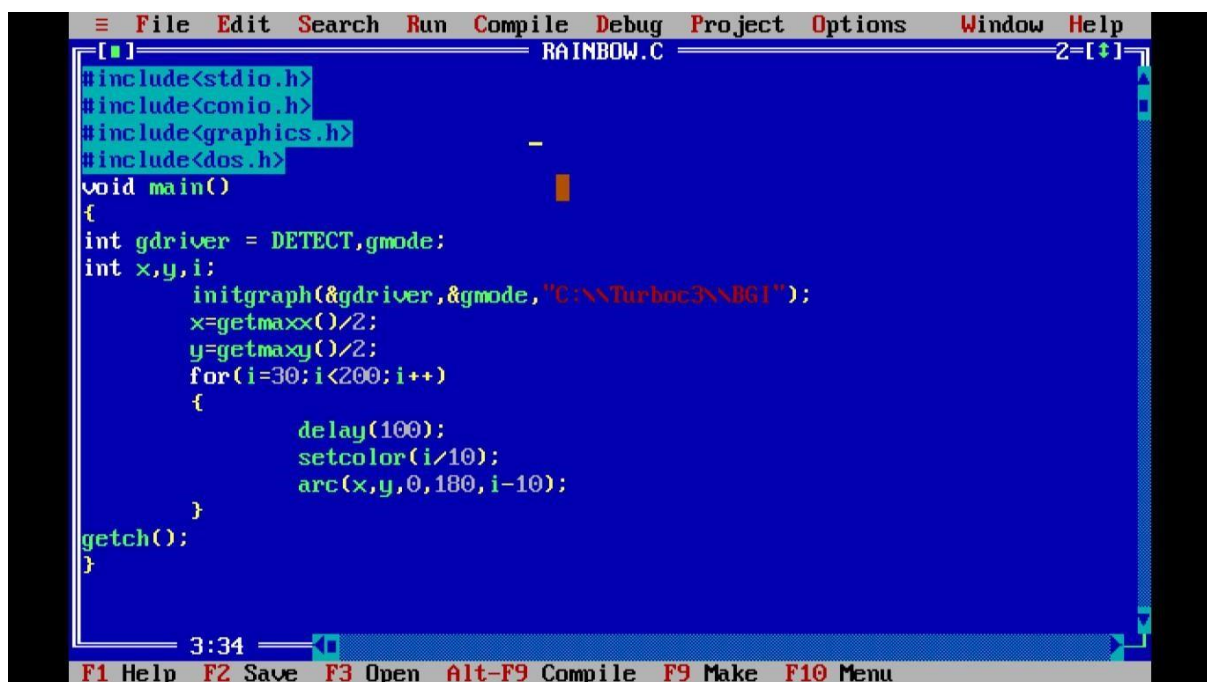


## Execution Flow

\* C program (source code) is sent to preprocessor first.

\* Expanded source code is sent to compiler which compiles the code and converts it into assembly code.

\* The assembly code is sent to assembler which assembles the code and converts it into object code.



```
File Edit Search Run Compile Debug Project Options Window Help
[ ] RAINBOW.C 2-[ ]
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void main()
{
    int gdriver = DETECT, gmode;
    int x, y, i;
    initgraph(&gdriver, &gmode, "C:\\\\Turbo3\\\\BGI");
    x = getmaxx() / 2;
    y = getmaxy() / 2;
    for(i = 30; i < 200; i++)
    {
        delay(100);
        setcolor(i / 10);
        arc(x, y, 0, 180, i - 10);
    }
    getch();
}
```

3:34

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Usually, when possible, this testing happens in a conference or a war room sort of a set up where the users, PM, QA team representatives all sit together for a day or two and work through all the acceptance test cases.

Once all the tests are run and the results are in hand, the **Acceptance Decision** is made. This is also called the **Go/No-Go decision**. If the users are satisfied it's a Go, or else it's a No-go.

Reaching the acceptance decision is typically the end of this phase.

## Conclusion

UAT is not about the pages, fields or buttons. The underlying assumption even before this test begins is that all that basic stuff is tested and is working fine. God forbid, the users find a bug as basic as that – it is a piece of very bad news for the QA team.

This testing is about the entity that is the primary element in the business.

```
class Program
{
    0 references
    static async Task Main(string[] args)
    {
        WriteLine("Please type the username for the desired user:");
        var username = ReadLine();

        var github = new GitHubClient(new ProductHeaderValue("MyAmazingApp"));

        try
        {
            var user = await github.User.Get(username);
            WriteLine($"The user {user.Name} was succesfully retrieved!");
            WriteLine($"The user {user.Name} has {user.PublicRepos} public repositories. Do you want to see the list? (y/n)");
            var response = ReadLine();

            if (string.Equals(
                "y",
                response,
                StringComparison.InvariantCultureIgnoreCase))
            {
                var repos = await github.Repository.GetAllForUser(username);
                foreach (var repo in repos.OrderBy(x => x.CreatedAt))
                {
                    WriteLine($"{repo.CreatedAt:yyyy-MM-dd} | {repo.Name}");
                }
            }
        }
    }
}
```