1. Introduction

The Importance of properly selecting Testing techniques is widely accepted in software engineering community. One of the hobbyhorses of software systems development has been and still is the construction of quality products that do not exceed the originality budgeted price and delivered according to a pre-established schedule [Blum, 1992]. The endemic impossibility of achieving this objective was historically baptized as software crisis [Naur and Randell, 1969] and mainly strikes complex projects. This problem become apparent in the 1960s, when computer started to become popular. Far from disappearing, the problem has become yet more serious, as the computer industry has continued to expand and the market demanded increasingly more complex systems. Quality control and assurance activities not only effect the code, but also each and every one of intermediate products generated during software construction.

Several empirical studies have been carried out with college students as subjects in the last few years. Researchers often use these studies to pilot experiments before they are carried out in industrial environments. Empirical studies in software engineering are often used to gain insight into new techniques and methods.

Empirical studies may have different types of goals, including assessing the strength or Weakness of products and processes, highlighting areas for improvements, and evaluating software engineering techniques.

1.1. Software Testing

Computer users want software that is easy to use, helpful for them and flawless with their work. To develop such kind of software is not an easy task in real time world. There are lots of failed projects in the world that had started with enthusiastic ideas but they stumbled from some reason. Very Often reason was that they
Software testing is an indissoluble component of software products and their development processes. To be successful in software testing we have to understand the various parts of Software testing. Testing is very important and helpful during development process but many organizations have problems to manage and perform it to gain profit. Software testing requires being well prepared and executed according to some best practices from quality assurance and their domains.

In first part, this work summarizes information and knowledge necessary to understand quality assurance, Quality Control and software testing domains. List of some important definitions is focused on principles of Software Testing and does not provide too specific and detailed information. There is also demonstration how testing is Very important in the software development life cycle.

In next part there is description of situation about quality assurance and software testing from IT organizations’ perspective. Short description about possible approaches how to perform testing without testers is present and also problems that organizations meet during their attempts to test software. We will discuss some software testing education strategies by the active learning from IT industry.

1.2. Testing Phase

The testing phase is an important and critical part of software development, consuming even more than half of the effort required for producing deliverable software.

In this Thesis we will present different methodologies and approaches to manage and control the testing phase under different
complementary aspects. In particular, we are interested in what it is observable by testing, i.e., the failures.

1.2.1. Static Techniques
The static techniques are based on the examination of the project documentation, the software and other related information about requirements and design and not on software execution. The static techniques include software inspection, reviews, code reading, algorithm analysis and tracing. Thus the use of static techniques is not limited to the testing phase; their application during the entire development phase is even more important.

1.2.2. Definition of Software Testing
Software testing is a complementary approach of the static techniques described which involves the Execution of the code. The term “dynamic” means precisely that the “testing always implies executing the program on (valued) inputs” in a specific environment. Principally for the non-deterministic systems, the results obtained by testing depend strictly on the input provided as well as state of the system. Therefore when speaking about input values the definition of the parameters and environmental conditions characteristic of a specific system state must be included when necessary.

1.3 Testing Level
Generally the testing is performed at different levels during the development process and can involve the whole system or parts of it. Here we distinguish three different stages: unit, integration and system test providing in the following a brief description of each. It is important to clarify that no stage is more important than another. Each one has its specific target and difficulties and only a good combination of them can provide products of quality.
### Unit Test

A unit is the smallest testable piece of software, consisting of hundreds or a few lines of source code, and generally representing the result of the work of one programmer. The Unit test’s purpose is to ensure that the unit satisfies its functional specification and/or that its implemented structure matches the intended design structure. When the tests reveal an anomalous behavior, it is said that there is a unit bug.

### Integration Test

Integration is a process by which components are aggregated to create a larger component. Even though the single components are individually acceptable when tested in isolation, they could result incorrect or inconsistent when combined in order to build complex systems.

### System Test

System test involves the whole system embedded in its actual hardware environment and is mainly aimed to verify that the system behaves according to the requirements document. In particular it attempt to reveal bugs that cannot be attributed to components as such, to the inconsistencies between components, or to the planned interactions of components and other objects.

### 1.3.1 Objectives of Testing

The Software Testing can be applied for different purposes, such as verifying that the functional specifications are implemented correctly or that the system shows specific reliability.

Acceptance testing is the final test action prior to deploying the
software. Its goal is to verify that the software respects the customer’s requirement, i.e., it can be used by the end-users to perform those functions and tasks the software was built to do.

Alpha testing before releasing the system it is given to the in-house user for exploring the functions and business tasks. Generally there is no test plan to follow; the individual tester determines what to do.

Beta Testing the same as Alpha testing but the system is given to external users. In this case the amount of detail, the data, and approach taken is entirely up to the individual tester. Each tester is responsible for creating their own environment, selecting their data, and determining what functions, features, or tasks to explore. Each tester is responsible for identifying their own criteria for whether to accept the system in its current state or not. Beta testing is thus the less controlled phase.

Reliability achievement: testing is a means to improve reliability; therefore the test case must be randomly generated according to the operational profile.

Functional Testing: Tests focused on validating whether the observed behavior of the tested system conforms to the specification. In particular it checks whether the functions are as intended and provides required service(s) and method(s). This test is implemented and executed against different tests targets, including units, integrated units, and systems.

1.4. Software Engineering Education
The computer science education community has been active for more than three decades.
A growing body of literature provides general guidance for effective software engineering Education.

For example:

• The International conference on software engineering (ICSE) has devoted a session to Education & Training since 1994.

• The ACM Journal on Educational Resources in Computing (started in 2001) is an example of a journal devoted to computer science education. In an article devoted to computing curricula, a task force agrees on the importance of a significant team project encompassing both design and implementation and highlights the possibilities to work with local companies to allow students to engage in projects in a professional setting.

• IEEE Software recently devoted a special issue to the education. In this issue, there are articles about a framework for teaching software project courses, an approach to teaching in a simulated professional environment, and educating software engineers from a professional point of view, including the benefits of interaction between Professionals and academia. In addition, the editors, Hilburn and Humphrey, include a list of fundamental software engineering education literature.

1.4.1 Benefits and Cost for the Researchers

The benefits that researchers gain from empirical studies with students are to:

• Obtain preliminary evidence to confirm or refute hypotheses.

• Control factors that may affect the study.

• Show software companies the relevance of the research.

• Show software companies the usefulness of carrying out empirical studies in their own environments.

• Show software companies the feasibility of carrying out full-fledged empirical studies in industrial environments.
1.5. Benefits and Cost for the Students
We believe that empirical studies in software engineering classes will also provide the following benefits to the students:

- Education on state-of-the-art topics.
- Industrial relevance.
- Hands-on practice.
- Empirical methods.
- Third party assessment

1.6 Benefits and Cost for the Industry

- Obtaining preliminary evidence to confirm or refute hypotheses about new technology or method adoption.
- Obtaining preliminary evidence on methods and technology they have already adopted. Getting ideas about new product development.
- Learning about new software methods.
- Obtaining knowledge of empirical software engineering.

1.7 Testing Process
A process has to be followed to satisfactorily achieve the objective pursued by software testing. Objective achievement will depend on how good this process is.

There is sometimes some confusion as to the testing process and it is mistakenly thought that a testing techniques output faults, when it really outputs test cases.

Test planning, Test design; Test case specification, Test procedure definition, Test procedure execution and analysis of results are the activities during testing. We need to focus onto the Test design as it is critical step in Testing process.
1.7.1. **Most Common Software Problems**
- Incorrect calculation.
- Incorrect data edits & ineffective data edits.
- Incorrect matching and merging of data.
- Data searches that yields incorrect results.
- Incorrect processing of data relations.
- Incorrect coding / implementation of business rules.
- Inadequate software performance.
- Confusing or misleading data.
- Software usability by end users & Obsolete Software
- Inconsistent processing
- Unreliable results or performance
- Inadequate support of business needs
- Incorrect or inadequate interfaces with other systems.
- Inadequate performance and security controls.
- Incorrect file handling.

1.7.2. **Objectives of Testing**
- Executing a program with the intent of finding an error.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- To check if the system is — fit for purpose.
- To check if the system does what it is expected to do.
- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers a yet undiscovered error.
- A good test is not redundant.
- A good test should be — best of breed.
- A good test should neither be too simple nor too complex.
1.7.3. Objectives of a Software Tester

- Find bugs as early as possible and make sure they get fixed.
- To understand the application well.
- Study the functionality in detail to find where the bugs are likely to occur.
- Study the code to ensure that each and every line of code is tested.
- Create test cases in such a way that testing is done to uncover the hidden bugs and also ensure that the software is usable and reliable.

1.8. What IT Says About the Product

- Correctness
- Does it accurately describe the program? Controversy
- Which parts are controversial? Who are the stakeholders who disagree and why do they disagree? Adequacy
- Does it provide enough information for programming, documentation and testing? Completeness
- Does it cover the feature set? Design
- Can you tell whether it specifies design errors?
- Is it understandable, usable, trainable, consistent, and appropriate for the market?
- Does it set up the program / programmer for common errors?

1.8.1. What IT Says About Testing

Early in the project, you can review the spec’s implications for testing, and change them or prepare for them.

- Implications for test design what test techniques will be most appropriate for this project? Will you need additional training or
tools for them? Are there ways to simplify (or otherwise change) to product in ways that would call for simpler or cheaper or more easily structured techniques? How much exploring will this project require?

- Does your staff have the knowledge, skills and connections?
- Test schedule and resource commitments / implications when will you receive deliverables from others? When are you to deliver your work? What do you need to get this done? Are any of your commitments unreasonable?
- Testability support

1.9. Summary

Objectives

- Check the product’s conformance with every Statement in every spec, requirements document, etc.
- The specific objectives are context-dependent Why did they write the spec? Why are you testing against it? Who cares about your results? Paradigmatic case(s)
- Testing against contractual specifications
- Testing dominated by traceability to written specifications
- User documentation testing Strengths
- Critical defense against warranty claims, fraud charges
- Effective for managing scope / expectations of regulatory-driven testing
- Reduces support costs / customer complaints by reducing risk of misrepresentations to customers. Risks
- Issues not in the specs or treated badly in the specs / documentation.
- Focus is on coverage, not risk
- Test case management tools that lock onto traceability may harm your efficiency