Chapter 10

Using the Cloud for Testing
NOT Adjunct to Development

W. Morven Gentleman

Computers for People, Canada

ABSTRACT

Software testing, whether performed by the software development organization itself or on behalf of the software development organization by an independent testing organization, is typically described in the literature as part of the development or maintenance process for the purpose of improving the quality of the software product (i.e. finding and removing defects). Nevertheless, interested parties other than the software development organization perform software testing for reasons other than finding and removing defects, and such testing can be facilitated when the software is available as a service in the cloud. Unfortunately, access to software only as a service in the cloud can inhibit certain kinds of testing. In this chapter, the author discusses who such other interested parties might be, what they intend to learn from software testing, and what some of the techniques are they might use.

WHAT IS SOFTWARE TESTING?

This chapter focuses on the opportunities that are created by testing of software that is offered as a service in the cloud. (Software testing could itself be offered as a service in the cloud, but this chapter does not address that.) Software testing is the empirical systematic scientific investigation of a specific existing software artifact. Most commonly, software testing is dynamic in that it involves execution of the software under test, although use of tools to perform static analysis of the software under test is also software testing. Dynamic testing enables examination of the behavior of the software artifact on one particular execution, with particular input in a particular environment. Software that can be available as a service in the cloud opens up the possibility of dynamic testing of that software by many different interested parties. Static testing enables the examination of the behavior in any possible execution, with any possible input in any possible environment. Unfortunately, the access enabled to most software provided as a service in the
cloud prevents such static testing. Persistent data for such software in many cases could be made part of the service-offering tool-accessible in the cloud without undue risk. Whether the manual inspection of a software artifact should also be considered software testing is more contentious. Are the results of manual inspection reproducible, systematic and scientific or merely anecdotal? For Software as a Service, what manual inspection might be plausible?

Software testing is primarily intended to reveal attributes and behavior of the software itself. Sometimes, however, what we need to explore is the behavior and reactions of people when they have to work with that software. We will elaborate on this later. When the software is accessed as a service in the cloud, the people being studied can be anywhere, and the observations can be made at any time, which can be enormously advantageous.

Since software testing is empirical, we need to consider what kind of experiments might be performed, what observations might be made, and how such observations might be analyzed, as well as how the results should be interpreted. Software as a Service in the cloud can affect these.

Risk-based testing is a unifying notion: all software testing can be viewed as risk reduction. A test must enable reduction of some risk for someone. Risks do not relate only to software developers. Other interested parties are often referred to as stakeholders. Unfortunately, many authors restrict the use of the term “stakeholder” to those with financial interest in the product, whereas many interested parties have interest that is not strictly financial, so for increased generality we will stick to the term “other interested parties”. Different interested parties are concerned about different risks, and both the potential damage and the associated probability of occurrence differ for different interested parties even for a risk of the same event.

Defects represent a significant risk to software development organizations. If there are too many defects, or if there are defects that are too serious, the software development organization may not be able to ship the product. In a custom development contract, the client may not accept the product due to defects. For commercial software products, a reputation for buggy software can affect future sales even if the actual known defects are not that relevant. The developer may be liable for damages due to faults. And so on. However, for software development organizations the risk of defects is manageable, in that the developer can make investments to locate and repair defects.

Other interested parties, such as potential customers, are not in a position to repair defects nor are they motivated to do so. These parties are generally more interested in identifying, counting, and summarizing defects than in locating specific defects. That is, their interest is in the consequences of working with the software under test despite its defects. They may be interested in defect density, i.e. which aspects of the software seem to be associated with more bugs, and which aspects seem relatively bug free. Defect density may influence the use of aspects of the software. We shall explore who some of these interested parties may be, and what they hope to learn from testing the software. Note that when interested parties are not part of the development process, getting access to the software in order to learn about it is often a major challenge, which makes the software being available in the cloud as a service very appealing.

Software as a Service has the distinct advantage when performing testing that the testers do not need to have direct access to the source code or the executables of the software under test and do not need to install that software on their own execution platform. Unless the hardware execution platform is being co-developed, the software development organization normally installs the software on their own platforms as it is being produced; in the co-development situation standard practice is to use simulators. On the other hand, not having access to the software itself or to the execution platform (let alone to the source code