International Journal of Service Science, Management, Engineering, and Technology
Volume 17 • Issue 1 • January-December 2026

Experimental Test and Run Using Equivalence Partitioning and Control-Limit Techniques for Mobile Software

Nalinee Sophatsathit
https://orcid.org/0009-0009-6196-2186
Suan Sunandha Rajabhat University, Thailand
Received: September 16th, 2025 | **Accepted:** December 28th, 2025

ABSTRACT

Mobile software development turns around quickly, which induces inadequate testing to meet this short-lived pace of development and release process. The traditional testing procedures just do not fit such a rapid mobile paradigm. This study sets out to exploit two straightforward techniques, namely equivalence partitioning and control-limit techniques, based on software cost and quality. The first technique divides input domain into proper partitions that suit the software functions and selects cost representative data from each partition for testing. The second technique keeps the outcomes within the limits to ensure acceptable output. The contributions of these techniques prove to be reliable steppingstones for mobile software development without losing sight of the founding methodologies. The study envisions that future work should incorporate well-established techniques for developers to set up rapid test-and-run procedures, whereby quality mobile software can be developed from this fast paced and short-lived cycle.

KEYWORDS

Equivalence Partitioning, Control-Limit, Software Testing, Mobile Software Development

INTRODUCTION

Traditional software development has been ongoing for decades and, in essence, has become a well-established process over the years. The advent of mobile applications has brought about new development paradigms that change the game entirely. Perhaps the key factor of this change is time. The fact that most mobile apps are small, short-lived, and abandoned by users when new comparable apps arrive makes it unsuitable to maintain them in the same fashion as the traditional software counterparts. One area to look into is the testing stage of the development process. The work processes performed by developers (during unit tests, integration tests, combined test-run during system tests, user acceptance tests, and the remaining stress tests, regression tests, etc.), are all time-consuming processes that can take months to complete before the software is released. This is one thing mobile apps cannot embrace in their development process because this process may not exceed 6–9 months from start to release. Hence, it is obvious that, to abide by the typical testing process, the procedures must be adjusted and arranged to fit this time constraint.

In principle, the test method involves many manual procedures that are time-consuming. Many traditional test procedures are straightforward to use yet may not be suitable in some test scenarios

DOI: 10.4018/IJSSMET.399173

and environments. For example, in the mobile environment, users are anonymous, which makes it hard to set up proper coverage of input data partitions manually (let alone performing the analyses of user acceptance tests, test efficiency, cost, etc.). Conversely, this may inevitably lengthen the mobile development life cycle.

To reduce this development life cycle, this study focused on shortening the test-and-run process. The unit, integration, and system tests were combined into one test-and-run step whose procedures were governed by two well practiced techniques in traditional software testing: equivalence partitioning and control-limit techniques. These two techniques were chosen because they are straightforward to understand, apply, and verify since they can be observed, performed, and examined visually both by the developers and users, making the agile team work smoothly.

The first technique arranges input data in groups whose members are said to be equivalent, that is, each datum can represent the group. The number of groups, or partitions, confines the size of input data sets to be used in each test step. Thus, exhaustive testing does not constitute an issue. The second technique sets up an interval to keep test outputs within the predefined range, having lower and upper bounds. This scope is known as the control limit, which is a well-known statistical technique. Should any of the test outputs fall outside this predefined range, a procedural correction is called for to rectify the corresponding software functions. In so doing, the development process can be sped up considerably.

The above two techniques offer developers the flexibility of data selection and representation within a defined operational range. They are simple and straightforward to implement based on well-defined theoretical fundaments. It will be evident how the development progresses as the proposed methodology unfolds.

This paper is organized into six sections: the introduction in Section 1, Section 2 recounting some prior works relating to testing, a systematic derivation of proposed methodology is prepared to set up the reference framework in Section 3 with some pertinent procedures presented along with the derivation, Section 4 explaining the experiment in detail as to how the proposed methodology is carried out and some idiosyncrasies demonstrated to express the oddity, Section 5 discussing a few important issues precipitated from the proposed methodology, Section 6 points out some threats to validity, and final thoughts and future work are given in the concluding Section 7.

RELATED WORK

Modernization has increasingly used software to operate all functions for some time. This software runs on traditional systems such as the world wide web, which gradually switched to mobile devices in the form of mobile apps. Their development life cycle runs from the annual scale in traditional systems to months in mobile systems. The most important deciding factor of this development is the time required to release the product. This constraint imposes not only the crunch on development pressure but also on development quality and cost. The rush on releasing apps in shorter timeframe creates faulty software products, which in turn increases the “faulty” cost. Tassey (2002) revealed the estimates of the economic costs of faulty U.S. software that ranged in the tens of billions of U.S. dollars per year, representing just under 1% of the nation’s gross domestic product.

At the core of these issues is the difficulty in defining and measuring software quality. Common attributes include functionality, reliability, usability, efficiency, maintainability, and portability. These quality metrics are largely subjective and do not support quantification that can be used to design testing methods for software developers. Muccini et al. (2012) investigated the challenges and future research directions of testing mobile applications. They found contextual and mobility issues to look into, namely, performance, security, reliability, and energy. The concerns were fault detection, test case (random) selection, and test artifacts.

Edwards (2004) presented one aspect of studies on software testing based on Bloom’s taxonomy, pointing out five perceived roadblocks that plagued learners, from trial-and-error to

Related Content