Enhancing the Effectiveness of Software Test Automation

A Master's Thesis

Presented to

The Department of Computer and Information Sciences

State University of New York Polytechnic Institute

Utica, New York

in partial fulfillment

of the requirements for the

Master of Science Degree

by

David Wayne Jansing

December 2015

SUNYIT

DEPARTMENT OF COMPUTER SCIENCE

Approved and recommended for acceptance as a thesis in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Science.

____

DATE

____

Jorge E. Novillo

____

Roger Cavallo

____

Scott Spetka

ABSTRACT

Effective software testing can save money and effort by catching problems before they make it very far through the software development process. It is known that the longer a defect remains undetected, the more expensive it is to fix. Testing is, therefore a critical part of the development process. It can also be expensive and labor intensive, particularly when done by hand. It is estimated that the total effort testing software consumes at least half of a project's overall labor. Automation can make much of the testing an organization does more accurate and cheaper than merely putting several people in a room and having them run tests from a paper script. It also frees the testing staff to do more specific and in-depth testing than would otherwise be possible. This paper focuses mainly on software test automation techniques and how automation can enhance the efficiency of a software team as well as the quality of the final product.

Acknowledgements

I would like to thank my wife Ursula and son Alexander, without whose encouragement this undertaking would not have been possible.

I would also like to thank Dr. Jorge Novillo for his help and guidance both during the composition of this thesis and over the past several years.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

Agile process – a test-driven software development process that emphasizes short development cycles with deliverable products at the end of each cycle. Agile also encourages involvement by all “stakeholders,” including the customer.

Black box testing – testing in which the tester exercises the application as the end-user might see it, without seeing the source code. This testing is usually done by hand, but it can also be automated, usually by means of a script.

Coverage – the portion of code for which some test exists.

Expect – a UNIX tool for automating interactive command-line applications that provides inputs for an expected output, such as prompting text.

Extreme programming (XP) – a type of Agile process with even shorter production cycles (in some cases, one week) in which the tests are written before the actual programming code. For this reason it is often called a “test-driven” development process.

Oracle – A test oracle is a means of determining whether a test has passed. For example, the expected output from a function call. There are two parts to an oracle: the expected output itself, and the function or script that evaluates the actual output.

QA – stands for “Quality Assurance,” referring to the part of an organization concerned with preventing mistakes in industrial products, in this case, computer software.

Reference testing – testing where the output of a test is compared against previously captured output. This is what we have been calling a “master file” at my company, and is also called the “golden copy” in the literature. This can take the form of an xml file, a series of log entries, a screenshot, or other artifact.

Regression test – a test that ensures that existing functionality is not affected by changes in the software under test. These tests are prime candidates for automation because they are run fairly often, and they tend to increase in number over time. It would be very expensive to do regression testing by hand.

Smoke test – a test, usually automated, that exercises the basic functionality of a new build to determine viability and whether further testing would be worthwhile.

Unit test – a test that is run by a developer to verify that the code is correct before turning it in. These test individual methods to ensure that they function as expected. The focus is mainly on program logic and correct implementation.[18] In some environments, such as XP, these might be written, at least in part, before the actual source code.

VM – Stands for Vitrual Machine. This is a computer implemented in software that can be run on any computer operating system for which it has been built. The Java Virtual Machine (JVM) is probably the most well-known example.

VNC – stands for Virtual Network Computing. It is a means by which a computer can be used remotely.

White box testing – testing using scripts that test the essential logic of the source code. Unit testing falls under this category. In general, it is testing done by a developer prior to turning in a piece of work. This kind of testing might also be called “formal testing” or “unit testing”.

Chapter 1. Introduction

This thesis will focus on the automation of test procedures, particularly efforts executed using tools created in-house as well as those done with tools purchased from third-party vendors. As one can imagine, software test automation is a software development project in itself. It requires the same attention to process, best practices, and usually some level of programming skill to implement it effectively and economically.[3] Several approaches to automated testing will be discussed in detail. Some examples are testing tools such as JUnit that are included free with various development environments, pre-packaged black-box automation tools and home-grown testing suites. The last of these will be undertaken as a practical exercise; observations and results of this exercise, as well as source code, will be shown in Section 7 and the appendices. I shall also offer my own observations on the subject, drawing upon my previous experiences and those of some of my colleagues.

1.1 Background: Although numerous books and papers have been published on the general topic of software testing, automated or otherwise, the degree to which testing processes are developed and maintained by a given organization or a given project depends on who is leading the test effort. In particular, the value placed on test automation varies considerably from organization to organization.

Observations have been made that testing is generally not adequately covered in university computer science programs, with most testers and developers learning the craft in a much less well-organized way on the job. This is in spite of the fact that testing takes up more than half the time spent on software engineering activities.[15] The reasons given for this is that students often do not have time to do test development on their assignments, and although completed projects get graded, testing is an additional task that usually does not get a grade, so it just doesn't get as much emphasis as it should.[10] Worse yet, some professional developers regard the writing and maintaining of automated test code as a waste of time. Testing in general often gets short shrift when it comes to budgeting, and developers don't get much in the way of formal training in writing good tests. For example, here at SUNY Polytechnic Institute, there is no dedicated course in the catalog devoted to software testing, automated or otherwise, although the subject may come up in individual courses. This is typical of most computer science programs in the USA.[15] Much of what I learned about testing has been by first-hand experience at my job, with process improvement coming mainly from trial and error.

I have done a great deal of software testing in various forms over the past fifteen years. Because of my deep experience with testing, particularly the automation of it, this is the role into which I have been placed most often when joining a new software development team. When I began my software development career in the fall of 1998, I was one of a group of people who were just hired sitting in a room together running tests by hand from a paper script. This approach has its advantages, namely that no one on the testing team need have much if any coding experience[5], and human testers are better at improvising when paper test scripts are out-of-date, but it is also error-prone and slow.

A few years later, I did what we called "build integration." The engineer in charge of managing the baseline would write and maintain a large suite of "smoke" test scripts using Perl and Expect to verify basic functionality. The build was started at the end of the day by means of a Perl script with an automated install and run of the test scripts following immediately afterward. If everything went well, one would have a finished build packaged for shipment, installed and ready to run on a machine, along with a set of logs for the installation and the test results, waiting upon arrival in the morning. The workday could then be used analyzing failed tests, writing bug reports, troubleshooting test scripts, testing the bug fixes newly integrated into the baseline, and so on, before the whole process started again in the late afternoon. This was my introduction to test automation.

Over time, these scripts became more elaborate and the set of tests grew until the smoke test suite was expected to do all sorts of things, not all of which had to do with answering the basic question of whether anything that was just added to the build introduced bugs. They became a sort of additional regression test suite that ran before the actual test team received the build to do their own regression tests. This expansion of the integrator's role was possible because all tests run by the automated scripts could be done at little to no cost because there were about fifteen or so hours available from quitting time to the next morning to run tests that would otherwise go to waste. Ideally, the test team would have to work harder to find problems, since all the obvious ones have already been found and fixed by the time they received a CD. [4]

Eventually, we added a GUI-based testing tool that allowed the creation of test scripts by means of capturing mouse and keyboard activity and replaying it. Since much of the user interactions with the software took place via the graphic interface, this tool enhanced our testing capability a great deal. It also presented its own set of problems, not the least of which were those that become evident when the GUI itself had been changed. A more thorough discussion of third-party testing tools will be presented in Chapter 6.

The remainder of this thesis will consist of a discussion of the motivations for automating testing, real and perceived disadvantages of automated testing approaches, possible strategies for overcoming these disadvantages, and practical exercises demonstrating some automated testing strategies.

1.2 General considerations for software testing: In order for any form of software testing, automated or not, to be effective, certain considerations need to be addressed.

1.2.1 Requirements: Every software project begins with a set of requirements, that is, what the software under test is expected to do. For each of these requirements, at least one test case may be written that is part of a test plan. This goal is to be able to sufficiently test each requirement.[20]

1.2.2 Coverage: This describes the percentage of code that is tested by the set of test cases. This is the main concern of the individual developer when implementing unit tests. Numerous sources indicate that getting to 100% coverage is impossible for a software project of any significant size, so emphasis is instead focused on covering the most critical parts of the code. In fact, metrics that suggest something approaching 100% coverage may mean that the development team is implementing many low-quality tests with the sole goal of achieving 100% coverage.[21] More important than code coverage is test coverage, which is the percentage of requirements, not code, that is covered by the tests. This is the metric over which most QA departments are concerned.[22]

1.2.3 Time and budget: How much time is allotted for completing all of the testing tasks, and what resources (hardware, testing tools, personnel, etc.) are to be allocated for the testing effort. This includes expertise; for example, whether your testers are comfortable with writing test scripts.

Chapter 2. Motivation for the Automation of Testing

Testing software is oftentimes repetitive in nature. A list of steps is executed, the output is examined, the system is restored to its pre-test state, and the list is executed again. Perhaps there is a room full of testers, all running tests on the same system. Maybe the testers have programming skills, or maybe not. In any case, running test cases by hand is error prone, slow, and expensive. Fatigue and ennui have the effect of reducing the level of attention to detail paid to test execution and critical test steps may be missed or haphazardly done. Automated tests, on the other hand, run the same way every time and reduce errors that result from these factors.

2.1 Increases in productivity: Ideally, an automated test suite should run without any user intervention at all. One would, at the end of the work day, have the latest build installed and ready to go, and the last thing that would need to be done before going home for the day would be to start the tests. The next morning, there would be a report ready to inspect, or at a minimum, a list of which tests passed and which failed. Since no one was in the office running tests overnight, the company just saved the expense of having people in the office after hours running tests, and the test team can get more testing done by being freed up to do the kinds of tasks that are best done manually, such as investigating specific problems, including those not normally encountered by the automated test suite.[11]

Automated testing makes possible the kind of short release cycles demanded by development processes such as Extreme Programming (XP). Bugs introduced by new changes in the code are detected at an earlier stage of the development process, thus saving money and effort. [17] For example, a group of testers at Microsoft were surveyed, and they said most of the more easily found defects were being caught by the automated tests, leaving them with more time to deal with the more difficult ones, which are usually smaller in scope and less obvious than the more comprehensive ones found by the test suites.[4] Lastly, an automated test can process vastly greater quantities of test data than is possible when doing testing by hand. A test requiring a few thousand JPEG files, for example, would be impractical to do by hand, but nearly trivial when done by an automated script running overnight.

2.2 Increases in reliability: The best tests are those that are easily repeatable, and having to do everything by hand can undermines that. Automated tests are more reliable than tests run manually because they run exactly the same way each time they are run. A test run or set up by hand is subject to human error, particularly if the test is repetitive, or requires much preparation, or any other effort that requires great attention to detail. Automating them ensures that the same test is run every time. Thus, the quality of testing improves when testers are freed from tedious, repetitive tasks to develop more and better test cases that go deeper into the system and provide greater coverage than would be possible otherwise. [17]

2.3 Increases in coverage: Automated tests are fast. A computer isn't going to go back and forth between a written script and execution of the tests. The test software already has those instructions built in. To do stress testing, that is, testing a system until it fails due to the system on which it is running having reached its physical limitations, there usually isn't any better way to do it than by running a script. Since more tests are run in less time, greater "coverage" of the software under test can be achieved.

Chapter 3. General Automation Strategies

Organizations who use the waterfall method of software development do most of the testing at the end of the development effort, when there is a complete, running piece of software to test. Defects are then found and fixed. Those who use some form of Agile will develop tests closer to the beginning of the process, during the initial planning and design phases. Ideally, testability should be built into the code; this is easier when the unit tests are written before the code, which is what is usually meant by "test-driven development." End users should also be consulted on the test plan to ensure that the proposed test scripts satisfy the user's requirements.

3.1 Types of Test Strategies: Test strategies are lumped into two different groups:

Defect prevention strategies – These provide the greatest cost and schedule savings over the course of a project because they are employed from the outset of a software project and therefore tend to find defects earlier, when they can be more easily resolved. The existence of these indicates a proactive organization with a high level of test discipline and maturity.

Defect detection strategies, which ideally supplement prevention strategies. These are usually employed after the completion of a runnable software product.

There are constraints that may prevent the implementation of a more ideal test strategy, such as a short development schedule, limitations on engineering resources, both human and material, and a lack of familiarity with the design process or testing tools. [1]

3.2 Definition of testability: An informal definition of testability is, given an initial system state and set of inputs, the outputs of testing, whether done by hand or by using automated test scripts, are predictable and repeatable, that is, deterministic.[1] When it is not possible to guarantee this, such as when developing a video game, automated testing is likely not to be possible.

The IEEE offers a more formal definition of testability:

> *1. The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.*
>
> *2. The degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met.* [27] [33]

The testability of code is enhanced by the cohesiveness of the individual methods, that is, the tests should be small in scope and do just one thing, and loosely coupled, that is, they are linked only by required data, thus, there are fewer opportunities for one test to interfere with the functioning of another test. The errors that remain should be mostly those that are caught by the compiler or by code analysis tools such as Rational Purify.[1]

3.3 Order of Testing: Analysis of a large number of failed tests can take some time. To shorten this, the testing team should do a risk assessment to determine the priority of the various tests to be done. Priority should be given to those functions that are most likely to fail or cause problems that render the software unacceptable.[1] Tests can then be grouped in a hierarchy, with the most critical tests coming first, which are likely to be general in scope, and more specific tests running after that if the general test passed. The idea here is that there usually isn't much sense running the subtests at all if the general tests, which test for more serious problems than the subtests, indicate that the build is broken.

I have a set of unofficial tests at my worksite where I test the new build I just installed before going on with the "real" tests. Before this, I was oftentimes disappointed in what I thought was a good install only to find the next morning that hardly any of my tests ran properly because I had forgotten some part of the post-installation setup (for example) or there was a major problem with the build that could have been detected right away instead of waiting until the next day when the automated test results were available. For this reason, I like to start testing around 2:30 or 3:00 in the afternoon, so I can watch the first few tests pass or fail and give the developers a chance to fix any show-stoppers then rather than the next day.

Again, the more this process is automated, the less chance of that happening; instead of taking two to three days to get a clean run of tests, I can now have test results the next day in most instances.[16]

Chapter 4. Mitigation of Perceived Disadvantages to Automation

This is an example of what I might encounter as a tester running a suite of automated tests that I developed and ran overnight on a new build. I would come in to the office in the morning, check the results of the test run I started the evening before, and find some of the tests had failed. I would then check the logs and find the exceptions that were thrown. I consider this complete enough information to give to the developers so they can investigate and find the problem, provided that the test was well-written. Sometimes, I will repeat just that test suite to make sure. If the retest found the same problem, I consider my testing effort successful, and I can explain exactly how it failed before the 8 AM meeting. I would then report my findings at that time, armed with enough information to pinpoint the problem.

“Did you test it by hand?” is what the lead would ask me. I did not, because I only had about half an hour to do a quick retest and get the information that I did have. Since there isn’t time to do this before 8 AM, all I can offer is the log segment where the failure occurred and a brief explanation of what my automated test was doing at the time. Investigation by hand would have to take place after the meeting, and it will take awhile, especially if I make mistakes, which is why the test is automated. The truth is, the developers still do not trust that a problem exists when an automated program found it while I was at home doing something else. There appears to be a mentality at some organizations that tests that run without user supervision don’t really test anything, that somehow the person using the automated test suites isn't doing his or her job when the reality is that test automation is adding to and facilitating the other testing that is typically done.

We trust software because it presumably has been extensively tested, but what about the tests? How have the tests themselves been tested? Since an automated test is a piece of software itself, subject to the same problems as production code, how was that tested? The answer, as I see it, is that the automated tests are designed to alert the developers that a problem may exist, not to be the last word on whether the software is free from defects. In my experience, the automated tests do find actual problems quite reliably, but they require some verification before the problems are brought to the developers. They are trusted, with subsequent verification, because they deliver consistent results over time.[2] This and, at some point, the tests have to be tested manually.

Manual testing is an important part of the software development process, as is automation. Many problems will not be found any other way than getting one's hands dirty, so to speak, with a piece of software. One might have found a novel way to run the software under test that is not covered by the automated test suite. Tests such as these that are run infrequently or only once are poor candidates for automation, since it takes between three and ten runs of an automated test to recoup the cost of the effort creating it. [16] Other tests fail frequently for reasons having nothing to do with faults in the build. Maybe a variable name was changed for clarity, or a configuration file moved from one location to another. Maybe the program output is nondeterministic. For example, I cannot think of another good way to test a video game other than playing it. Regardless of the circumstances, it doesn't take many failures to undermine confidence in the automated tests among the developers and the leads.

Automated and manual testing are actually two completely different processes. Automation is best used to exercise previously existing functionality (regression testing), while a manual tester is focused on finding new ways to break the system. In this way, the automated tests free the testers to do more in-depth manual testing instead of running basic tests by hand.[11] Knowing which test cases are to be run fairly often, and which only intermittently, is important to those who seek to maximize the benefits of test automation.

4.1 Reducing the Complexity of Automated Tests: Now-defunct online service provider Prodigy had an ambitions AST program called Gremlin that failed to catch on after much fanfare. It was a capture/replay tool that captured keystrokes and screen images to generate tests.   For awhile, there were dozens of regular users, but eventually this number fell off from 71 to less than 20. This was the first of two instances during this effort in which the test package was shelved for various reasons.

The falloff was caused by unnecessary complexity. While the tests were automated, the tools still required much manual setup and teardown. This made them error-prone and not much more usable than manual testing, even with practice. There was hardcoded data in the scripts themselves. This made the scripts unusable probably with every build. Having to reprogram the scripts nearly every time they were run, resetting all the data, recompiling the test program, and then running the test was tedious to the point where users would give up and test by hand. Users also wanted useful metrics generated post-test (how many tests passed/failed, portions of code covered, etc.), and this also wasn't done very well. [16]

In my own experience, I inherited an very ambitious attempt at an automated test system from someone who had left the company. It used IBM STAF to do a complete reinstall every time a set of tests completed on a "farm" of machines running in parallel, and then run the tests again. It took me three months to get the suite running again after it had sat for a year. The biggest problem with it was that the install required a system reboot, which halted STAF, and when STAF failed to recover, the whole system would hang. I solved this by doing the install manually, and have STAFF merely run the parallel automated tests, clean up after itself, and repeat. It still didn't quite live up to expectations, mainly due to the massive amount of maintenance the tests still required, and after I had been reassigned, this effort became a sort of "zombie" project that would come back from the dead from time to time, so to speak. Another developer would ask me about the autotest effort from time to time because I was the last one to touch it; apparently, since there were considerable sunk costs in terms of man-hours, management found it difficult to just let it go. I would refer them to the README I had written describing my effort at resuscitating this set of suites, but that ended up raising more questions than it answered because of the layers upon layers of complexity in the system. When I had a chance to design my own automated test framework, it was considerably simpler than that, as I shown below.

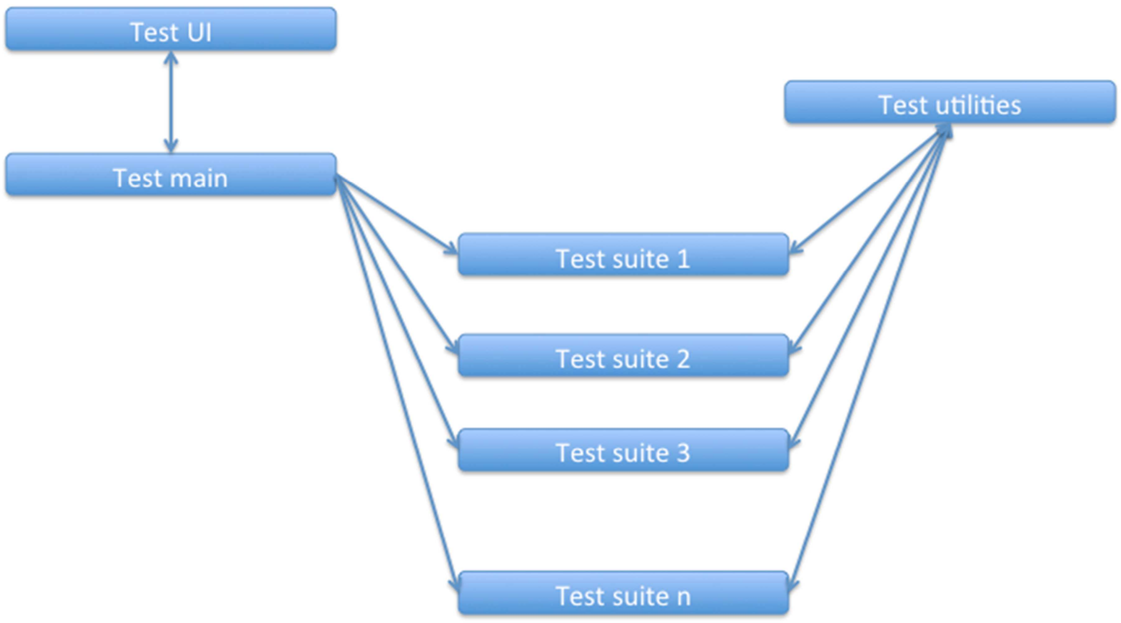


Figure 1. A diagram of the structure of an automated test system designed for simplicity, maintainability, and ease of use. The test utilities package may consist of several classes whose methods are grouped by function. Each test suite does just one thing, and is responsible for its own setup and tear down.

It was suggested in an article by David McClure [2] that the most valuable tests are the ones that are easy to write. That is, it is concise, which leaves less room for error, and tests a small, well-defined part of the program. For example, let us say the system under test has a function whereby an incoming file is processed in some way and placed into a repository. The most basic test of this function would be one that took a file, saw that it was deposited, and pronounced the file processor's basic functionality to be working properly. If this test passes, other more in-depth tests can be run after that. Otherwise, there isn't much sense in continuing.

The following graphic illustrates how simplicity results in net value for an organization. If the test is more complicated than the code it is testing, it isn't of much use to the organization.[2]

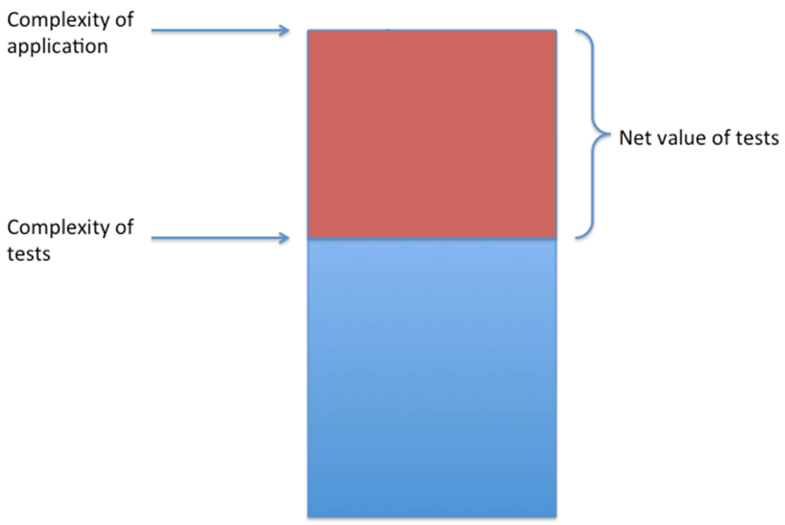


Figure 2. A measure of the net value of a test.

The developers at Microsoft from the McClure article also felt that the quality of tests was greater when they were written more incrementally as is the usual practice with TDD. The actual results of the case study indicated that coverage tended to be greater when tests were written this way. In general, this is what they have found:

- The leadership needs to realize the benefits of unit testing. They will want to know the size of the project, the staff training requirements, how much the tools and the programming effort will cost, the return on this investment, and a host of other things. The better management is informed about an automation project, the better chance it will catch on.

- Unit tests should be treated as part of the production code. That is to say, the same version control protocols, code reviews, coding standards, and so on, should be applied to test code as the rest of the code base. Test code that was produced according to some established standard should be more trustworthy than code that has been put together without such regard.

- Testability of the system should be part of the design. TDD aids in this greatly by requiring the tests to be written first, before the production code. Involvement from the developers is also important. Testability is discussed in more detail in Chapter 5.

- The quantity of tests, as also stated in the previous article, is a poor measurement of success. Better is some measurement of coverage, which is the amount or percentage of the source code for which tests exist.

- Execution of the test suite should be easy. In the real world, I wouldn't know how easy my test suites are to run because I am the only one running them.

> They're easy for me because I have had a lot of practice. Another developer, on the other hand, might have difficulty due to unforeseen considerations that have not been addressed, such as how one's environment is to be set up, or the structure of the file system to be used. Since Microsoft's programmers are encouraged to run each other's tests, they tended to be more user-friendly than would otherwise have been the case. [33]

With my own AST efforts, I recognized early on that in order for this to work, tester intervention had to be eliminated, that is, the tester issues a command or clicks a button, and the tests run to completion overnight when nobody is working. Setup requirements were also mostly automated, reduced to less than a dozen settings in an XML file that can be saved in some location where it can be easily retrieved. The actual setup, which would have taken most of an afternoon, was finished in less than a minute, and done the same way each time.

4.2 Startup costs: The startup costs associated with test automation is significant in terms of the programming effort, training, and tool selection.[12] The design must be decided upon, the tools selected, licenses paid for, the personnel trained to use the tools, and all before even one test script can be written. Since test automation is a software development project with associated expenses, it may be some time before any practical benefit can be realized. Usually, it is not until the second major release of the software, or even later, that the costs are recouped.[16]

The expense involved in the initial creation of the automated test cases includes the design effort as well as coding. Generally, the return on investment of automated testing efforts can be described thus:

$$ROI = \frac{\text{Lifecycle Benefit} - \text{Cost of Effort}}{\text{Cost of Effort}}$$

It sounds fairly obvious, but the benefit has to be greater than the cost. The Mueller-Padberg paper[18] is mainly focused on Test-Driven Development, which a key component of XP and other Agile processes, but it can be applied to any project where some part of the testing process is to be automated.

Using a store-bought tool requires less development effort and can therefore be cheaper than writing test software from scratch. Such a tool will be well-supported because the person in charge of purchasing it with the company's money will make sure of this. Fewster[16] also suggests that there is a certain "cool" factor associated with vendor-purchased test tools. Developers will be more eager to be part of its integration than the development of a home-grown tool. He also points out that purchasing a tool will not completely eliminate the need to write test code, which is usually done in a proprietary scripting language specific to that tool.

Choosing a tool also requires effort. A co-worker of mine suggested that about 80% of commercial test software is "junk." What he meant was that he has to expend much time downloading and evaluating several tools before he finds one that meets his needs. He cites user-friendliness problems and long learning curves as the main problem with many of them, followed by additional demand on system resources and system incompatibility.

After this initial analysis, one might decide that the commercial tools do not meet the needs of the development team and build a tool in-house rather than buy one. The advantage of this approach is that anything one builds oneself is automatically well suited to the specific needs of the project team. However, it isn't as likely to be well documented as something bought from a vendor. It would be even more designed "by geeks for geeks" than the commercial tool, and its overall usefulness is dependent on the skills of the members of the test development team. As a result, it could suffer from image problems; working on the test tool isn't considered as exciting as working on the main project.[16]

4.3 Maintainability: Maintainability is probably of at least equal consideration with the speed at which the tests run when composing automated test scripts. A test that is easily maintained is usually more reliable than one that is not. The temptation exists to employ programming tricks when writing a script to make it run just a bit faster. That's what software developers do. We are, after all, talking about part of a suite of tests that takes several hours to run end to end, and to shave a few minutes from the running time of each test is a big deal. At the same time, one must keep in mind that somebody else unfamiliar with the programming style of the original author of the test software might end up having to maintain it. This happened to me last summer. I was pulled off the project for a few months, and when I returned, some of the tests were no longer working because they hadn't been run in awhile, and there were some new tests that, while they were functional, they were not easily maintainable and had to be reworked. In at least one instance, my intent was misinterpreted, leading to problems that took longer to resolve than would otherwise be the case.

The fact that a test program is solely to be used in house, and not shown to customers, is no excuse for not using the same software development process that is used to develop the software being tested. For example, during the test automation effort I have been describing, I inherited a test package that consisted of only a few Java files, with the main implementation taking place in the constructor of the actual test class. This made the test function like a shell script might, and did not take advantage of many of the features of the Java programming language at all, but the scope of this test was so small that it was good enough at the time.

My task was to use this class as a template to create several new automated tests. While such a program will compile and run, it is not considered a programming "best practice." It also doesn't easily lend itself to the development of an expanded set of tests. The reason the original test was written this way was likely because it needed to be put together quickly because it was needed right away, so the focus was on merely making the test class work. According to Berner/Weber/Keller, this is commonplace in the software industry[18].

The first problem I noticed with the copy-paste method of test development was that by merely cloning the original test to make new tests, maintenance was a nightmare. I had numerous methods that were identical in each class, and updating one of them meant updating them in each of the test classes. I placed these most commonly used methods into a utility class where they only had to be updated once. The test code was then taken out of the constructors and placed into their own static methods, which could eventually be reused by the developers in their unit testing. The initial effort was great, but the eventual payoff was much greater. In the end, the tests became easier to maintain, and the development time for new tests decreased by about half.

I had discovered what Fewster[16] stated in his book, that there are attributes to well-behaved tests:

- Tests should be maintainable, as mentioned above.
- Tests should be modular, that is, each test tests just one thing. They should not be dependent upon other tests passing in order to give the desired results. Part of achieving this is having each test responsible for its own setup and restoring the system to its original parameters upon completion.
- Tests should be robust, which I take to mean that changes in the software under test are less likely to break the test.
- Tests should be well-documented, which goes back to maintainability. Other test developers should be able to determine what the original author had in mind, and how the test is generally supposed to work.[16]

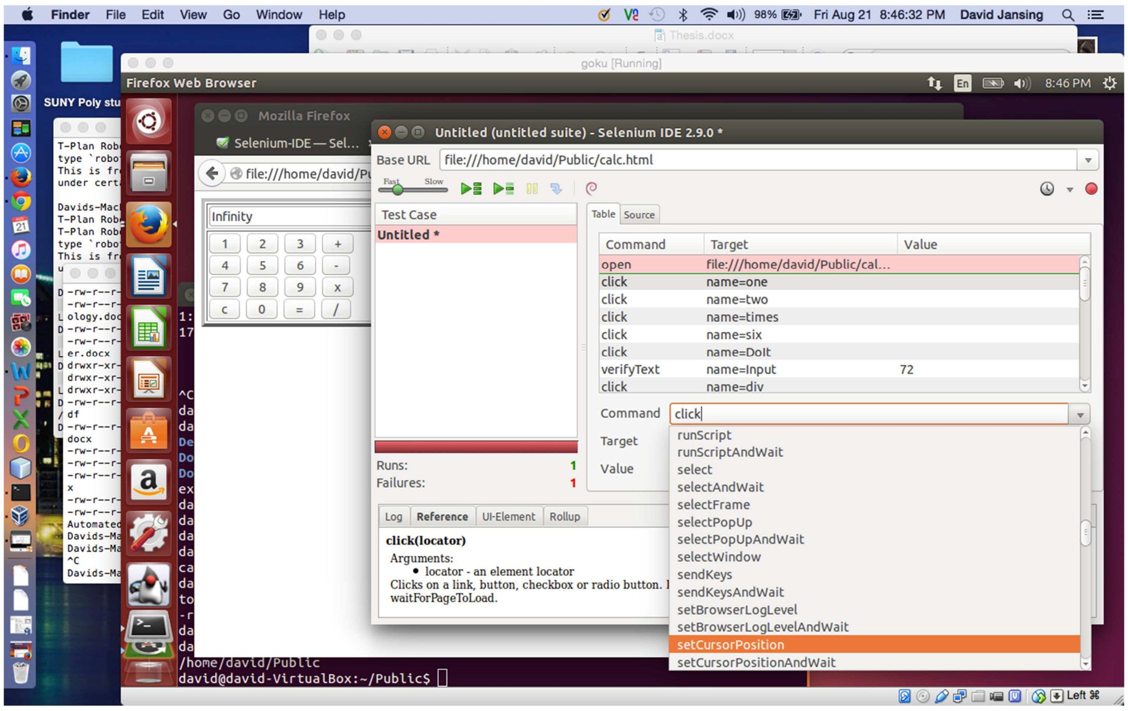
- Tests should be built of reusable components. Being able to use methods from a utility class is a lot easier than having to repeatedly rewrite and revise them in each test.

4.4 Commercial testing packages: Commercial testing packages can be expensive, although they can reduce the required effort and expertise in generating and maintaining automated test cases. For example, A license for a single developer for QF-Test, a popular testing package with which I am personally familiar, starts at €1,995 (about $2300)[6]. According to Aspire Systems, Quick Test Pro (QTP), developed by Hewlett-Packard, is the dominant commercial testing tool on the market. It is not only expensive, the cost is scaled to how much it is used and how large the operation is.[19]

In addition to the cost involved in commercial testing packages, many such tools use proprietary scripting languages for which the testers may require additional training and practice before being able to use them effectively. Quick Test Pro, described above, is one example. QF-Test, which I describe in more detail in Chapter 6, uses something similar to XML as the basis for their test scripts. These scripts tend to be rather large, even for a small number of test steps, so one is dependent on the IDE to do work on them. There could be problems with the portability of test scripts generated by commercial test tools, should it becomes necessary to change tools, whereas a test script written in a more universally known language such as Perl would still be usable.[23]

4.5 Open Source Testing Packages: Selenium is an open-source capture/replay test tool for Web applications and is therefore free. The interface is about as simple as it gets. There is a red button to click on the toolbar for recording tests, the tester would then do the test manually, then click the button again to stop recording. The test is recorded in html, and the IDE expresses commands in much the same way QF-Test does. To do checking on results, there is a large number of verification commands. For purposes of this sample test, I used VerifyText after each calculation and VerifyTextNotPresent upon clearing the display. To save a test script, merely use the File menu selection in Firefox, which is smart enough to recognize that one is using Selenium, and click "Save Test Case."

Figure 3 shows a screenshot of me using Selenium on a Javascript calculator I found on the Internet. I could just as easily have run a test on Google's home page or any page on the Web. Figure 4 shows an example of one command from the script in Figure 3 expressed in html. Note that neither format is particularly complicated. I created and ran a simple test of the calculator's basic functions in a couple of minutes.


Finder File Edit View Go Window Help
Fri Aug 21 8:46:32 PM David Jansing
Firefox Web Browser
Mozilla Firefox
Selenium-IDE — Sel...
Untitled (untitled suite) - Selenium IDE 2.9.0 *
Test Case
Untitled *
Table Source
Log Reference UI-Element Rollup

Base URL: file:///home/david/Public/calc.html

Command	Target	Value
open	file:///home/david/Public/cal...	
click	name=one	
click	name=two	
click	name=times	
click	name=six	
click	name=DoIt	
verifyText	name=Input	72
click	name=div	

Command: click

Runs: 1
Failures: 1

click(locator)
Arguments:
- locator - an element locator

Clicks on a link, button, checkbox or radio button. I
waitForPageToLoad.

```
/home/david/Public
david@david-VirtualBox:~/Public$
```

Figure 3. Selenium being run on a web application.

```
<tr>
     <td>verifyText</td>
     <td>name=Input</td>
     <td>Infinity</td>
</tr>
```

Figure 4. A single Selenium command expressed in html

Chapter 5. Automated Testing Under Alternative Development Methodologies

Traditionally, software development teams use a process often called the "waterfall" method, in which each aspect of programming, from requirements documentation to design to actual programming, is done sequentially, with each phase starting only when the previous one is completed, and nothing being available to test until it was mostly complete. Then the testers were given the list of requirements and tasked with deriving test cases using the completed software, which they have not previously seen, as a guide.

This was the way we worked in the 1990s when I received my BS and started my first job. There were several of us in a room, each sitting at a terminal, performing test cases by hand. Stress testing consisted of testers opening several windows, each running the UI, setting up all the transactions, and executing them one after the other as fast as we could. As one can see, this can be a both expensive and relatively ineffective way of doing testing. Testing of basic functionality can take many days to complete, and many subtle defects can be overlooked due to lack of time. Oftentimes these defects persist well into the product lifecycle, where fixing such defects becomes more and more difficult and expensive. Since then, other methodologies have emerged, some of which I will discuss briefly in the sections below.

5.1. Agile: There are many processes that fall under the umbrella of Agile. Scrum is one of the most commonly known. Agile processes not only benefit greatly from test automation, but automated testing is an integral part of the development process. In most forms of Agile, the tests are written before the actual code. One version, called Extreme Programming is discussed in the next section.

James Bach [26] describes a "toolsmith," in his paper, who is a dedicated programmer assigned to the test team whose job it is to write/generate test scripts and test data. This is in addition to unit testing, which developers do. Any work done would need to dovetail into the Agile development schedule, which in scrum is either 30 days or four weeks, and weekly for Extreme Programming (XP). Tests that take longer than one Agile cycle to develop would require some business justification. The toolsmith should also understand what it is to be a good tester.

In general, the process for generating tests in an Agile environment would go something like this:

- Understand how the testing is done.
- Identify some tool/script/procedure for automating it that makes the testing more productive/easier/more thorough/etc.
- Deliver the solution in less than the end of the sprint.

By test generator, he explains that he mostly means data generators. For example, a tester might ask the toolsmith,"Can you give me a table with 2000 rows of random input values for Test X?" Automated testing efforts in general should be "directed by testers," that is, the quality of the effort should be measured in how well it makes the non-programming testers' jobs easier.

Test automation depends on the "testability" of the software. Not everything is arranged nicely so that automated tests can be made for it. This is usually not the top priority for developers, who have their own jobs to do.[26] Tests for GUIs are particularly hard to automate effectively compared to programs that run from the command line. The chief complaint is that the interface itself is subject to frequent change; a test that relies upon screen captures to validate a test will need nearly equally frequent re-recording with every change. [3] Because of the requirement that there be some form of output such as a screenshot from the actual GUI under test, it is difficult to implement GUI testing in a test-driven development (TDD) environment.[31]

I managed to integrate GUI method calls into the testing framework for the demonstration program in Chapter 6 by noting that the Java Swing package also includes methods for using GUI controls in test methods. This was also not without problems. For one thing, NetBeans automatically gives GUI components private access. This access would either have to be lowered to some level that would allow a test program to see it, or accessor methods would have to be written for them. The former is a violation of the object-oriented principle of access control. The latter would require additional work, either from the test automation engineer or from the development team.

5.2. Extreme Programming (XP): XP is an explicitly test-driven development process in which the tests are written before the code. In XP, the code changes continuously and quickly, so having someone writing automated test cases while everyone else is writing code is like shooting at a moving target. The test cases never get done well, and they aren't expanded upon, and don't provide good coverage because it's all the test engineer can do to keep up with the developers. Having the developers write their unit tests before they write the code saves a lot of time because that frees the testers to do more in-depth testing than would otherwise be the case.

Since the tests are written at about the same time the software is being written, there will already be a considerable catalog of tests that can be run continuously, thus the whole product gets tested every day. [10]

5.3.1 JUnit: This is a testing suite for Java that has been ported for various other programming languages that does some test generation and provides a framework by which one can quickly write unit tests for each method in a program. It is built into both Eclipse and NetBeans and is considered essential for test-driven development processes such as Extreme Programming. In Chapter 6, I describe my work with JUnit in testing a sample application that I wrote for demonstration purposes.

The basic "building blocks" of JUnit that are most often used are assertions. Assertions compare test output against expected output and return a Boolean true or false. A very basic example might look like `assertEquals("failure: strings not equal", instance.getString(), expectedString)`, where the first argument is the error message to be displayed upon failure, the second is the result of the method call, and the third is the expected output. Closely related to assertions are matchers, which take the form `assertThat([value], [matcher statement])`, where value is the output from the test, and the matcher statement is some expected value or condition. Taken together, JUnit makes creating clean, maintainable test code easy.

5.3.2 TestNG: TestNG is also included in NetBeans, and is similar to JUnit with some new features. For example, there is wider use of annotations than in JUnit. In addition to @BeforeClass and @AfterClass, which designate methods to run before and after the set of tests, respectively, TestNG also has @BeforeSuite and @AfterSuite, @BeforeTest and @AfterTest, and @BeforeGroup and @AfterGroup, which set methods to run before or after suites, individual tests, or groups of tests. Grouping is another part of TestNG that is not supported by JUnit. Tests can be placed in one or several groups for various purposes, such as the case where there are a group of related tests that are often run together.

TestNG is also intended for use in a wider range of test types, such as integration or regression testing, than is JUnit, which is used mainly for unit testing at the developer level. [34]

From reading some of the forums on TestNG and JUnit, I found that they are generally equivalent to each other and the debate whether one is easier or more flexible, or more powerful is a matter of personal preference. For example, at first glance, you might see a TestNG assertion that looks like this:

```
assert 30.0 == instance.ingredients.get(0).getQuantity();
```

An assertion that tests any logical statement that can be put in an if() is a very useful thing, but the equivalent expression in JUnit would be just as easily done as:

```
assertTrue(30.0 == instance.ingredients.get(0).getQuantity());
```

These two statements are equivalent, but in JUnit, one would more commonly use assertEquals instead. In general, it is indeed possible to replicate the functionality of TestNG methods with JUnit, but sometimes it takes a little maneuvering to do so.

5.4 Testing As A Service: Large software projects usually include a large number of test cases, which take a long time to run, making the proper implementation of agile development processes like Extreme Programming, which rely heavily on automated testing, more difficult. Since each test takes a fixed period of time, the only way to reduce the turnaround time of such testing is through massively parallel execution.[25] If a branch is encountered during execution of a test, for example, threads are spawned where each of the options (say, true/false) is explored. It is efficient because it is concerned with classes of inputs instead of individual ones. It is also much more efficient in a parallel environment than on an individual machine. Problems with symbolic execution in a cloud setting are fluctuations in the quality of connections and availability of resources that do not arise when you own the cluster.[24]

A paper published by Candea, Bucur, and Zamfir [24] presents a scheme to take automated software testing to the Cloud as a Web Service, where developers, end users, and certification agencies alike can have access to it. The testing package checks the various paths through a piece of code against a collection of "predicates" which behave like JUnit assert() statements.

Universal predicates describe common bugs of the type that throw exceptions for such problems as null pointers or array overflows. Application predicates are unique to the software under test. For instance, if a user clicks "download," does the download actually take place?

Because it is run on very large cloud servers, Testing as a Service (TaaS) can use parallelism and symbolic execution to explore more paths in less time than would otherwise be feasible on a desktop computer.

An interesting idea asserted here is that end users would find such testing useful. An example is given where someone's grandmother could have a testing service automatically test software updates to her mobile phone for critical applications that needs be kept running all the time before installing them, thus preventing outages that inadequately tested updates can sometimes cause. Crowdsourcing testing in this way could provide much greater coverage than is otherwise possible.

The certification service, which is a proposed independent testing service for software akin to Underwriter's Laboratories, is something I also find interesting. One would certainly have more confidence in a piece of software that has been tested independently and given a seal of approval. I might suggest that Consumer's Union, publishers of the magazine *Consumer Reports*, might also be interested in such an idea, especially if detailed results such as the type and prevalence of bugs could be made available for publication. What was not made clear was who would be responsible for providing the list of requirements or developing the test cases in such a scenario.[24]

The results of employing parallelism in testing are illustrated in the experiments cited in [25]. This experiment had 18 machines running in parallel on Amazon Web

Service (AWS). The efficiency measured by these experiments depends on several factors:

- Cost of distributing packets over the network
- Processing power of the machines in the cluster
- Ability to input/output data in the cloud
- Amount of available resources on the cloud server

Average speedup in the “medium” scenario (greater available computing resources than “micro” or “small”), which provided the most reliable data, was 7.83x and the parallel efficiency was 0.43. Note that speedup, that is, efficiency is roughly equal to the size of the cluster, which was 18. The actual calculation comes to 18.2, which would indicate some rounding error involved in their numbers.

Cost/benefit ratios were also provided. The cheapest, that is, “micro” scenarios provided the greatest value, 36 cents per round of testing at an average speedup multiple of 2.61. The “medium” scenario was much more expensive in relation to the improvement in speed ($3.06 for a speedup multiple of 7.83), so it appears as if Amazon is charging more per unit of work to customers who require a larger footprint.

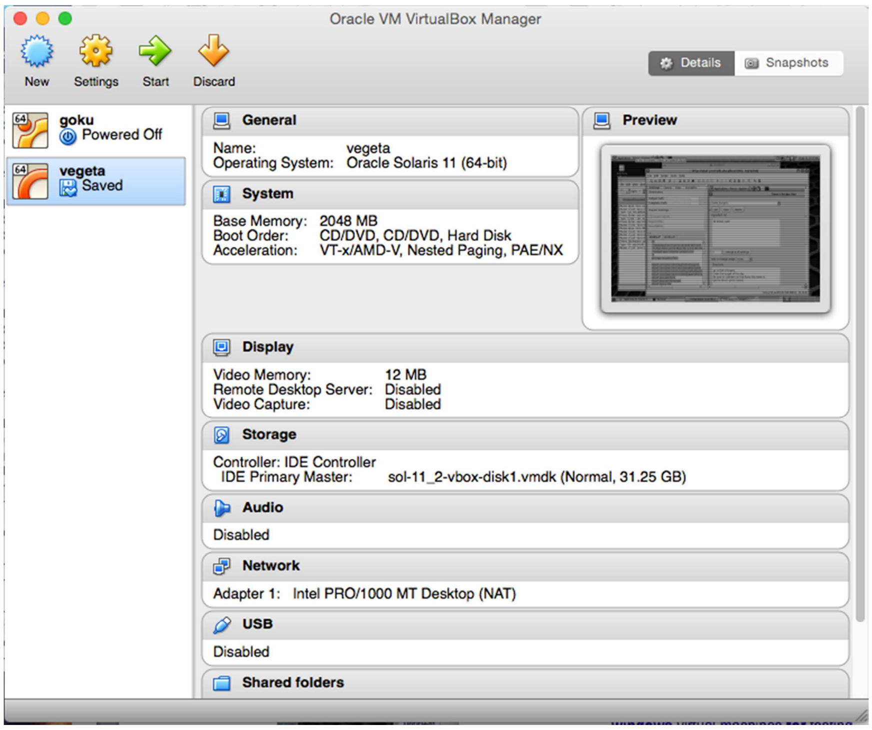
Chapter 6. Exercise 1 - Commercial and Open Source AST Tools

For the test demonstrations, I created a small application called RecipeBox that maintains a database of recipes. It has some nice features, like the ability to adjust quantities for varying serving sizes, the ability to create, edit, and delete recipes, and so on. Since I threw it together rather hurriedly, there were still a few bugs. For example, there is a toggle button to switch from standard U.S. measurement units to metric that doesn't function at all, the adjusted quantities are all in decimal (X.xx) format instead of fractional (X x/x) format that an end user might like, and so on. For these reasons, some of the tests described in Appendix B may fail. I left the software in this incomplete state precisely because I wanted to see some of the tests fail for demonstration purposes.

I looked at two commercial testing packages that I have used in the past. The first is QF-Test, which is sold by Quality First Software GmbH in Germany. The other is T-Plan Robot, which is a commercial product that also offers a free version available at SourceForge. These tools are best suited to black box testing, since they concern themselves only with the graphical interface in the software under test.

6.1 Virtual Machines – Not every commercial test package is compatible with every platform or every programming language. The occasion also arises where more the software must be tested on multiple platforms, for example, a web application that should work with all of a set of the most popular web browsers, or should work equally well in iOS and Android as it does in Windows. These problems can be overcome by the use of virtual machines (VMs). VMs also save the expense of purchasing a large number of computers for testing purposes. More than one VM can run on a particular machine, for example, a series of VMs all running different versions of Microsoft Windows to test compatibility. Another cost advantage to using VMs is that when maintenance is required, one need only update the VM once, rather than push the update to a large number of separate machines. [30]

Oracle Corporation offers a "virtualization" program for free on its website called VirtualBox [29] which I installed on my MacBook, then went looking for an operating system that would work with TPlan Robot through a VM. This was also problematic in the case of my first choice, Ubuntu Linux, mainly because of compatibility issues with that operating system and VirtualBox. It eventually occurred to me that Oracle also offers a free version of Solaris 11, and had a disk image already set up to be used with VirtualBox. Figure 5 shows the VM manager for VirtualBox. Machine goku runs Ubuntu, while machine vegeta runs Solaris 11.



goku
Powered Off

vegeta
Saved

General

Name: vegeta
Operating System: Oracle Solaris 11 (64-bit)

System

Base Memory: 2048 MB
Boot Order: CD/DVD, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE Controller
IDE Primary Master: sol-11_2-vbox-disk1.vmdk (Normal, 31.25 GB)

Audio

Disabled

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

Disabled

Shared folders

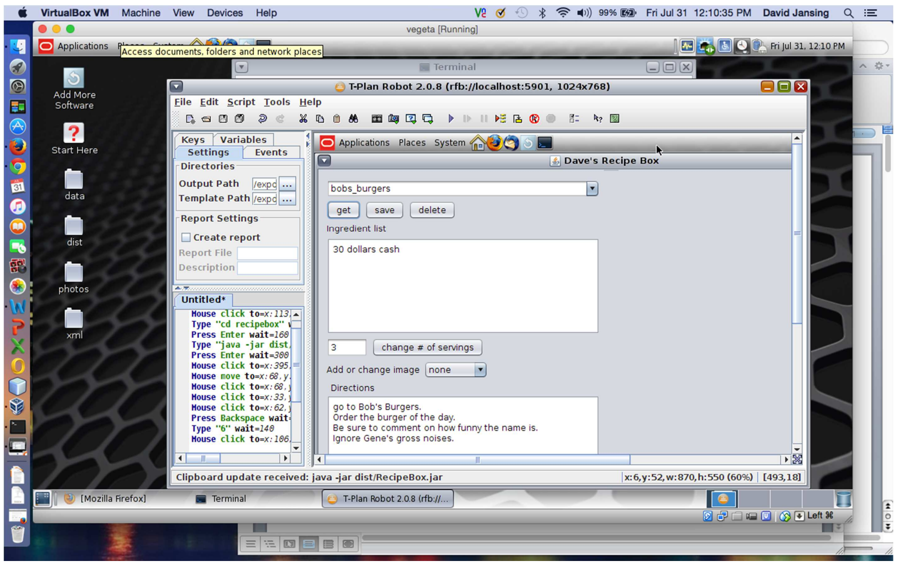
Figure 5. The VirtualBox VM manager.

6.2 T-Plan Robot: T-Plan Robot, formerly known as VNC Robot, is a "capture/replay" tool, that creates test scripts by recording user inputs, such as mouse clicks and keyboard inputs, to generate test scripts. The success or failure of a test is determined by comparing a screen shot of the current test being run with on that was recorded previously. This makes Robot very flexible; any sort of program can be tested, from a web application to a Java program with Swing interface to even a command-line program. On the other hand, comparing screenshots can create a problem when the interface changes or when the output is nondeterministic. This is why black-box automated testing is not usually done for programs such as video games. A test can even fail if a terminal or GUI window appears in a different location on the screen from where it is expected. Some effort should be taken to ensure that the environment at the start of the test is the same for each run.

Robot attempts to mitigate this by allowing the tester to set a tolerance level for how much of the captured screenshot may differ from the master file and still register a passing test. This doesn't alter the fact that the test itself will eventually need to be re-recorded if the pixel comparisons drop significantly from 100%, but it will allow the tester to report that the test passed with a reasonable level of confidence.

Because it relies on VNC to operate, it may not be suitable for all operating platforms, despite having been written in Java. For example, there are no VNC clients, at least none that T-Plan has successfully tested, that are compatible with both Robot and later versions of Mac OS X (My 2011 model MacBook Pro is currently running Yosemite). TPlan does, however, guarantee compatibility with Tight VNC for Windows, so long as the program under test is on a different Windows machine, since Windows does not support multiple desktops on the same machine.

Figure 6 shows VirtualBox running Solaris 11 on the MacBook, with the Robot program running inside it, running a test on the RecipeBox program I wrote and will describe further in Chapter 7.



```
Mouse click to=x:113,
Type "cd recipebox"
Press Enter wait=160
Type "java -jar dist,
Press Enter wait=300
Mouse click to=x:395,
Mouse move to=x:68,y,
Mouse click to=x:68,
Mouse click to=x:33,
Mouse click to=x:62,
Press Backspace wait
Type "6" wait=140
Mouse click to=x:106,
```

bobs_burgers

Ingredient list

30 dollars cash

3

Add or change image: none

Directions

go to Bob's Burgers.
Order the burger of the day.
Be sure to comment on how funny the name is.
Ignore Gene's gross noises.

Clipboard update received: java -jar dist/RecipeBox.jar   x:6,y:52,w:870,h:550 (60%)   [493,18]

Figure 6. VirtualBox running in Mac OSX, running Solaris 11, which is running T-Plan Robot on a small Java test program.

The test scripts that Robot generates are proprietary, that is to say the coding language is not readily familiar to a plurality of software developers the way Perl or Ruby might be. They are nonetheless easy to figure out, as shown in Figure 7.

```
Mouse move to=x:113,y:239 wait=100
Mouse click to=x:113,y:239 wait=100
Type "cd recipebox" wait=100
Press Enter wait=100
Type "java -jar dist/RecipeBox.jar" wait=100
Press Enter wait=300
Mouse click to=x:395,y:83 wait=100
Mouse move to=x:68,y:139 wait=100
Mouse click to=x:68,y:139 wait=100
Mouse click to=x:33,y:110 wait=100
Mouse click to=x:62,y:308 wait=100
Press Backspace wait=100
Type "6" wait=100
Mouse click to=x:106,y:314 wait=100
```

Figure 7. A test script generated by T-Plan Robot.

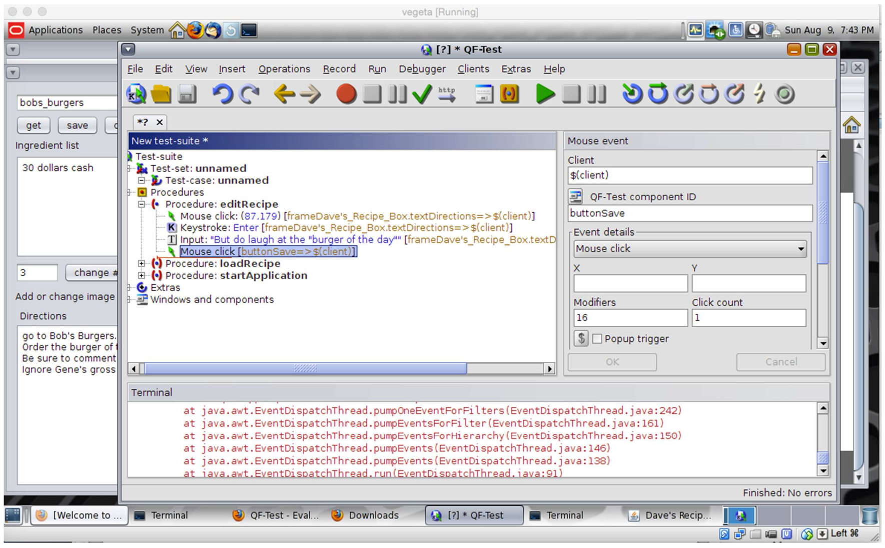
Since every movement of the mouse, backspaces due to keyboarding mistakes, sitting around thinking about what to do next, and on and on, gets captured and reflected in the generated script, some cleanup will have to be done afterwards to optimize it. Most of the wait variables in Figure 7 can be either removed or reduced, multiple mouse moves reduced to just one, and so on. In Figure 7, I shortened most of the waits to just what is needed for the script to run properly. The proprietary test scripts can also be converted to Java code, which is not limited by the functionality of T-Plan’s proprietary scripting language. Figure 8 shows the same test script as a Java class.

```
package recipeboxTest;

import com.tplan.robot.ApplicationSupport;
import com.tplan.robot.AutomatedRunnable;
import com.tplan.robot.scripting.DefaultJavaTestScript;
import com.tplan.robot.scripting.JavaTestScript;
import java.awt.Point;
import java.awt.Rectangle;
import java.io.File;
import java.io.IOException;

public class MyTest extends DefaultJavaTestScript implements
JavaTestScript {

  public void test() {
      try {
        mouseMove(new Point(113, 239), "100");
        mouseClick(new Point(113, 239), "100");
        type("cd recipebox", "100");
        press("Enter", "100");
        type("java -jar dist/RecipeBox.jar", "100");
        press("Enter", "300");
        mouseClick(new Point(395, 83), "100");
        mouseMove(new Point(68, 139), "100");
        mouseClick(new Point(68, 139), "100");
        mouseClick(new Point(33, 110), "100");
        mouseClick(new Point(62, 308), "100");
        press("Backspace", "100");
        type("6", "140");
        mouseClick(new Point(106, 314), "100");
        mouseMove(new Point(866, 38), "100");
        mouseClick(new Point(866, 38), "100");
        screenshot(new File("screenshot_bob.bmp"), new
Rectangle(11, 61, 430, 519));
        mouseMove(new Point(478, 233), "100");
        mouseClick(new Point(478, 233), "100");
        typeLine("cd", "100");
      } catch (IOException ex) {
        ex.printStackTrace();
      }
  }

  public static void main(String args[]) {
      MyTest test = new MyTest();
      ApplicationSupport robot = new ApplicationSupport();
      AutomatedRunnable t = robot.createAutomatedRunnable(test,
"javatest", args, System.out, false);
      new Thread(t).start();
  }
}
```

Figure 8. A Java program generated by T-Plan Robot based on its own test code. [7]

6.3 QF-Test: QF-Test is another kind of "capture/replay" tool that is more sophisticated than T-Plan Robot. It also includes a graphic programming environment in which keystrokes and mouse clicks are recorded to produce a script. The script can then be edited in the environment itself, in which the script is shown to the test developer as an easily read list of test steps. This is useful because the actual test script used to execute the test consists of a sizeable XML document that can be tricky to edit in its raw form. The syntax of the commands shown on screen within this environment are straightforward enough to be easily understood by someone with little programming experience. Once the test developer gains experienced with the tool, there is a large set of features for making more sophisticated and useful test scripts.

Although QF-Test is a commercial product for which licenses can be quite costly (see Chapter 4), a 30 day evaluation license is available for just about anyone with a professional interest. I installed QF-Test on my Solaris 11 VM and was recording tests within a few minutes. Figure 9 shows an instance of QF-Test running in the virtual Solaris 11 environment.



```
Test-suite
  Test-set: unnamed
    Test-case: unnamed
  Procedures
    Procedure: editRecipe
      Mouse click: (87,179) [frameDave's_Recipe_Box.textDirections=>$(client)]
      Keystroke: Enter [frameDave's_Recipe_Box.textDirections=>$(client)]
      Input: "But do laugh at the "burger of the day"" [frameDave's_Recipe_Box.textD
      Mouse click [buttonSave=>$(client)]
    Procedure: loadRecipe
    Procedure: startApplication
  Extras
  Windows and components
```

Mouse event
Client: $(client)
QF-Test component ID: buttonSave
Event details: Mouse click
Modifiers: 16
Click count: 1

```
        at java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:242)
        at java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:161)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:150)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:146)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:138)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:91)
```

Finished: No errors

Figure 9. QF Test running in Solaris 11 on VirtualBox

In addition to recording mouse movements, clicks, and keyboard activity, it also tracks references to the actual components, buttons, checkboxes, text boxes, and so on. Instead of recording a step that says, "move the mouse to position 325,125 and click," the same command in QF-Test might say "Mouse Click: (325, 125) [frame TestApplication.textBox => $(client)]." As long as some future software release didn't change the name of the aforementioned text box or reduce the size of the box smaller than 325x125 pixels, the test should withstand GUI changes without needing to re-record the test.

Being able to make use of the testing software's native language helps by allowing edits to the recorded test procedure, pruning out unneeded steps introduced during recording to make the test more efficient and easier to maintain. For example, hard-coded data could be pulled a text file or generated by a script. Selectable items can be referenced by name rather than by index, which can change every time the database containing the list of items changes. With practice, one would only need to re-record the part of the test that was causing trouble, and substitute the good lines for the bad lines.

The QF-Test work environment hides a great deal of complexity from the user. It saves tests as xml files that can easily reach an unwieldy size if one were to try editing it directly. This is because the test not only needs to know what to do, it also needs to know what all the various controls are called, what type of controls they are, their dimensions in some cases, . A saved xml file corresponding to the set of test steps such as the one step shown in Figure 10, for example, runs over 190 lines and about 12 KB. One could possibly edit it directly, but that would take a lot more work and know-how compared to opening the file in the GUI work environment and editing it from there. Figure 10 shows the XML entry for just one button one the GUI, thus giving us an idea just how much data QF-Test keeps on the system being tested.

```
<ComponentStep class="Button" classcount="14" classindex="12"
```

```
      feature="change # of servings" height="27"
      id="buttonChange___of_servings" uid="_M" width="161"
      x="83" y="267">
      <extrafeature name="qfs:class" negate="false" regexp="false"
      state="0">javax.swing.JButton</extrafeature>
      <extrafeature name="qfs:genericclass" negate="false"
      regexp="false"
      state="0">Button</extrafeature>
      <extrafeature name="qfs:label" negate="false" regexp="false"
      state="1">change # of servings</extrafeature>
      <extrafeature name="qfs:systemclass" negate="false" regexp="false"
      state="0">javax.swing.JButton</extrafeature>
</ComponentStep>
```

Figure 10. The XML entry used by QF-Test for the button in RecipeBox that adjusts recipes for varying serving sizes.

Chapter 7. Exercise 2 – Homegrown Test Suites

In addition to the demonstration test cases I created using commercial testing tools, I also created my own test suites in Java. For these test suites, I used the JUnit package, which is included in most popular development environments, including NetBeans. One suite was created based on a list of requirements described in the appendices. I would write a Java class to do a particular task, outlined in the requirements document, which I have included in Appendix A. Since these tests map to the list of requirements, and were written after the code that they test, they constitute a set of regression tests of the sort that the test team might use to ensure that new changes and functionality do not break any of the existing functionality.

The other was a set of unit tests generated by a wizard included with NetBeans, The wizard generates one test method as a shell for each method in RecipeBox that I fleshed out afterward with input and expected output values in order to achieve a reasonable level of coverage. I decided to combine some of the tests in the case where a pair of mutator (setter) and accessor (getter) methods were being tested, since in order to test a getter, one must first run the setter anyway. Otherwise, I maintained the one method – one test principle of unit testing. These would constitute my unit test suite that I would run whenever I needed to test some new or changed code. The idea that the method under test did not necessarily need to be implemented to create the unit test for it became immediately evident, and it helped a great deal with ensuring that the method would do what it was supposed to do.

I also solved the problem with a command-line based test exercising functions in a GUI. Since a button in Java has a doClick() method, there isn't much that a test developer cannot do with a test class. Java will let the test programmer create the GUI object without actually putting the GUI on the screen, and from there it is merely a matter of calling the GUI's methods. Figure 11 shows a test method from the suite I developed for RecipeBox:

```
public void testSetServings()
{
   rb.recipeList.setSelectedItem("bobs_burgers");
   rb.getRecipeButton.doClick();
   assertEquals("bobs_burgers",
            rb.recipeList.getSelectedItem());
   rb.servingsBox.setText("6");
   rb.servingsButton.doClick();
   assertEquals("60.00 dollars cash\n",
             rb.ingredientListBox.getText());
}
```

Figure 11. A test method in which both Java GUI methods and JUnit methods are used.

NetBeans will display an error message saying the GUI elements have private access and suggest that the tester make them "package private," that is available to any class or method in the same package, in order for this to work. The problem is, this subverts the access protection build into the program. O'Reilly Media has an article on their On Java website suggesting that there is a way around private access using the setAccessible() method in the java.lang.reflect.Field package. What the tester would need to do is something like in Figure 12.[28]

```
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

public class myTest
{
      public myTest
      {
            try
            {
                  final Field[] Fields =
                  MyClass.class.getDeclaredFields();
                  for (Field f: Fields)
                  {
                        Modifier[] modifiers = f.getModifiers();
                        for (Modifier m : modifiers)
                        if (isPrivate(m))
                        {
                              f.setAccessible(true);
                              break;
                        }
                  }
            }
            catch (Exception ex)
            {
                  System.out.println(ex.getMessage());
            }
      }

      // do your testing
}
```

Figure 12. An example of using the setAccessible() method in the Field class.

The way I got access to the private GUI elements was to include accessor methods, or "getters" for the GUI elements in my class, which accomplishes the same thing without having to do an end run around access control or introducing unneeded complexity to the tests. Figure 13 shows what would be a better way of exercising the GUI:

```
public void testSetServingsWithAccessorMethods()
```

```
{
   rb.getRecipeList().setSelectedItem("bobs_burgers");
   rb.getRButton().doClick();
   assertEquals("bobs_burgers",
           rb.getRecipeList().getSelectedItem());
   rb.getServingsBox().setText("6");
   rb.getServingsButton().doClick();
   assertEquals("60.00 dollars cash\n",
            rb.getIngredientListBox().getText());
}
```

Figure 13. A method that uses accessor methods to manipulate the GUI instead of setting these items “package private”.

In this way, it is possible to set up a comprehensive automated test suite that covers all requirements without spending a lot of money for a capture/playback test tool like QF-Test to do testing through the GUI, so long as an experience programmer is available to create and maintain the test classes. To make this work, either the tester would need to have access to the main source code, or someone on the development team would need to write the accessor methods. Depending on the development process and the level of cooperation between the test team and the development team, this might fall under the heading of “testability,” with the accessor methods being required as part of that process and automatically created as the code is written.

8. Conclusions

While automation of some kinds of software testing can dramatically increase the productivity and effectiveness of software development projects, rarely is it found that 100% of requirements can be covered by automated tests. Some testing is still best done by hand because human testers are far more flexible and creative than any computer program. Automation frees human testers to delve more deeply into the system be relieving them from having to do repetitive tasks for which the use of automation is more appropriate.

With a little cooperation between developers and testers, software code can be made more testable. Coding for testability is a key consideration for programmers working in an Agile development environment. For example, in my practical exercise with the Recipe Box program, I demonstrated how GUI components that are declared to be private class members may still be tested with a homegrown test script without violating key principles of object-oriented programming or resorting to questionable workarounds.

Automated software testing is a software development project in its own right, and should be treated as such, to include all of the usual software development processes, such as version control, code reviews, coding standards, and so on. Suggestions were also made in this paper on how the costs associated with automation efforts may be minimized.

There are many pre-packaged tools available to aid testers in the creation of test scripts. Some of them from commercial venders can be expensive, while there are other tools that are open-source and therefore free. In fact, the open-source community is about as capable as commercial vendors of producing high-quality software testing tools. I also found that creating test scripts in-house can be a worthwhile endeavor if the person creating them is well-versed in both testing and software development. With a little foresight, even graphic interfaces can be tested using command-line tools, as I have demonstrated.

Appendix A – The Requirements Document for Recipe Box. [13]

Recipe Box Requirements Document (version 1.0)
Project: Recipe Box
Date(s): March 1st, 2015
Prepared by: David Jansing

Document status: __Draft ___ Proposed __ Validated X Approved

1. Introduction

This document contains the system requirements for Recipe Box. These requirements have been derived mainly from discussions with my wife about the project. It is based on a template provided by the Center for Distributed Learning, California State University. Most companies, including my own, have a standard format for the requirements document. This is the one that I will be using for this project.

1.1 Purpose of This Document

This document is intended to guide development of Recipe Box. It would ordinarily go through several stages during the course of the project:

1. **Draft:** The first version, or draft version, is compiled after requirements have been discovered, recorded, classified, and prioritized.
2. **Proposed:** The draft document is then proposed as a potential requirements specification for the project. The proposed document should be reviewed by several parties, who may comment on any requirements and any priorities, either to agree, to disagree, or to identify missing requirements. Readers include end-users, developers, project managers, and any other stakeholders. The document may be amended and reproposed several times before moving to the next stage.
3. **Validated:** Once the various stakeholders have agreed to the requirements in the document, it is considered validated.
4. **Approved:** The validated document is accepted by representatives of each party of stakeholders as an appropriate statement of requirements for the project. The developers then use the requirements document as a guide to implementation and to check the progress of the project as it develops.

For purposes of this demonstration project, let us assume that the requirements document has been approved by all concerned. Approval allows me to start work on Recipe Box.

1.2 How to Use This Document

I expect that this document will be used by people with different skill sets. This section explains which parts of this document should be reviewed by various types of readers.

Types of Reader
This document is aimed at any participants who might have a stake in this project. These include Java programmers, graphic designers, technical writers, testers, end users, and project managers. End users can probably skip to Section 3.

Technical Background Required
This document is not intended to be overly technical, that is, a deep knowledge of Java should not be required to understand it. However, a fundamental understanding of computer science would be helpful.

1.3 Scope of the Product
Recipe Box is a platform-independent software project for cataloging, displaying, and manipulating recipes, like those kept on 3x5 cards in someone's kitchen. It provides a convenient means of keeping them handy and available.

1.4 Business Case for the Product
This product's purpose is to provide a means of demonstrating various automated testing techniques, thus it is not an actual commerical product. It will, however, be useful enough to actually use as a recipe storage program, if desired.

Recipe Box will be simple enough for even a beginning computer user to use effectively. Instead of having a big pile of clippings, 3x5 cards, or recipes printed from the Internet, the user will have a single location from which to store, browse, recall, manipulate, and create recipes.

1.5 Overview of the Requirements Document
This document is comprised of the following parts:
Section 2: A general description of the project.
Section 3: A list of user, system, and interface requirements.
Section 4: A glossary of terms used in the document.
Section 5: A reference citing the source material for the format and content of this document.

2. General Description
This section will give the reader an overview of the project, including why it was conceived, what it will do when complete, and the types of people we expect will use it. We also list constraints that were faced during development and assumptions we made about how we would proceed.

2.1 Product Perspective
This is a somewhat less than trivial, but small software project developed in order to demonstrate test automation techinques. It is nevertheless functional in accordance with most of the requirements listed below. Those requirements that

2.2 Product Functions
The essential parts of this system are:

- A data area where the recipe data are kept.
- A GUI for recalling, editing, and deleting the data. This consists of:
    - A pull-down list of the recipe titles.
    - A collection of buttons for manipulating data.
    - A text box to display the ingredients list.
    - A text box to display the directions for preparation.
    - A text box and button to increase or decrease the number of intended servings, thus scaling the recipe as needed.

2.3 User Characteristics
We would expect this product to be enjoyed by anyone with a love of cooking and the need for a system by which one might save, maintain, and otherwise manipulate a "book" of recipes. We do not expect that using Recipe Box would require any more technical expertise than that required to use a text editor.

2.4 General Constraints
None noted.

2.5 Assumptions and Dependencies
None noted.

3. Specific Requirements
This section of the document lists specific requirements for Recipe Box. Requirements are divided into the following sections:

1. User requirements. These are requirements written from the point of view of end users, usually expressed in narrative form.
2. System requirements. These are detailed specifications describing the functions the system must be capable of doing.
3. Interface requirements. These are requirements about the user interface, which may be expressed as a list, as a narrative, or as images of screen mock-ups.

3.1 User Requirements
1. The recipe data are to be kept in a data directory, in a format that can be interpreted by the system.
2. Recalling, editing, and deleting the data shall be controlled by a graphic interface which shall consist of:
    a. A pull-down list of the recipe titles.
    b. A collection of buttons for manipulating data.
    c. A text box to display the ingredients list.
    d. A text box to display the directions for preparation.

    e. A text box and button to increase or decrease the number of intended servings.

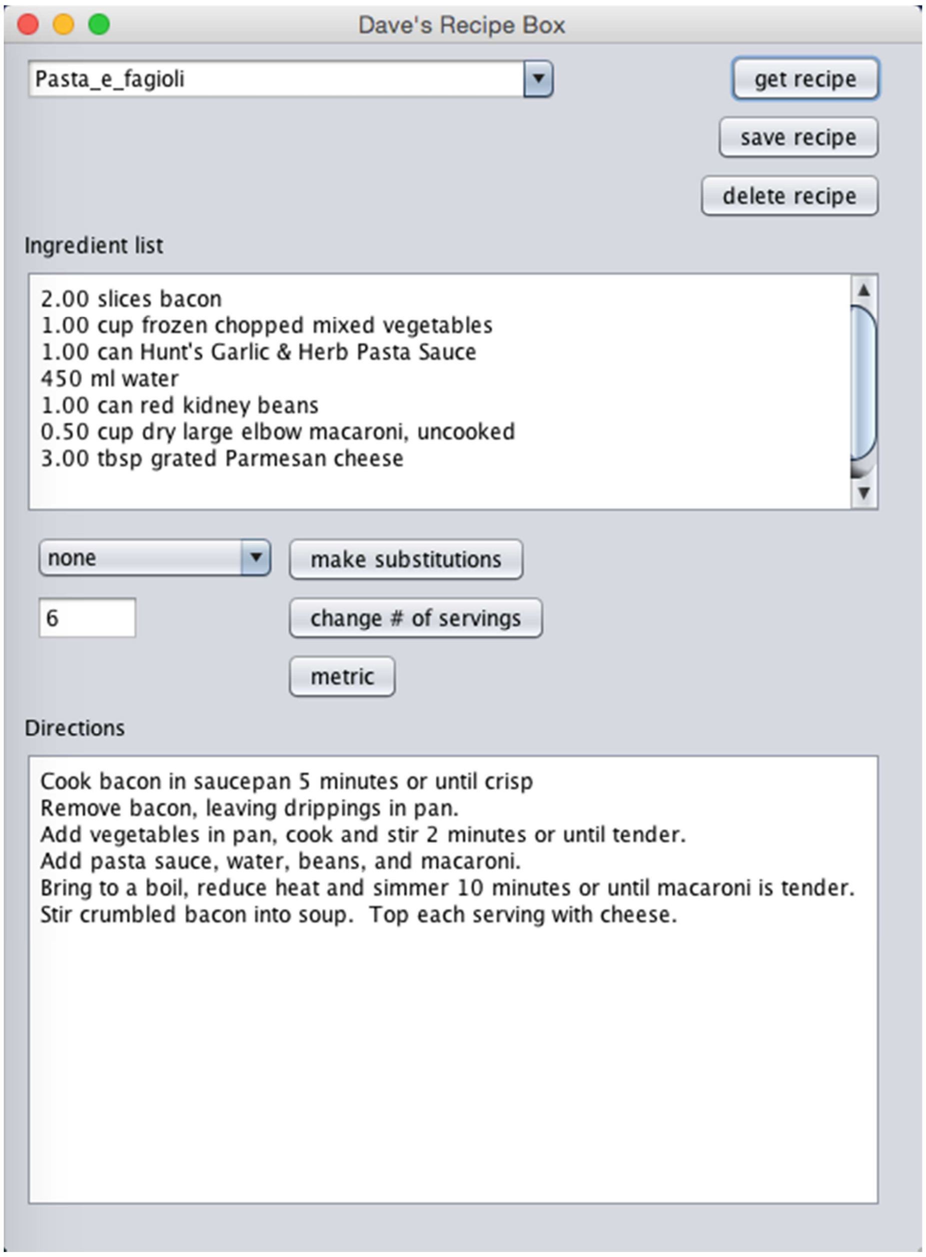
3. It shall be possible for the end user to enter new recipes into the system.
4. It shall be possible for the end user to edit an existing recipe.
5. It shall be possible for the end user to adjust ingredient quantities for varying number of servings.
6. It shall be possible to delete a recipe.
7. Quantities consisting of non-whole numbers shall be expressed in fractions in the user interface.
8. Quantities consisting of non-whole numbers shall nevertheless be stored in the system as decimal quantities, regardless of the measurement system used.
9. During the servings adjustment, large numbers of smaller units shall be automatically converted to larger units. For example, there are 16 tablespoons in ¼ cup.
10. Fractional quantities should be congruent to those found in a typical set of kitchen measuring cups (1/4, 1/3, 1/2, 1)
11. It should be made possible to convert measurements to and from U.S. and metric units.
12. If a graphic of the finished dish is available, it should be displayable by the system.

3.2 System Requirements:

Recipe Box should be usable on any machine on which the Java Runtime Environment, version 1.7.0_11 or higher, has been installed.

3.3 Interface Requirements

A preliminary screen shot of the GUI design is provided below:



Ingredient list

```
2.00 slices bacon
1.00 cup frozen chopped mixed vegetables
1.00 can Hunt's Garlic & Herb Pasta Sauce
450 ml water
1.00 can red kidney beans
0.50 cup dry large elbow macaroni, uncooked
3.00 tbsp grated Parmesan cheese
```

6

Directions

```
Cook bacon in saucepan 5 minutes or until crisp
Remove bacon, leaving drippings in pan.
Add vegetables in pan, cook and stir 2 minutes or until tender.
Add pasta sauce, water, beans, and macaroni.
Bring to a boil, reduce heat and simmer 10 minutes or until macaroni is tender.
Stir crumbled bacon into soup.  Top each serving with cheese.
```

There is a pull-down box that holds the recipe titles, along with buttons for getting, editing, and deleting recipes. There is a box for the ingredient list, a pull-down box for suggested ingredient substitutions, a box and button for adjusting the recipe for a desired number of servings, and a box for the cooking directions.

4. Glossary

Pull-down box – a JComboBox
Text box – a JTextField or JTextArea, as applicable
Button – a JButton or jToggleButon, as applicable

Ingredient – this normally consists of a quantity, a unit of measure (optional), and an item.
Serving – this is an arbitrary measure, determined by the recipe author, of how many people a recipe is intended to serve. It also acts as a means to scale a recipe up or down.
Directions – cooking directions, which is a free-form string.

5. References

Rachel S. Smith, *Writing a Requirements Document: Workshop Materials*, Center for Distributed Learning, California State University [13]

Appendix B – Test design document for Recipe Box [14] (MS Excel document).

This is intended to map testing requirements to user requirements, as shown in Appendix A, Section 3.1. Not everything in the requirements list has been implemented fully in the actual Recipe Box program. This is intentional, since it should be demonstrable for the test suite to indicate at least some actual failures as well as successful tests.

Test case ID:	TC_01
Test case description:	Browse the recipe list
Preconditions:	No recipe has yet been selected.
Requirements tested:	2

step number	**step description**	**test data**	**expected result**
1	Click the arrow next to the recipe list (where it should say, "Find a recipe")	The list of recipes, as provided by the development team.	The list of recipes should appear in a pull-down list.
2	Click "Pasta e fagioli" to select it.		"Pasta e fagioli" should appear in the box.
3	Click the "get recipe" button		The ingredient list, the number of servings, and the directions should appear in the appropriate boxes.
4 (Post condition)	Open a terminal window, and in the data directory, open the file for Pasta e fagioli		The information in the file should match that in the boxes.

Test case ID:	TC_02
Test case description:	Enter a new recipe
Preconditions:	No recipe has yet been selected.
Requirements tested:	3

step number	**step description**	**test data**	**expected result**
1	Click the recipe list box.		The recipe list box should be editable. There should be a cursor in the box.
2	Highlight the text in the recipe list box and delete it.		The recipe list box should be clear.
3	Enter "Beer can chicken" in the recipe list box.		"Beer can chicken" should appear in the recipe list box.
4	Click the ingredients box and enter some text.		This box should also be editable, and the text entered should appear in the box.
5	Click the servings box and enter "4" in the box.		The number 4 should appear in the servings box.
6	Click the directions box and enter some text.		The text entered should appear in the directions box.
7	Click the save button.		A "recipe saved" message should appear.
8 (postcondition)	Open a terminal window, and do a directory listing of the data directory		There should be a new file in the the data directory for "Beer can chicken." Open this file to ensure that the recipe text has been saved.

Test case ID:	TC_03
Test case description:	Editing a recipe
Preconditions:	Test case TC_02 has already been performed successfully.
Requirements tested:	4

step number	**step description**	**test data**	**expected result**
1	Select the recipe "Beer Can Chicken" from the pull down recipe list, and press the "get recipe" button.	This recipe was created during test TC_02, which should be run first.	The recipe data for Beer Can Chicken should appear on the screen.
2	In the box of ingredients, change some of the text and click "save recipe".		The text is edited.
3 (postcondition)	In a terminal window, open the xml file for "Beer Can Chicken".		The text should have been edited to match that which is on the screen.

Test case ID:	TC_04
Test case description:	Adjusting serving size
Preconditions:	Test case TC_02 has already been performed successfully.
Requirements tested:	5

step number	**step description**	**test data**	**expected result**
1	Select the recipe "Beer Can Chicken" from the pull down recipe list, and press the "get recipe" button.	This recipe was created during test TC_02, which should be run first.	The recipe data for Beer Can Chicken should appear on the screen.
2	Click the servings box and change the number of servings to 4.		The quantities for all the ingredients are adjusted for a four-serving batch.
3 (postcondition)	Open a terminal window, and in the data directory, open the file for Beer can chicken.		Since the new/edit recipe button has not been clicked, the recipe should still be in its original state.

Test case ID:	TC_05
Test case description:	Remove a recipe from the system.
Preconditions:	Test case TC_02 has already been performed successfully.
Requirements tested:	6

step number	step description	test data	expected result
1	Click the "get recipe" button	This recipe was created during test TC_02, which should be run first.	The ingredient list, the number of servings, and the directions should appear in the appropriate boxes.
2	Click the "delete" button		The ingredient list, the number of servings, and the directions should be cleared. The current recipe title should read "Find a recipe"
3 (postcondition)	Open a terminal window, and do a directory listing of the data directory		The file for "Beer can chicken" should no longer be there.

Test case ID:	TC_06
Test case description:	Parse ingredients
Preconditions:	No recipe has yet been selected.
Requirements tested:	7,8

step number	**step description**	**test data**	**expected result**
1	Highlight the selected line on the recipe list		The list of recipes should appear in a pull-down list.
2	In place of the selected recipe, type "test_cake.xml"		"test_cake.xml" should appear in the box.
3	If any text is in the ingredient list box, highlight and delete it.		The ingredient box should be empty.
4	Type the following data in the recipe list box:	"2 1/2 cups four token flour 2 tbsp four tokenSalt 1 cup threeTokenSugar 1 whole threeTokenChicken 1/2 twoTokenSpiceRubMix 2 twoTokenEggs"	The text should appear in the ingredient list box
5	Type "1" in the servings box.		Servings should be set at 1.
6	Click the directions box and enter the following:	"This is a test recipe. It isn't supposed to make any sense"	The text entered should appear in the directions box.
7	Click the save recipe button.		A "recipe saved" message should appear.

8 (postcondition)	Inspect the xml file that was saved		The recipe, with particular attention given to the ingredient list in this case, should have been saved in the xml location, with data as it was written in the UI, with the exception that the quantities have been saved in decimal units.
9	Press delete to remove the file.		This is a tear-down step to keep the filesystem clean.

Test case ID:	TC_07
Test case description:	Convert units
Preconditions:	No recipe has yet been selected.
Requirements tested:	9

step number	**step description**	**test data**	**expected result**
1	Highlight the selected line on the recipe list		The list of recipes should appear in a pull-down list.
2	In place of the selected recipe, type "test_cake.xml"		"test_cake.xml" should appear in the box.
3	If any text is in the ingredient list box, highlight and delete it.		The ingredient box should be empty.
4	Type the following data in the recipe list box:	1 teaspoon ingredient 1 tsp ingredient 1 tablespoon ingredient 1 tbsp ingredient 1 cup ingredient 1 ounce ingredient 1 quart ingredient	The text should appear in the ingredient list box
5	Type "1" in the servings box.		Servings should be set at 1.
6	Click the directions box and enter the following:	"This is a test recipe. It isn't supposed to make any sense"	The text entered should appear in the directions box.
7	Click the save recipe button.		A "recipe saved" message should appear.

8 (postcondition #1)	Type "10" in the servings box and click the "change # of servings" box.	3 1/3 tablespoons ingredient 3 1/3 tablespoons ingredient 2/3 cups ingredient 2/3 cups ingredient 2 1/2 quarts ingredient 2/3 pounds ingredient 2 1/2 gallons ingredient	The scaled-upward recipe should read as shown in the test data column.
9 (postcondition #2)	Type "1" in the servings box and click the "change # of servings" box.		The scaled-downward recipe should read as shown in the test data column for step 4.
10	Press delete to remove the file.		This is a tear-down step to keep the filesystem clean.

Test case ID:	TC_08
Test case description:	Convert quantities
Preconditions:	No recipe has yet been selected.
Requirements tested:	10

step number	step description	test data	expected result
1	Highlight the selected line on the recipe list		The list of recipes should appear in a pull-down list.
2	In place of the selected recipe, type "test pie"		"test pie" should appear in the box.
3	If any text is in the ingredient list box, highlight and delete it.		The ingredient box should be empty.
4	Type the following data in the recipe list box:	The fractions from 1/16 to 16/16, inclusive, for the quantites, any string for the item. The unit is irrelevant.	The text should appear in the ingredient list box
5	Type "1" in the servings box.		Servings should be set at 1.
6	Click the directions box and enter the following:	"This is a test recipe. It isn't supposed to make any sense"	The text entered should appear in the directions box.
7	Click the save recipe button.		A "recipe saved" message should appear.

8 (postcondition #1)	Click the get recipe button.		The quantities shown in the ingredients list should be converted to multiples of thirds, quarters, or halves, or to the next whole number, that represent close approximations of the original non-standard fractions.
9	Press delete to remove the file.		This is a tear-down step to keep the filesystem clean.

Appendix C – Test code in support of RecipeBox

```
/*
 * Class RecipeBoxUITest
 * The shell for this class was generated by NetBeans,
 * and uses JUnit assertions to verify unit tests.
 */

package my.recipebox;

import java.util.ArrayList;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.Rule;
import org.junit.rules.ExpectedException;
/**
 *
 * @author davidjansing
 */
public class RecipeBoxUITest {
    protected static RecipeBoxUI instance;
    @Rule
    public ExpectedException thrown= ExpectedException.none();

    public RecipeBoxUITest() {
        instance = new RecipeBoxUI();
    }

    @BeforeClass
    public static void setUpClass() {
         RecipeBoxUITest uiTest = new RecipeBoxUITest();
    }

    @AfterClass
    public static void tearDownClass() {
        instance.dispose();
    }

    /**
     * Test of readXmlFile method, of class RecipeBoxUI.
     */
    @Test
    public void testReadXmlFile() {
        System.out.println("readXmlFile");
        String fileLocation = "xml/bobs_burgers";
        instance.readXmlFile(fileLocation);

        assertEquals
            (30.0,instance.ingredients.get(0).getQuantity(),0.0);
        assertEquals("",instance.ingredients.get(0).getUnit());
        assertEquals
            ("dollars cash",instance.ingredients.get(0).getItem());
        assertEquals(3,instance.getServings());
        assertEquals("go to Bob's Burgers.  \n" +
                     "Order the burger of the day.\n" +
                     "Be sure to comment on how funny the name is.\n" +
                     "Ignore Gene's gross noises.",
                     instance.getDirections());
    }

    /**
```

```
     * Test of isUnit method, of class RecipeBoxUI.
     */
    @Test
    public void testIsUnit() {
        System.out.println("isUnit");
        boolean expResult = true;
        boolean result = instance.isUnit("cups");
        assertEquals(expResult, result);
        result = instance.isUnit("tablespoons");
        assertEquals(expResult, result);
        result = instance.isUnit("gallons");
        assertEquals(expResult, result);
        result = instance.isUnit("quarts");
        assertEquals(expResult, result);
        result = instance.isUnit("ounces");
        assertEquals(expResult, result);
        result = instance.isUnit("pounds");
        assertEquals(expResult, result);
        result = instance.isUnit("teaspoons");
        assertEquals(expResult, result);
        // test expected failure
        expResult = false;
        result = instance.isUnit("barleycorns");
        assertEquals(expResult, result);
    }

    /**
     * Test of setServings method, of class RecipeBoxUI.
     */
    @Test
    public void testSetGetServings() {
        System.out.println("getServings");
        instance.getRecipeList().addItem("item");
        instance.getRecipeList().setSelectedItem("item");
        // instead of hitting the method directly,
        // I'm going to let the UI handle it.
        instance.getServingsBox().setText("5");
        instance.getEditRecipeButton().doClick();
        int result = instance.getServings();
        assertEquals(result, 5);
        // I'm going to expect that the UI will just revert
        // to what was already there if I put in a non-numeric string
        // The NumberFormatException should get handled
        thrown.expect(NumberFormatException.class);
        instance.getServingsBox().setText("meatballs");
        instance.getEditRecipeButton().doClick();
        result = instance.getServings();
        assertEquals(result, 5);
        // It should also handle the case
        // where a real number is put in
        instance.getServingsBox().setText("4.5");
        instance.getEditRecipeButton().doClick();
        result = instance.getServings();
        assertEquals(result, 4);
        // get rid of the test recipe
        instance.getDeleteRecipeButton().doClick();
    }

    @Test
    public void testGetSetImage() {
        System.out.println("getImage");
        instance.getImageList().setSelectedItem("Pasta.jpg");
        String expResult = "Pasta.jpg";
        String result = instance.getImage();
        assertEquals(result, expResult);
    }

    /**
     * Test of setDirections method, of class RecipeBoxUI.
     */
    @Test
    public void testSetGetDirections() {
        System.out.println("setDirections");
        String s = "directions";
        //RecipeBoxUI instance = new RecipeBoxUI();
        instance.setDirections(s);
        assertEquals("directions",instance.getDirections());
        //instance.dispose();
    }

    /**
     * Test of setIngredients method, of class RecipeBoxUI.
     * Method calls setIngredients method that takes an
     * ArrayList argument
     * This test includes quantities and units that should
     * convert during processing
     */
    @Test
    public void testSetIngredients() {
        System.out.println("setIngredients");
        instance.ingredients.add(new Ingredient
            (3.5,"cups","water"));
        instance.ingredients.add(new Ingredient
            (0.0625,"tablespoons","sea salt"));
        instance.ingredients.add(new Ingredient
            (2.9375,"ounces","cocoa powder"));
        instance.setIngredients();
        assertEquals("3 1/2 cups water\n" +
                     "1/4 teaspoons sea salt\n" +
                     "3 ounces cocoa powder\n",
                     instance.getIngredientListBox().getText());
    }

    /**
     * Test of addItemSafely method, of class RecipeBoxUI.
     * The point of the addItemSafely method
     * is to avoid adding a recipe to the RecipeList
     * if a recipe of that title already exists.
     */
    @Test
    public void testAddItemSafely() {
        System.out.println("addItemSafely");
        int itemCount = instance.getRecipeList().getItemCount();
        String item = "new_item";
        instance.setServings(8);
        instance.setDirections("directions");
        ArrayList<Ingredient> ingList = new ArrayList();
        Ingredient ing = new Ingredient(1.0, "cup", "water");
        ingList.add(ing);
        instance.setIngredients(ingList);
        instance.addItemSafely(item);
        // add the same recipe again
        instance.addItemSafely(item);
        // recipe should only have been added once to the list.
        assertEquals(itemCount+1,
                     instance.getRecipeList().getItemCount());
    }
```

```
/**
 * Test of getIngredients method, of class RecipeBoxUI.
 */
@Test
public void testGetIngredients() {
    System.out.println("getIngredients");
    instance.getRecipeList().setSelectedItem("bobs_burgers");
    instance.getGetRecipeButton().doClick();
    ArrayList<Ingredient> expResult = new ArrayList();
    Ingredient ing = new Ingredient(30, "", "dollars cash");
    expResult.add(ing);
    ArrayList<Ingredient> result = instance.getIngredients();
    assertEquals(result.get(0).getQuantity(),
                 expResult.get(0).getQuantity(),0.0);
    assertEquals(result.get(0).getUnit(),
                 expResult.get(0).getUnit());
    assertEquals(result.get(0).getItem(),
                 expResult.get(0).getItem());
}

/**
 * Test of convertLargerUnit method, of class RecipeBoxUI.
 */
@Test
public void testConvertLargerUnit() {
    System.out.println("convertLargerUnit");
    Ingredient i = new Ingredient();
    i.setQuantity(16);
    i.setUnit("tablespoons");
    i.setItem("some ingredient");
    instance.convertLargerUnit(i);
    assertEquals(1, i.getQuantity(),0.0);
    assertEquals("cups",i.getUnit());
    // There are 16 tablespoons in a cup,
    // but only four cups in a quart,
    // so test both.
    i.setQuantity(4);
    i.setUnit("cups");
    instance.convertLargerUnit(i);
    assertEquals(1, i.getQuantity(),0.0);
    assertEquals("quarts",i.getUnit());
}

/**
 * Test of convertSmallerUnit method, of class RecipeBoxUI.
 */
@Test
public void testConvertSmallerUnit() {
    System.out.println("convertSmallerUnit");
    Ingredient i = new Ingredient();
    i.setQuantity(0.125);
    i.setUnit("cups");
    i.setItem("some ingredient");
    instance.convertSmallerUnit(i);
    assertEquals(2,i.getQuantity(),0.0);
    assertEquals("tablespoons",i.getUnit());
    // Likewise, as with up-conversion, since the tbsp-cups
    // and cups-quarts conversions are different,
    // an additional test is needed for the latter
    i.setQuantity(0.25);
    i.setUnit("quarts");
    instance.convertSmallerUnit(i);
    assertEquals(1,i.getQuantity(),0.0);
    assertEquals("cups",i.getUnit());
    }

    /**
     * Test of convertToFractional method, of class RecipeBoxUI.
     */
    @Test
    public void testConvertToFractional() {
        System.out.println("convertToFractional");
        double dec = 2.5;
        String expResult = "2 1/2";
        String result = instance.convertToFractional(dec);
        assertEquals(result, expResult);
        // Also test the instance where the quantity is less than one
        dec = 0.5;
        expResult = "1/2";
        result = instance.convertToFractional(dec);
        assertEquals(result, expResult);
    }

    /**
     * Test of convertToDecimal method, of class RecipeBoxUI.
     * convertToDecimal doesn't concern itself with the entire number,
     * just the fractional part,returning an integer representing the
     * quantity right of the decimal point.
     */
    @Test
    public void testConvertToDecimal() {
        System.out.println("convertToDecimal");
        String fraction = "1/2";
        String expResult = "5";
        String result = instance.convertToDecimal(fraction);
        assertEquals(result, expResult);
    }

}
```

```
/*
 * This version of the unit tests uses TestNG
 * instead of JUnit
 */
package my.recipebox;

import java.util.ArrayList;
import static org.testng.Assert.*;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

/**
 *
 * @author davidjansing
 */
public class RecipeBoxUINGTest {

    public RecipeBoxUINGTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    /**
     * Test of readXmlFile method, of class RecipeBoxUI.
     */
    @Test
    public void testReadXmlFile() {
        System.out.println("readXmlFile");
        String fileLocation = "xml/bobs_burgers";
        RecipeBoxUI instance = new RecipeBoxUI();
        instance.readXmlFile(fileLocation);
        assert 30.0 == instance.ingredients.get(0).getQuantity();
        assert "".equals(instance.ingredients.get(0).getUnit());
        assert "dollars cash".equals
                  (instance.ingredients.get(0).getItem()));
        instance.dispose();
    }

    /**
     * Test of isUnit method, of class RecipeBoxUI.
     */
    @Test
    public void testIsUnit() {
        System.out.println("isUnit");
        String u = "cups";
        RecipeBoxUI instance = new RecipeBoxUI();
        boolean expResult = true;
        boolean result = instance.isUnit(u);
        assertEquals(result, expResult);
        String v = "barleycorns";
        expResult = false;
        result = instance.isUnit(v);
        assertEquals(result, expResult);
        instance.dispose();
    }
```

```
/**
 * Test of getServings method, of class RecipeBoxUI.
 * I combined these because they're a pair of setter/getter methods
 */
@Test
public void testGetSetServings() {
    System.out.println("getServings");
    RecipeBoxUI instance = new RecipeBoxUI();
    instance.setServings(5);
    int expResult = 5;
    int result = instance.getServings();
    assertEquals(result, expResult);
    instance.dispose();
}


/**
 * Test of getImage, setImage method, of class RecipeBoxUI.
 * I combined these because they're a pair of setter/getter methods
 */
@Test
public void testGetSetImage() {
    System.out.println("getImage");
    RecipeBoxUI instance = new RecipeBoxUI();
    instance.setImage("Pasta.jpg");
    String expResult = "Pasta.jpg";
    String result = instance.getImage();
    assertEquals(result, expResult);
    instance.dispose();
}

/**
 * Test of setDirections method, of class RecipeBoxUI.
 */
@Test
public void testSetGetDirections() {
    System.out.println("setDirections");
    String s = "directions";
    RecipeBoxUI instance = new RecipeBoxUI();
    instance.setDirections(s);
    assert "directions".equals(instance.getDirections());
    instance.dispose();
}

/**
 * Test of setIngredients method, of class RecipeBoxUI.
 */
@Test
public void testSetIngredients_0args() {
    System.out.println("setIngredients");
    RecipeBoxUI instance = new RecipeBoxUI();
    Ingredient ing = new Ingredient(3, "cups", "water");
    instance.ingredients.add(ing);
    instance.setIngredients();
    assert 3.0 == instance.ingredients.get(0).getQuantity();
    assert "cups".equals(instance.ingredients.get(0).getUnit());
    assert "water".equals(instance.ingredients.get(0).getItem());
    instance.dispose();
}

/**
 * Test of setIngredients method, of class RecipeBoxUI.
 */
@Test
```

```
public void testSetIngredients_ArrayList() {
    System.out.println("setIngredients");
    ArrayList<Ingredient> ing = new ArrayList();
    RecipeBoxUI instance = new RecipeBoxUI();
    Ingredient i = new Ingredient(3, "cups", "something or other");
    ing.add(i);
    instance.setIngredients(ing);
    ArrayList<Ingredient> result = instance.getIngredients();
    assert 3.0 == result.get(0).getQuantity();
    assert "cups".equals(result.get(0).getUnit());
    assert "something or other".equals(result.get(0).getItem());
    instance.dispose();
}
/**
 * Test of addItemSafely method, of class RecipeBoxUI.
 */
@Test
public void testAddItemSafely() {
    System.out.println("addItemSafely");
    String item = "new item";
    RecipeBoxUI instance = new RecipeBoxUI();
    instance.setServings(8);
    instance.setDirections("directions");
    ArrayList<Ingredient> ingList = new ArrayList();
    Ingredient ing = new Ingredient(1.0, "cup", "water");
    ingList.add(ing);
    instance.setIngredients(ingList);
    instance.addItemSafely(item);
    instance.getRecipeList().setSelectedItem(item);
    assert "new item".equals
              (instance.getRecipeList().getSelectedItem());
    instance.dispose();
}

/**
 * Test of getIngredients method, of class RecipeBoxUI.
 */
@Test
public void testGetIngredients() {
    System.out.println("getIngredients");
    RecipeBoxUI instance = new RecipeBoxUI();
    instance.getRecipeList().setSelectedItem("bobs_burgers");
    instance.getGetRecipeButton().doClick();
    ArrayList<Ingredient> expResult = new ArrayList();
    Ingredient ing = new Ingredient(30, "", "dollars cash");
    expResult.add(ing);
    ArrayList<Ingredient> result = instance.getIngredients();
    assertEquals(result.get(0).getQuantity(),
                 expResult.get(0).getQuantity());
    assertEquals(result.get(0).getUnit(),
                 expResult.get(0).getUnit());
    assertEquals(result.get(0).getItem(),
                 expResult.get(0).getItem());
    instance.dispose();
}

/**
 * Test of convertLargerUnit method, of class RecipeBoxUI.
 */
@Test
public void testConvertLargerUnit() {
    System.out.println("convertLargerUnit");
    Ingredient i = new Ingredient();
    i.setQuantity(16);
```

```
        i.setUnit("tablespoons");
        i.setItem("some ingredient");
        RecipeBoxUI instance = new RecipeBoxUI();
        instance.convertLargerUnit(i);
        assert i.getQuantity() == 1;
        assert "cups".equals(i.getUnit());
        instance.dispose();
    }

    /**
     * Test of convertSmallerUnit method, of class RecipeBoxUI.
     */
    @Test
    public void testConvertSmallerUnit() {
        System.out.println("convertSmallerUnit");
        Ingredient i = new Ingredient();
        i.setQuantity(0.125);
        i.setUnit("cups");
        i.setItem("some ingredient");
        RecipeBoxUI instance = new RecipeBoxUI();
        instance.convertSmallerUnit(i);
        assert i.getQuantity() == 2;
        assert "tablespoons".equals(i.getUnit());
        instance.dispose();
    }

    /**
     * Test of convertToFractional method, of class RecipeBoxUI.
     */
    @Test
    public void testConvertToFractional() {
        System.out.println("convertToFractional");
        double dec = 2.5;
        RecipeBoxUI instance = new RecipeBoxUI();
        String expResult = "2 1/2";
        String result = instance.convertToFractional(dec);
        assertEquals(result, expResult);
        instance.dispose();
    }

    /**
     * Test of convertToDecimal method, of class RecipeBoxUI.
     */
    @Test
    public void testConvertToDecimal() {
        System.out.println("convertToDecimal");
        String fraction = "1/2";
        RecipeBoxUI instance = new RecipeBoxUI();
        String expResult = "5";
        String result = instance.convertToDecimal(fraction);
        assertEquals(result, expResult);
        instance.dispose();
    }

}
```

```
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 *
 * The recipe used in this test was transcribed from the label
 * of a can of Hunt’s Garlic & Herb pasta sauce. [32]
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;
import static org.junit.internal.matchers.StringContains.containsString;
import static org.junit.matchers.JUnitMatchers.both;
/**
 *
 * @author davidjansing
 */
public class Test1 {

    private static RecipeBoxUI rb;
    // components under test
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
    }

    private void tearDownClass() {
       rb.dispose();
    }


    // This test simply pulls a recipe and makes sure
    // everything is there.
    public void testSetIngredients()
    {
        // Test for null - the ingredient list, servings,
        // and directions boxes should all be empty
        // at the start of the test.
        assertNull(ingredientListBox.getText(), null);
        assertNull(directionsBox.getText(), null);
        assertNull(servingsBox.getText(), null);
        // Select a recipe that is assumed to exist
        rb.getRecipeList().setSelectedItem("Pasta e fagioli");
        rb.getGetRecipeButton().doClick();
        // assertEquals tests for an exact match of what
        // you're checking, similar to String.equals()
        assertEquals("2 slices bacon\n" +
                    "1 cups frozen chopped mixed vegetables\n" +
                    "1 can Hunt's Garlic & Herb Pasta Sauce\n" +
                    "2 cups water\n" +
                    "1 can red kidney beans\n" +
                    "1/2 cups dry large elbow macaroni, uncooked\n" +
                    "1/4 cups grated Parmesan cheese\n",
                    ingredientListBox.getText());

        assertEquals("6",servingsBox.getText());
        // running assertTrue on the method I wrote to test
        // whether a string represents a number
        assertTrue(isNumber(servingsBox.getText()));
        // I probably could have avoided some headache had I known
        // to use this to do the equivalent of String.contains() rather
        // than trying to match the entire text.
        assertThat(directionsBox.getText(),
                   both(containsString("Cook bacon in saucepan 5 “ +
                                       “minutes or until crisp")).
                   and(containsString("Remove bacon, leaving “ +
                                      “drippings in pan.")));
        assertThat(directionsBox.getText(),
                   containsString("Add vegetables in pan, cook and “ +
                                  “stir 2 minutes or until tender."));
        // Here's the original assertion that had to be dead-on exact
        assertEquals("Cook bacon in saucepan 5 minutes or until “ +
                      “crisp\n"+
                     "Remove bacon, leaving drippings in pan.\n"+
                     "Add vegetables in pan, cook and stir 2 minutes “ +
                     “or until tender.\n"+
                     "Add pasta sauce, water, beans, and macaroni.\n"+
                     "Bring to a boil, reduce heat and simmer 10 “ +
                     “minutes or until macaroni is tender.\n"+
                     "Stir crumbled bacon into soup.  Top each “ +
                     “serving with cheese.",
                     directionsBox.getText());
    }

    // There really isn't anything out there that compares
    // a string to a regular expression, so you pretty much
    // have to write your own.  Here's one that checks a
    // string to see if it's a number or a fraction
    private boolean isNumber (String num)
    {
        // if it's a fraction, it will have a slash in it
        if (num.indexOf("/") != -1)
        {
            String[] numberParts = num.split("/");
            for (String s : numberParts)
            {
                // try to turn it into an integer
                // if it's not possible, catch the exception
                // and return false
                try
                {
                    int i = Integer.parseInt(s);
                }
                catch(NumberFormatException e)
                {
                    return false;
                }
            }
        }
        else // it's a whole number
        {
            try
            {
                int i = Integer.parseInt(num);
            }
            catch(Exception e)
            {
                return false;
            }
        }
        return true;
    }


    /**
     * Test of main method, of class recipeBoxUI.
     */

    // You can set timeouts right there in the test
    @Test(timeout=10000)
    public void testMain() {
        System.out.println("Test1::testMain");
        setUpClass();
        testSetIngredients();
        tearDownClass();
    }
}
```

```java
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test2 {

    private static RecipeBoxUI rb;
        // components under test
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton getRecipeButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        getRecipeButton = rb.getGetRecipeButton();
    }

    private void tearDownClass() {
       recipeList.setSelectedItem("concrete_patio");
       rb.getDeleteRecipeButton().doClick();
       rb.dispose();
    }

    public void testCreateRecipe()
    {
        recipeList.addItem("concrete_patio");
        recipeList.setSelectedItem("concrete_patio");
        String ingredients = "1 bag concrete mix\n" +
                             "5 gallons water";
        String directions  = "Mix well in wheelbarrow.\n" +
                             "Pour in pre-constructed form.";
        ingredientListBox.setText(ingredients);
        directionsBox.setText(directions);
        servingsBox.setText("1");
        editRecipeButton.doClick();
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        recipeList.setSelectedItem("concrete_patio");
        getRecipeButton.doClick();
        assertEquals("1 bag concrete mix\n" +
                     "5 gallons water\n",
                     ingredientListBox.getText());
        assertEquals("Mix well in wheelbarrow.\n" +
                     "Pour in pre-constructed form.",
                     directionsBox.getText());
        assertEquals("1", servingsBox.getText());
    }

    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test2::testMain");
        setUpClass();
        testCreateRecipe();
        tearDownClass();
    }
}
```

```
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test3 {

    private static RecipeBoxUI rb;
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton getRecipeButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        getRecipeButton = rb.getGetRecipeButton();
    }

    private void tearDownClass() {
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        ingredientListBox.setText("30 dollars cash");
        directionsBox.setText("go to Bob's Burgers.  \n" +
                              "Order the burger of the day.\n" +
                              "Be sure to comment on how funny “ +
                              “the name is.\n" +
                              "Ignore Gene's gross noises.");
        servingsBox.setText("3");
        editRecipeButton.doClick();
        rb.dispose();
    }


    public void testEditRecipe()
    {
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        assertEquals("bobs_burgers",
                    recipeList.getSelectedItem());
        ingredientListBox.setText("12 euros cash");
        directionsBox.setText("Dial 555-5555 and ask for Gene.");
        editRecipeButton.doClick();
        recipeList.setSelectedItem("Pasta e fagioli");
        getRecipeButton.doClick();
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        assertEquals("12 euros cash\n",
                    ingredientListBox.getText());
        assertEquals("Dial 555-5555 and ask for Gene.",
                     directionsBox.getText());
    }

    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test3::testMain");
        setUpClass();
        testEditRecipe();
        tearDownClass();
    }
}
```

```java
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test4 {

    private static RecipeBoxUI rb;

    private void setUpClass() {
        rb = new RecipeBoxUI();
    }

    private void tearDownClass() {
       rb.dispose();
    }

    // When the decimals-to-fractions functionality is implemented,
    // this test may fail and require maintenance.
    public void testSetServings()
    {
        rb.getRecipeList().setSelectedItem("bobs_burgers");
        rb.getGetRecipeButton().doClick();
        assertEquals("bobs_burgers",
                    rb.getRecipeList().getSelectedItem());
        rb.getServingsBox().setText("6");
        rb.getServingsButton().doClick();
        assertEquals("60 dollars cash\n",
                     rb.getIngredientListBox().getText());
    }


    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test4::testMain");
        setUpClass();
        testSetServings();
        tearDownClass();
    }
}
```

```
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import java.io.File;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test5 {

    private static RecipeBoxUI rb;
        // components under test
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton getRecipeButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        getRecipeButton = rb.getGetRecipeButton();
    }

    private void tearDownClass() {
       rb.dispose();
    }

    // When the decimals-to-fractions functionality is implemented,
    // this test may fail and require maintenance.
    public void testDeleteRecipe()
    {
        // Create a minimal recipe
        recipeList.addItem("bathtub_gin.xml");
        recipeList.setSelectedItem("bathtub_gin.xml");
        ingredientListBox.setText("50 gallons alcoholic stuff");
        directionsBox.setText("mix in bathtub");
        servingsBox.setText("1");
        editRecipeButton.doClick();
        // Select another recipe
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        // Select the new recipe
        recipeList.setSelectedItem("bathtub_gin.xml");
        getRecipeButton.doClick();
        // Click "delete"
        rb.getDeleteRecipeButton().doClick();
        // Is it gone?  If not, fail the test.
        File file = new File("./xml/bathtub_gin.xml");
        if (file.exists())
            fail("File deletion failed.");
    }
    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test5::testMain");
        setUpClass();
        testDeleteRecipe();
        tearDownClass();
    }
}
```

```
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test6 {

    private static RecipeBoxUI rb;
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton getRecipeButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        getRecipeButton = rb.getGetRecipeButton();
    }

    private void tearDownClass() {
       recipeList.setSelectedItem("test_cake.xml");
       rb.getDeleteRecipeButton().doClick();
       rb.dispose();
    }

    // When the decimals-to-fractions functionality is implemented,
    // this test may fail and require maintenance.
    public void testRecipeCoverage()
    {
        // Create a nonsense recipe that tests
        // all types of ingredient strings.
        recipeList.addItem("test_cake.xml");
        recipeList.setSelectedItem("test_cake.xml");
        ingredientListBox.setText("2 1/2 cups four token flour\n" +
                                  "2 tbsp four tokenSalt\n" +
                                  "1 cup threeTokenSugar\n" +
                                  "1 whole threeTokenChicken\n" +
                                  "1/2 twoTokenSpiceRubMix\n" +
                                  "2 twoTokenEggs");

        directionsBox.setText("This is a test recipe.  It isn't “ +
                              supposed to make any sense");
        servingsBox.setText("1");
        editRecipeButton.doClick();
        // Select another recipe
        recipeList.setSelectedItem("bobs_burgers");
        getRecipeButton.doClick();
        // Select the new recipe
        // This technically meets the requirements of
        // test step 8 of test case TC_06, because
        // the text would not appear if the xml file was faulty
        recipeList.setSelectedItem("test_cake.xml");
        getRecipeButton.doClick();
        assertEquals("2 1/2 cups four token flour\n" +
                     "2 tbsp four tokenSalt\n" +
                     "1 cup threeTokenSugar\n" +
                     "1 whole threeTokenChicken\n" +
                     "1/2 twoTokenSpiceRubMix\n" +
                     "2 twoTokenEggs\n",
                     ingredientListBox.getText());
    }

    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test6::testMain");
        setUpClass();
        testRecipeCoverage();
        tearDownClass();
    }
}
```

```java
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test7 {

    private static RecipeBoxUI rb;
    // components under test
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton servingsButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        servingsButton = rb.getServingsButton();
    }

    private void tearDownClass() {
       recipeList.setSelectedItem("test cake");
       rb.getDeleteRecipeButton().doClick();
       rb.dispose();
    }

    // I probably need a specially-made recipe to get
    // full coverage of the unit conversion bits
    public void testConvertUnits()
    {
        // Create a nonsense recipe that tests all units
        recipeList.addItem("test cake");
        recipeList.setSelectedItem("test cake");
        ingredientListBox.setText("1 teaspoon ingredient\n" +
                                  "1 tsp ingredient\n" +
                                  "1 tablespoon ingredient\n" +
                                  "1 tbsp ingredient\n" +
                                  "1 cup ingredient\n" +
                                  "1 ounce ingredient\n" +
                                  "1 quart ingredient");
        directionsBox.setText("This is a test recipe.  It isn't “ +
                              ” supposed to make any sense");
        servingsBox.setText("1");
        editRecipeButton.doClick();

        servingsBox.setText("10");
        servingsButton.doClick();

        // tsp goes to tbsp, tbsp goes to cups, cups goes to qts,
        // oz goes to lbs, and qts goes to gals.
        assertEquals("3 1/3 tablespoons ingredient\n" +
                     "3 1/3 tablespoons ingredient\n" +
                     "2/3 cups ingredient\n" +
                     "2/3 cups ingredient\n" +
                     "2 1/2 quarts ingredient\n" +
                     "2/3 pounds ingredient\n" +
                     "2 1/2 gallons ingredient\n",
                    ingredientListBox.getText());
        servingsBox.setText("1");
        servingsButton.doClick();
        assertEquals("1 teaspoons ingredient\n" +
                     "1 teaspoons ingredient\n" +
                     "1 tablespoons ingredient\n" +
                     "1 tablespoons ingredient\n" +
                     "1 cups ingredient\n" +
                     "1 ounces ingredient\n" +
                     "1 quarts ingredient\n",
                    ingredientListBox.getText());
    }

    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test7::testMain");
        setUpClass();
        testConvertUnits();
        tearDownClass();
    }
}
```

```
/*
 * This version of Test7 uses JUnit theories
 */
package my.recipebox;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.*;
import org.junit.experimental.theories.DataPoint;
import org.junit.experimental.theories.Theories;
import org.junit.experimental.theories.Theory;
import org.junit.runner.RunWith;

/**
 *
 * @author davidjansing
 */
@RunWith(Theories.class)
public class Test7a {

    private static RecipeBoxUI rb;
    @DataPoint
    public static String[] tsp = {"1 teaspoons ingredient",
                                  "3 1/3 tablespoons"};
    @DataPoint
    public static String[] tbsp = {"1 tablespoons ingredient",
                                   "2/3 cups"};
    @DataPoint
    public static String[] cup = {"1 cups ingredient",
                                  "2 1/2 quarts"};
    @DataPoint
    public static String[] qt = {"1 quarts ingredient",
                                 "2 1/2 gallons"};
    @DataPoint
    public static String[] oz = {"1 ounces ingredient",
                                 "2/3 pounds"};

    // components under test
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton servingsButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        servingsButton = rb.getServingsButton();
    }

    private void tearDownClass() {
       recipeList.setSelectedItem("test cake");
       rb.getDeleteRecipeButton().doClick();
       rb.dispose();
    }

    @Theory
    public void testConvertUnits(String[] linePair)
    {
        assertThat(convertUnits(linePair), is(true));
```

```
            System.out.println(linePair[0] + " x10 equals " + linePair[1]);
        }

        // This is a "theoretical" version of the test from Test7.java
        // It's more general, but it costs more to run.
        public boolean convertUnits(String[] linePair)
        {
            boolean pass = true;
            setUpClass();
            // Create a nonsense recipe that tests all units
            recipeList.addItem("test cake");
            recipeList.setSelectedItem("test cake");
            ingredientListBox.setText(linePair[0]);
            directionsBox.setText("This is a test recipe.  It isn't “ +
                                  supposed to make any sense");
            servingsBox.setText("1");
            editRecipeButton.doClick();

            servingsBox.setText("10");
            servingsButton.doClick();

            if (!ingredientListBox.getText().contains(linePair[1]))
            {
                pass = false;
            }
            servingsBox.setText("1");
            servingsButton.doClick();

            if (!ingredientListBox.getText().contains(linePair[0]))
            {
                pass = false;
            }
            tearDownClass();
            return pass;
        }

    }
```

```
/*
 * The following numbered tests are matched to requirements
 * as described in Appendix B.  These also make use of JUnit
 * for verification of test results.
 */
package my.recipebox;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author davidjansing
 */
public class Test8 {

    private static RecipeBoxUI rb;
    private static javax.swing.JTextArea ingredientListBox;
    private static javax.swing.JTextArea directionsBox;
    private static javax.swing.JTextField servingsBox;
    private static javax.swing.JComboBox recipeList;
    private static javax.swing.JButton editRecipeButton;
    private static javax.swing.JButton getRecipeButton;

    private void setUpClass() {
        rb = new RecipeBoxUI();
        ingredientListBox = rb.getIngredientListBox();
        directionsBox = rb.getDirectionsBox();
        servingsBox = rb.getServingsBox();
        recipeList = rb.getRecipeList();
        editRecipeButton = rb.getEditRecipeButton();
        getRecipeButton = rb.getGetRecipeButton();
    }

    private void tearDownClass() {
       recipeList.setSelectedItem("test pie");
       rb.getDeleteRecipeButton().doClick();
       rb.dispose();
    }

    // Test to check proper conversion of fractions
    // to standard measuring cup sizes
    public void testConvertQuantity()
    {
        recipeList.addItem("test pie");
        recipeList.setSelectedItem("test pie");

        // 15 tbsp is 15/16 cup,
        // which should translate to 1 cup when the system
        // converts the unit upward
        ingredientListBox.setText("1/16 bags\n" +
                                  "2/16 bags\n" +
                                  "3/16 bags\n" +
                                  "4/16 bags\n" +
                                  "5/16 bags\n" +
                                  "6/16 bags\n" +
                                  "7/16 bags\n" +
                                  "8/16 bags\n" +
                                  "9/16 bags\n" +
                                  "10/16 bags\n" +
                                  "11/16 bags\n" +
                                  "12/16 bags\n" +
                                  "13/16 bags\n" +
                                  "14/16 bags\n" +
                                    "15/16 bags\n" +
                                    "16/16 bags");
        servingsBox.setText("1");
        directionsBox.setText("This is a test recipe.  It isn't supposed
" +
                                   "to make any sense");
        editRecipeButton.doClick();
        recipeList.setSelectedItem("test pie");
        getRecipeButton.doClick();
        assertEquals("< 1/4 bags\n" +
                     "< 1/4 bags\n" +
                     "1/4 bags\n" +
                     "1/4 bags\n" +
                     "1/3 bags\n" +
                     "1/3 bags\n" +
                     "1/2 bags\n" +
                     "1/2 bags\n" +
                     "1/2 bags\n" +
                     "2/3 bags\n" +
                     "2/3 bags\n" +
                     "3/4 bags\n" +
                     "3/4 bags\n" +
                     "1 bags\n" +
                     "1 bags\n" +
                     "1 bags\n",
                     ingredientListBox.getText());
    }

    /**
     * Test of main method, of class recipeBoxUI.
     */
    @Test
    public void testMain() {
        System.out.println("Test8::testMain");
        setUpClass();
        testConvertQuantity();
        tearDownClass();
    }
}
```

```
/*
 * This class contains mainly a list of test classes
 * for NetBeans to run as part of a test suite.
 */
package my.recipebox;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
/**
 *
 * @author davidjansing
 */

@RunWith(Suite.class)
@Suite.SuiteClasses({
  Test1.class,
  Test2.class,
  Test3.class,
  Test4.class,
  Test5.class,
  Test6.class,
  Test7.class,
  Test8.class
})


public class TestSuite {

}
```

Appendix D – The RecipeBox source code.

The source code for RecipeBox comprises :

- The main RecipeBox class, which handles all the data input/output and calculations.
- The RecipeBoxUI class, which consists of the graphical user interface.
- The Ingredient class, which contains a data structure for a single ingredient in a recipe.

```
/*
 * The Ingredient class is a data structure that holds all the
 * required data for a single ingredient in a recipe.
 */
package my.recipebox;


public class Ingredient extends RecipeBoxUI{
    private double quantity;
    private String unit;
    private String item;

    public Ingredient()
    {
        // default ctor
        this.quantity = 0.0;
        this.unit = "";
        this.item = "";
    }

    public Ingredient(double q, String u, String i)
    {
        this.quantity = q;
        this.unit = u;
        this.item = i;
    }

    public double getQuantity()
    {
        return quantity;
    }

    public String getUnit()
    {
        return unit;
    }

    public String getItem()
    {
        return item;
    }

    public void setQuantity(double q)
    {
        this.quantity = q;
    }

    public void setUnit(String u)
    {
        this.unit = u;
    }
    public void setItem(String i)
    {
        this.item = i;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--This is a form that describes the GUI layout.  It is generated by
NetBeans for use by the Design View part of the IDE -->
<Form version="1.3" maxVersion="1.8"
type="org.netbeans.modules.form.forminfo.JFrameFormInfo">
  <Properties>
    <Property name="defaultCloseOperation" type="int" value="3"/>
    <Property name="title" type="java.lang.String" value="Dave&apos;s
Recipe Box"/>
  </Properties>
  <SyntheticProperties>
    <SyntheticProperty name="formSizePolicy" type="int" value="1"/>
    <SyntheticProperty name="generateCenter" type="boolean"
value="false"/>
  </SyntheticProperties>
  <AuxValues>
    <AuxValue name="FormSettings_autoResourcing"
type="java.lang.Integer" value="0"/>
    <AuxValue name="FormSettings_autoSetComponentName"
type="java.lang.Boolean" value="false"/>
    <AuxValue name="FormSettings_generateFQN" type="java.lang.Boolean"
value="true"/>
    <AuxValue name="FormSettings_generateMnemonicsCode"
type="java.lang.Boolean" value="false"/>
    <AuxValue name="FormSettings_i18nAutoMode" type="java.lang.Boolean"
value="false"/>
    <AuxValue name="FormSettings_layoutCodeTarget"
type="java.lang.Integer" value="2"/>
    <AuxValue name="FormSettings_listenerGenerationStyle"
type="java.lang.Integer" value="0"/>
    <AuxValue name="FormSettings_variablesLocal"
type="java.lang.Boolean" value="false"/>
    <AuxValue name="FormSettings_variablesModifier"
type="java.lang.Integer" value="2"/>
  </AuxValues>

  <Layout>
    <DimensionLayout dim="0">
      <Group type="103" groupAlignment="0" attributes="0">
          <Group type="102" alignment="0" attributes="0">
              <EmptySpace max="-2" attributes="0"/>
              <Component id="mainPanel" min="-2" max="-2"
attributes="0"/>
              <EmptySpace type="separate" max="-2" attributes="0"/>
              <Component id="imagePanel" max="32767" attributes="0"/>
              <EmptySpace min="-2" pref="18" max="-2" attributes="0"/>
          </Group>
      </Group>
    </DimensionLayout>
    <DimensionLayout dim="1">
      <Group type="103" groupAlignment="0" attributes="0">
          <Group type="102" alignment="0" attributes="0">
              <EmptySpace max="-2" attributes="0"/>
              <Component id="imagePanel" max="-2" attributes="0"/>
              <EmptySpace max="32767" attributes="0"/>
          </Group>
          <Component id="mainPanel" alignment="1" max="32767"
attributes="0"/>
      </Group>
    </DimensionLayout>
  </Layout>
  <SubComponents>
    <Container class="javax.swing.JPanel" name="imagePanel">
```

```
      <Layout>
        <DimensionLayout dim="0">
          <Group type="103" groupAlignment="0" attributes="0">
              <Group type="102" alignment="0" attributes="0">
                  <EmptySpace min="-2" pref="45" max="-2"
attributes="0"/>
                  <Component id="imageLabel" min="-2" max="-2"
attributes="0"/>
                  <EmptySpace pref="351" max="32767" attributes="0"/>
              </Group>
          </Group>
        </DimensionLayout>
        <DimensionLayout dim="1">
          <Group type="103" groupAlignment="0" attributes="0">
              <Group type="102" alignment="0" attributes="0">
                  <EmptySpace min="-2" pref="35" max="-2"
attributes="0"/>
                  <Component id="imageLabel" min="-2" max="-2"
attributes="0"/>
                  <EmptySpace pref="503" max="32767" attributes="0"/>
              </Group>
          </Group>
        </DimensionLayout>
      </Layout>
      <SubComponents>
        <Component class="javax.swing.JLabel" name="imageLabel">
        </Component>
      </SubComponents>
    </Container>
    <Container class="javax.swing.JPanel" name="mainPanel">

      <Layout>
        <DimensionLayout dim="0">
          <Group type="103" groupAlignment="0" attributes="0">
              <Group type="102" attributes="0">
                  <EmptySpace max="-2" attributes="0"/>
                  <Group type="103" groupAlignment="0" attributes="0">
                      <Component id="recipeList" alignment="1"
max="32767" attributes="0"/>
                      <Component id="ingredientListPane" alignment="1"
pref="393" max="32767" attributes="0"/>
                      <Component id="directionsPane" alignment="0"
max="32767" attributes="0"/>
                      <Group type="102" attributes="0">
                          <Group type="103" groupAlignment="0"
attributes="0">
                              <Group type="102" attributes="0">
                                  <EmptySpace min="6" pref="6" max="-2"
attributes="0"/>
                                  <Component id="directionsLabel" min="-
2" max="-2" attributes="0"/>
                              </Group>
                              <Group type="102" alignment="0"
attributes="0">
                                  <Component id="getRecipeButton" min="-
2" max="-2" attributes="0"/>
                                  <EmptySpace max="-2" attributes="0"/>
                                  <Component id="editRecipeButton"
min="-2" max="-2" attributes="0"/>
                                  <EmptySpace max="-2" attributes="0"/>
                                  <Component id="deleteRecipeButton"
min="-2" max="-2" attributes="0"/>
                              </Group>
                              <Component id="ingredientListLabel"
```

```
alignment="0" min="-2" max="-2" attributes="0"/>
                              <Group type="102" alignment="0"
attributes="0">
                                  <Component id="imageListLabel" min="-
2" max="-2" attributes="0"/>
                                  <EmptySpace max="-2" attributes="0"/>
                                  <Component id="imageList" min="-2"
max="-2" attributes="0"/>
                              </Group>
                              <Group type="102" alignment="0"
attributes="0">
                                  <Component id="servingsBox" min="-2"
pref="59" max="-2" attributes="0"/>
                                  <EmptySpace max="-2" attributes="0"/>
                                  <Component id="servingsButton" min="-
2" max="-2" attributes="0"/>
                                  <EmptySpace max="-2" attributes="0"/>
                                  <Component id="jButton1" min="-2"
max="-2" attributes="0"/>
                              </Group>
                          </Group>
                          <EmptySpace min="0" pref="0" max="32767"
attributes="0"/>
                      </Group>
                  </Group>
                  <EmptySpace max="-2" attributes="0"/>
                  <Component id="jSeparator1" min="-2" pref="21" max="-
2" attributes="0"/>
                  <EmptySpace max="-2" attributes="0"/>
              </Group>
          </Group>
        </DimensionLayout>
        <DimensionLayout dim="1">
          <Group type="103" groupAlignment="0" attributes="0">
              <Group type="102" alignment="0" attributes="0">
                  <EmptySpace min="-2" pref="15" max="-2"
attributes="0"/>
                  <Component id="recipeList" min="-2" max="-2"
attributes="0"/>
                  <EmptySpace min="-2" pref="5" max="-2"
attributes="0"/>
                  <Group type="103" groupAlignment="3" attributes="0">
                      <Component id="getRecipeButton" alignment="3"
min="-2" max="-2" attributes="0"/>
                      <Component id="editRecipeButton" alignment="3"
min="-2" max="-2" attributes="0"/>
                      <Component id="deleteRecipeButton" alignment="3"
min="-2" max="-2" attributes="0"/>
                  </Group>
                  <EmptySpace max="-2" attributes="0"/>
                  <Component id="ingredientListLabel" min="-2" max="-2"
attributes="0"/>
                  <EmptySpace max="-2" attributes="0"/>
                  <Component id="ingredientListPane" min="-2" pref="138"
max="-2" attributes="0"/>
                  <EmptySpace max="-2" attributes="0"/>
                  <Group type="103" groupAlignment="3" attributes="0">
                      <Component id="servingsBox" alignment="3" min="-2"
max="-2" attributes="0"/>
                      <Component id="servingsButton" alignment="3"
min="-2" max="-2" attributes="0"/>
                      <Component id="jButton1" alignment="3" min="-2"
max="-2" attributes="0"/>
                  </Group>
                  <EmptySpace max="-2" attributes="0"/>
                  <Group type="103" groupAlignment="3" attributes="0">
                      <Component id="imageListLabel" alignment="3"
min="-2" max="-2" attributes="0"/>
                      <Component id="imageList" alignment="3" min="-2"
max="-2" attributes="0"/>
                  </Group>
                  <EmptySpace max="-2" attributes="0"/>
                  <Component id="directionsLabel" min="-2" max="-2"
attributes="0"/>
                  <EmptySpace min="-2" pref="3" max="-2"
attributes="0"/>
                  <Component id="directionsPane" max="32767"
attributes="0"/>
                  <EmptySpace max="-2" attributes="0"/>
              </Group>
              <Component id="jSeparator1" alignment="1" max="32767"
attributes="0"/>
          </Group>
        </DimensionLayout>
      </Layout>
      <SubComponents>
        <Component class="javax.swing.JComboBox" name="recipeList">
          <Properties>
            <Property name="editable" type="boolean" value="true"/>
            <Property name="model" type="javax.swing.ComboBoxModel"
editor="org.netbeans.modules.form.editors2.ComboBoxModelEditor">
              <StringArray count="0"/>
            </Property>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="recipeListActionPerformed"/>
          </Events>
        </Component>
        <Container class="javax.swing.JScrollPane"
name="ingredientListPane">
          <AuxValues>
            <AuxValue name="autoScrollPane" type="java.lang.Boolean"
value="true"/>
          </AuxValues>

          <Layout
class="org.netbeans.modules.form.compat2.layouts.support.JScrollPaneSupp
ortLayout"/>
          <SubComponents>
            <Component class="javax.swing.JTextArea"
name="ingredientListBox">
              <Properties>
                <Property name="columns" type="int" value="20"/>
                <Property name="rows" type="int" value="5"/>
                <Property name="minimumSize" type="java.awt.Dimension"
editor="org.netbeans.beaninfo.editors.DimensionEditor">
                  <Dimension value="[240, 80]"/>
                </Property>
              </Properties>
            </Component>
          </SubComponents>
        </Container>
        <Component class="javax.swing.JButton" name="getRecipeButton">
          <Properties>
            <Property name="text" type="java.lang.String" value="get"/>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="getRecipeButtonActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JButton" name="editRecipeButton">
          <Properties>
            <Property name="text" type="java.lang.String" value="save"/>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="editRecipeButtonActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JButton"
name="deleteRecipeButton">
          <Properties>
            <Property name="text" type="java.lang.String"
value="delete"/>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="deleteRecipeButtonActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JLabel"
name="ingredientListLabel">
          <Properties>
            <Property name="text" type="java.lang.String"
value="Ingredient list"/>
          </Properties>
        </Component>
        <Component class="javax.swing.JTextField" name="servingsBox">
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="servingsBoxActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JButton" name="servingsButton">
          <Properties>
            <Property name="text" type="java.lang.String" value="change
# of servings"/>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="servingsButtonActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JLabel" name="imageListLabel">
          <Properties>
            <Property name="text" type="java.lang.String" value="Add or
change image"/>
          </Properties>
        </Component>
        <Component class="javax.swing.JComboBox" name="imageList">
          <Properties>
            <Property name="model" type="javax.swing.ComboBoxModel"
editor="org.netbeans.modules.form.editors2.ComboBoxModelEditor">
              <StringArray count="1">
                <StringItem index="0" value="none"/>
              </StringArray>
            </Property>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="imageListActionPerformed"/>
          </Events>
        </Component>
        <Component class="javax.swing.JLabel" name="directionsLabel">
          <Properties>
            <Property name="text" type="java.lang.String"
value="Directions"/>
          </Properties>
        </Component>
        <Container class="javax.swing.JScrollPane"
name="directionsPane">
          <AuxValues>
            <AuxValue name="autoScrollPane" type="java.lang.Boolean"
value="true"/>
          </AuxValues>

          <Layout
class="org.netbeans.modules.form.compat2.layouts.support.JScrollPaneSupp
ortLayout"/>
          <SubComponents>
            <Component class="javax.swing.JTextArea"
name="directionsBox">
              <Properties>
                <Property name="columns" type="int" value="20"/>
                <Property name="rows" type="int" value="5"/>
                <Property name="minimumSize" type="java.awt.Dimension"
editor="org.netbeans.beaninfo.editors.DimensionEditor">
                  <Dimension value="[240, 80]"/>
                </Property>
              </Properties>
            </Component>
          </SubComponents>
        </Container>
        <Component class="javax.swing.JSeparator" name="jSeparator1">
        </Component>
        <Component class="javax.swing.JButton" name="jButton1">
          <Properties>
            <Property name="text" type="java.lang.String"
value="US/Metric"/>
          </Properties>
          <Events>
            <EventHandler event="actionPerformed"
listener="java.awt.event.ActionListener"
parameters="java.awt.event.ActionEvent"
handler="jButton1ActionPerformed"/>
          </Events>
        </Component>
      </SubComponents>
    </Container>
  </SubComponents>
```

```
</Form>
```

```
package my.recipebox;


import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.DOMException;

// for xml stuff
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;
/**
 *
 * @author davidjansing
 */
public class RecipeBoxUI extends javax.swing.JFrame {
    protected ArrayList<Ingredient> ingredients;
    private String directions = "";
    private String substitutions;
    int servings;
    private final String dataPath;
    private final String imgPath;
    private final String[] units;
    private String image = null;
    /**
     * Creates new form recipeBoxUI
     */
    public RecipeBoxUI() {
        this.units = new String[]{"tsp", "teaspoon", "teaspoons",
                                  "tbsp", "tablespoon",
                                  "tablespoons", "cup", "cups", "pint",
                                  "pints", "pt", "quart", "quarts",
                                  "qt", "gallon", "gallons", "gal",
                                  "dl", "cl", "liter", "liters", "l",
                                  "ml", "oz", "ounce", "ounces", "lb",
                                  "pound", "pounds", "g", "gram",
                                  "grams", "kg", "kilo", "kilos",
                                  "kilogram", "kilograms"};
        initComponents();
        ingredients = new ArrayList();
        dataPath = "./xml";
        imgPath = "./photos";
        File recipeFiles = new File(dataPath);
        String[] filenames = recipeFiles.list();
        recipeList.setSize(270,recipeList.getHeight());
        for (String filename : filenames) {
            recipeList.addItem(filename);
        }
        recipeList.setSelectedItem("none");
        File imageFiles = new File(imgPath);
        String[] images = imageFiles.list();

        for (String img : images) {
            imageList.addItem(img);
        }
        imageList.setSelectedIndex(0);
    }

    public void readXmlFile(String fileLocation)
    {
        try
        {
            ingredients.clear();
            File xmlFile = new File(fileLocation);
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);

            // Get the ingredient list
            NodeList ingList = doc.getElementsByTagName("ingredient");
            for (int i=0; i<ingList.getLength(); i++)
            {
                Ingredient ing = new Ingredient();
                Node ingNode = ingList.item(i);
                Element ingElem = (Element) ingNode.getChildNodes();
                Node qNode =
                  ingElem.getElementsByTagName("quantity").item(0);
                ing.setQuantity
                  (Double.parseDouble(qNode.getTextContent()));
                Node uNode =
                  ingElem.getElementsByTagName("unit").item(0);
                ing.setUnit(uNode.getTextContent());
                Node iNode =
                  ingElem.getElementsByTagName("item").item(0);
                ing.setItem(iNode.getTextContent());
                ingredients.add(ing);
            }

            // Get the servings
            Node servNode =
                  doc.getElementsByTagName("servings").item(0);
            servings = Integer.parseInt(servNode.getTextContent());

            // Get the directions
            Node dirNode =
                  doc.getElementsByTagName("directions").item(0);
            directions = dirNode.getTextContent();


            // Get the photo, if any
            Node imgNode = doc.getElementsByTagName("image").item(0);
            image = imgNode.getTextContent();
        }
        catch (ParserConfigurationException | SAXException | IOException
ex)
        {
            Logger.getLogger
             (RecipeBoxUI.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
public boolean isUnit(String u)
{
    for(String unitString: units)
    {
        if (u.equals(unitString))
        {
        return true;
        }
    }
    return false;
}

private void updateIngredients(String ing)
{
 // Convert fractional quantities to decimal for storage
    String[] lines = ing.split("\n");
    ingredients.clear();
    for (String line: lines)
    {
        Ingredient in = new Ingredient();
        if (!line.equals("") && !line.equals("\n"))
        {
            String[] part = line.split(" ", 4);

            if (part[1].contains("/"))
            {
                String decimalPart = convertToDecimal(part[1]);
                String quant = "";

                if (decimalPart.equals("1"))
                {
                    int newVal = Integer.parseInt(part[0]) + 1;
                    quant += newVal;
                }
                else
                {
                    // combine parts 0 and 1 and convert
                    quant = part[0] + "." + decimalPart;
                }

                in.setQuantity(Double.parseDouble(quant));
                if (isUnit(part[2]))
                {
                    in.setUnit(part[2].trim());
                    if (part.length > 3)
                    {
                        in.setItem(part[3].trim());
                    }
                }
            }
            else if (part[0].contains("/"))
            {
                String decimalPart = convertToDecimal(part[0]);
                String quant = "";

                if (decimalPart.equals("1"))
                {
                    quant = "1";
                }
                else
                {
                    // combine parts 0 and 1 and convert
                    quant = "0." + decimalPart;
                }
```

```
                    // send the first part only to the converter
                    in.setQuantity(Double.parseDouble(quant));
                    if (isUnit(part[1]))
                    {
                        in.setUnit(part[1].trim());
                        String it = "";
                        for(int i = 2; i < part.length; i++)
                        {
                            it += " " + part[i];
                        }
                        in.setItem(it.trim());
                    }
                    else //there is no unit
                    {
                        String it = "";
                        for(int i = 1; i < part.length; i++)
                        {
                            it += " " + part[i];
                        }
                        in.setItem(it.trim());
                    }
                }
                else // quantity is a whole number
                {
                    in.setQuantity(Double.parseDouble(part[0]));
                    // Unit is of the form "2 ounces vodka"
                    if (isUnit(part[1]))
                    {
                        in.setUnit(part[1].trim());
                        String it = "";
                        for(int i = 2; i < part.length; i++)
                        {
                            it += " " + part[i];
                        }
                        in.setItem(it.trim());
                    }
                    else
                    {
                        String it = "";
                        for(int i = 1; i < part.length; i++)
                        {
                            it += " " + part[i];
                        }
                        in.setItem(it.trim());
                    }
                }

            ingredients.add(in);
            }
        }
    }

    private void createXml(String filename)
    {
        try
        {
            DocumentBuilderFactory docFactory =
                  DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder =
                  docFactory.newDocumentBuilder();

            // root elements
            Document doc = docBuilder.newDocument();
```

```
Element recipe = doc.createElement("recipe");
recipe.setAttribute("title",
      recipeList.getSelectedItem().toString());
doc.appendChild(recipe);
Element ingredientList =
      doc.createElement("ingredientList");
recipe.appendChild(ingredientList);

int i=0;

for(Ingredient ingred : ingredients)
{
    // ingredient elements
    Element ingredientElem =
      doc.createElement("ingredient");
    ingredientList.appendChild(ingredientElem);

    // quantity elements
    Element quantity = doc.createElement("quantity");
    String quant = "";
    quant += ingred.getQuantity();
    quantity.appendChild(doc.createTextNode(quant));
    ingredientElem.appendChild(quantity);

    // unit elements
    Element unitElem = doc.createElement("unit");
    unitElem.appendChild
      (doc.createTextNode(ingred.getUnit()));
    ingredientElem.appendChild(unitElem);

    // item elements
    Element itemElem = doc.createElement("item");
    itemElem.appendChild
      (doc.createTextNode(ingred.getItem()));
    ingredientElem.appendChild(itemElem);

    i++;
}

Element servingsElem = doc.createElement("servings");
String svg = "";
svg += servings;
servingsElem.appendChild(doc.createTextNode(svg));
recipe.appendChild(servingsElem);

Element directionsElem = doc.createElement("directions");
directionsElem.appendChild(doc.createTextNode(directions));
recipe.appendChild(directionsElem);

Element imgElem = doc.createElement("image");
if (!image.equals(""))
    imgElem.appendChild(doc.createTextNode(image));
recipe.appendChild(imgElem);

// write the content into xml file
TransformerFactory transformerFactory =
      TransformerFactory.newInstance();
Transformer transformer =
      transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new File(filename));

transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty
```

```
                ("{http://xml.apache.org/xslt}indent-amount","3");
        transformer.transform(source, result);

        System.out.println("File saved to " + filename);

 } catch (ParserConfigurationException | DOMException |
         IllegalArgumentException | TransformerException e) {
    }
}


// Setters and getters
public void setServings(int s)
{
    this.servings = s;
    servingsBox.setText(Integer.toString(servings));
}

public int getServings()
{
    return this.servings;
}

    // Setters and getters
public void setImage(String img)
{
    if (!img.equals("none"))
        this.image = img;
    else
        this.image = "";
}

public String getImage()
{
    return this.image;
}

public void setDirections(String s)
{
    this.directions = s;
    directionsBox.setText(directions);
}


public String getDirections()
{
    return this.directions;
}

public void setIngredients()
{
    setIngredients(ingredients);
}

public void setIngredients(ArrayList<Ingredient> ing)
{
    String ingredientList = "";
    this.ingredients = ing;
    for(Ingredient i : ing)
    {
        if ((i.getQuantity() <= 0.125  &&
              (i.getUnit().equals("cup") ||
               i.getUnit().equals("cups") ||
               i.getUnit().equals("pounds") ||
               i.getUnit().equals("pound") ||
               i.getUnit().equals("lb")))

         || (i.getQuantity() <= 0.25 &&
              (i.getUnit().equals("quart") ||
               i.getUnit().equals("quarts") ||
               i.getUnit().equals("qt") ||
               i.getUnit().equals("gallon") ||
               i.getUnit().equals("gallons") ||
               i.getUnit().equals("gal")))

         || (i.getQuantity() <= 0.34 &&
              (i.getUnit().equals("tablespoon") ||
               i.getUnit().equals("tablespoons")||
               i.getUnit().equals("tbsp"))) )
        {
            convertSmallerUnit(i);
        }

        else if ( (i.getQuantity() >= 3 &&
                  (i.getUnit().equals("teaspoon") ||

              i.getUnit().equals("teaspoons") ||
              i.getUnit().equals("tsp")))

         || (i.getQuantity() >= 4 &&
              (i.getUnit().equals("quart") ||
               i.getUnit().equals("quarts") ||
               i.getUnit().equals("qt") ||
               i.getUnit().equals("cups") ||
               i.getUnit().equals("cup")))

         || (i.getQuantity() >= 8  &&
              (i.getUnit().equals("tablespoons") ||
               i.getUnit().equals("tablespoon") ||
               i.getUnit().equals("tbsp") ||
               i.getUnit().equals("ounces") ||
               i.getUnit().equals("ounce") ||
               i.getUnit().equals("oz"))) )

        {
            convertLargerUnit(i);
        }

        ingredientList += convertToFractional(i.getQuantity());

        if (i.getUnit().length() > 0)
        {
            ingredientList += " " + i.getUnit();
        }
        ingredientList += " " + i.getItem() + "\n";
    }
    ingredientListBox.setText(ingredientList);
}

    public void addItemSafely(String item)
{
    boolean itemExists = false;
    for (int i=0; i < recipeList.getItemCount(); i++)
    {
        if (recipeList.getItemAt(i).equals(item))
        {
            itemExists = true;
            break;
        }
    }
    if (!itemExists)
    {
        recipeList.addItem(item);
    }
}

public ArrayList<Ingredient> getIngredients()
{
    return this.ingredients;
}

public void convertLargerUnit(Ingredient i)
{
    String unit = i.getUnit();
    switch(unit)
    {
        case("teaspoons"):
        case("teaspoon"):
        case("tsp"):
        {
        i.setQuantity(i.getQuantity()/3);
        i.setUnit("tablespoons");
        break;
        }
        case ("tablespoons"):
        case ("tablespoon"):
        case ("tbsp"):
        {
        i.setQuantity(i.getQuantity()/16);
        i.setUnit("cups");
        break;
        }
        case("cups"):
        case("cup"):
        {
        i.setQuantity(i.getQuantity()/4);
        i.setUnit("quarts");
        break;
        }
        case("quarts"):
        case("quart"):
        case("qt"):
        {
        i.setQuantity(i.getQuantity()/4);
        i.setUnit("gallons");
        break;
        }
        case("ounces"):
        case("ounce"):
        case("oz"):
        {
        i.setQuantity(i.getQuantity()/16);
        i.setUnit("pounds");
        break;
        }
        default:
        {
        //do nothing
        }
    }
}
```

```
public void convertSmallerUnit(Ingredient i)
{
    String unit = i.getUnit();
    switch(unit)
    {
        case ("tablespoons"):
        case ("tablespoon"):
        case ("tbsp"):
        {
        // There are 3 teaspoons in one tablespoon
        i.setQuantity(i.getQuantity()*3);
        i.setUnit("teaspoons");
        break;
        }
        case("cups"):
        case("cup"):
        {
        // There are 16 tablespoons in one cup
        i.setQuantity(i.getQuantity()*16);
        i.setUnit("tablespoons");
        break;
        }
        case("quarts"):
        case("quart"):
        case("qt"):
        {
        // There are 4 cups in one quart
        i.setQuantity(i.getQuantity()*4);
        i.setUnit("cups");
        break;
        }
        case("gallons"):
        case("gallon"):
        case("gal"):
        {
        // There are 4 quarts in one gallon
        i.setQuantity(i.getQuantity()*4);
        i.setUnit("quarts");
        break;
        }
        case("pounds"):
        case("pound"):
        case("lb"):
        {
        // There are 16 ounces in one pound
        i.setQuantity(i.getQuantity()*16);
        i.setUnit("ounces");
        break;
        }
        default:
        {
        //do nothing
        }
    }
}

public String convertToFractional(double dec)
{
    double d = dec - (int)dec;
    String retval = "";
    if ((int)dec > 0)
    {
        if (d > 0.0)
        {
                // if there's a fraction, allow a space for it
                retval += (int)dec + " ";
            }
            else
            {
                // otherwise, leave it out
                retval += (int)dec;
            }
        }

        // The boundaries are kind of arbitrary, I admit,
        // but I did eyeball them on the number line and
        // they're close enough for the customer (me)
        if (d >= 0.0 && d < 0.15 && (int)dec == 0)
        {
            retval += "< 1/4";
        }
        else if (d >= 0.15 && d < 0.3)
        {
             retval += "1/4";
        }
        else if (d >= 0.3 && d < 0.4)
        {
             retval += "1/3";
        }
        else if (d >= 0.4 && d < 0.6)
        {
             retval += "1/2";
        }
        else if (d >= 0.6 && d < 0.7)
        {
             retval += "2/3";
        }
        else if (d >= 0.7 && d < 0.85)
        {
             retval += "3/4";
        }
        else if (d >= 0.85) //close enough to 1 to just increment the
value
        {
            int nextVal = (int)dec + 1;
            retval = "";
            retval += nextVal;
        }
        return retval;
    }

    public String convertToDecimal(String fraction)
    {
        String retval = "";
        String[] numbers = fraction.split("/");
        if (numbers[0].equals(numbers[1]))
        {
            return "1";
        }
        double numerator= Double.parseDouble(numbers[0]);
        double denominator = Double.parseDouble(numbers[1]);
        double decimalFraction = numerator/denominator;
        retval += decimalFraction;
        return retval.substring(retval.indexOf(".")+1);
    }

    private Object makeObj(final String str) {
        return new Object() {
```

```
        @Override
        public String toString() {
            return str;}};
        }

    /**
     * This method is called from within the constructor to initialize
the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        imagePanel = new javax.swing.JPanel();
        imageLabel = new javax.swing.JLabel();
        mainPanel = new javax.swing.JPanel();
        recipeList = new javax.swing.JComboBox();
        ingredientListPane = new javax.swing.JScrollPane();
        ingredientListBox = new javax.swing.JTextArea();
        getRecipeButton = new javax.swing.JButton();
        editRecipeButton = new javax.swing.JButton();
        deleteRecipeButton = new javax.swing.JButton();
        ingredientListLabel = new javax.swing.JLabel();
        servingsBox = new javax.swing.JTextField();
        servingsButton = new javax.swing.JButton();
        imageListLabel = new javax.swing.JLabel();
        imageList = new javax.swing.JComboBox();
        directionsLabel = new javax.swing.JLabel();
        directionsPane = new javax.swing.JScrollPane();
        directionsBox = new javax.swing.JTextArea();
        jSeparator1 = new javax.swing.JSeparator();
        jButton1 = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Dave's Recipe Box");

        org.jdesktop.layout.GroupLayout imagePanelLayout =
            new org.jdesktop.layout.GroupLayout(imagePanel);
        imagePanel.setLayout(imagePanelLayout);
        imagePanelLayout.setHorizontalGroup
            (imagePanelLayout.createParallelGroup
                  (org.jdesktop.layout.GroupLayout.LEADING)
            .add(imagePanelLayout.createSequentialGroup()
                .add(45, 45, 45)
                .add(imageLabel)
                .addContainerGap(351, Short.MAX_VALUE))
        );
        imagePanelLayout.setVerticalGroup
            (imagePanelLayout.createParallelGroup
                  (org.jdesktop.layout.GroupLayout.LEADING)
                  .add(imagePanelLayout.createSequentialGroup()
                .add(35, 35, 35)
                .add(imageLabel)
                .addContainerGap(503, Short.MAX_VALUE))
        );

        recipeList.setEditable(true);
        recipeList.addActionListener
            (new java.awt.event.ActionListener() {
```

```
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        recipeListActionPerformed(evt);
    }
});

ingredientListBox.setColumns(20);
ingredientListBox.setRows(5);
ingredientListBox.setMinimumSize
    (new java.awt.Dimension(240, 80));
ingredientListPane.setViewportView(ingredientListBox);

getRecipeButton.setText("get");
getRecipeButton.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        getRecipeButtonActionPerformed(evt);
    }
});

editRecipeButton.setText("save");
editRecipeButton.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        editRecipeButtonActionPerformed(evt);
    }
});

deleteRecipeButton.setText("delete");
deleteRecipeButton.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        deleteRecipeButtonActionPerformed(evt);
    }
});

ingredientListLabel.setText("Ingredient list");




servingsBox.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        servingsBoxActionPerformed(evt);
    }
});

servingsButton.setText("change # of servings");
servingsButton.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        servingsButtonActionPerformed(evt);
    }
});

imageListLabel.setText("Add or change image");
```

```
imageList.setModel(new javax.swing.DefaultComboBoxModel
    (new String[] { "none" }));
imageList.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        imageListActionPerformed(evt);
    }
});

directionsLabel.setText("Directions");

directionsBox.setColumns(20);
directionsBox.setRows(5);
directionsBox.setMinimumSize(new java.awt.Dimension(240, 80));
directionsPane.setViewportView(directionsBox);

jButton1.setText("US/Metric");
jButton1.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        jButton1ActionPerformed(evt);
    }
});

org.jdesktop.layout.GroupLayout mainPanelLayout =
    new org.jdesktop.layout.GroupLayout(mainPanel);
mainPanel.setLayout(mainPanelLayout);
mainPanelLayout.setHorizontalGroup
    (mainPanelLayout.createParallelGroup
          (org.jdesktop.layout.GroupLayout.LEADING)
          .add(mainPanelLayout.createSequentialGroup()
          .addContainerGap()
            .add
                (mainPanelLayout.createParallelGroup
                      (org.jdesktop.layout.GroupLayout.LEADING)
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
                 recipeList,
                 0,
                 org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                 Short.MAX_VALUE)
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
                 ingredientListPane,
                 org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                 393,
                 Short.MAX_VALUE)
            .add(directionsPane)
            .add(mainPanelLayout.createSequentialGroup()
            .add
                (mainPanelLayout.createParallelGroup
                      (org.jdesktop.layout.GroupLayout.LEADING)
            .add(mainPanelLayout.createSequentialGroup()
            .add(6, 6, 6)
            .add(directionsLabel))
            .add(mainPanelLayout.createSequentialGroup()
            .add(getRecipeButton)
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(editRecipeButton)
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(deleteRecipeButton))
            .add(ingredientListLabel)
            .add(mainPanelLayout.createSequentialGroup()
            .add(imageListLabel)
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(imageList,
               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
               org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
            .add(mainPanelLayout.createSequentialGroup()
            .add(servingsBox,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                59,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(servingsButton)
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(jButton1)))
            .add(0, 0, Short.MAX_VALUE)))
            .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
            .add(jSeparator1,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                21,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
          );
mainPanelLayout.setVerticalGroup
    (mainPanelLayout.createParallelGroup
          (org.jdesktop.layout.GroupLayout.LEADING)
          .add(mainPanelLayout.createSequentialGroup()
          .add(15, 15, 15)
          .add(recipeList,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
          .add(5, 5, 5)

          .add(mainPanelLayout.createParallelGroup
                (org.jdesktop.layout.GroupLayout.BASELINE)
                .add(getRecipeButton)
                .add(editRecipeButton)
                .add(deleteRecipeButton))
          .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
          .add(ingredientListLabel)
          .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
          .add(ingredientListPane,
               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
               138,
               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap
                (org.jdesktop.layout.LayoutStyle.RELATED)
          .add
                (mainPanelLayout.createParallelGroup
                    (org.jdesktop.layout.GroupLayout.BASELINE)
                .add(servingsBox,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .add(servingsButton)
```

```
                        .add(jButton1))
                  .addPreferredGap
                        (org.jdesktop.layout.LayoutStyle.RELATED)
                  .add
                        (mainPanelLayout.createParallelGroup
                             (org.jdesktop.layout.GroupLayout.BASELINE)
                        .add(imageListLabel)
                        .add(imageList,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                        org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                       org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                 .addPreferredGap
                        (org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(directionsLabel)
                        .add(3, 3, 3)
                        .add(directionsPane)
                        .addContainerGap())
                 .add(org.jdesktop.layout.GroupLayout.TRAILING,
                      jSeparator1)
        );

        org.jdesktop.layout.GroupLayout layout =
            new org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup
            (layout.createParallelGroup
                  (org.jdesktop.layout.GroupLayout.LEADING)
                  .add(layout.createSequentialGroup()
                  .addContainerGap()
                  .add(mainPanel,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                  org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                      .add(18, 18, 18)
                      .add(imagePanel,
                         org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                         org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                         Short.MAX_VALUE)
                     .add(18, 18, 18))
        );
        layout.setVerticalGroup
            (layout.createParallelGroup
                  (org.jdesktop.layout.GroupLayout.LEADING)
                      .add(layout.createSequentialGroup()
                      .addContainerGap()
                      .add(imagePanel,
                        org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                        org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                     .addContainerGap
                        (org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                         Short.MAX_VALUE))
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
                         mainPanel,
                         org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                         org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                         Short.MAX_VALUE)
        );
        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void servingsBoxActionPerformed
                  (java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_servingsBoxActionPerformed
```

```
    }//GEN-LAST:event_servingsBoxActionPerformed

    private void recipeListActionPerformed
                  (java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_recipeListActionPerformed

    }//GEN-LAST:event_recipeListActionPerformed

    private void
servingsButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_servingsButtonActionPerformed
        String servingsStr = (String) servingsBox.getText();
        // Shouldn't have zero servings of anything
        int newServings = servings;

        // Nor should we have half-servings of anything
        try
        {
            if (servingsStr.contains("."))
            {
                // drop the fractional part
                newServings = Integer.parseInt
                                 (servingsStr.substring
                                    (0, servingsStr.indexOf(".")));
            }
            // I said no zeroes, and I meant it.
            else if (Integer.parseInt(servingsStr) == 0)
            {
                // do nothing
            }
            else
            {
                newServings = Integer.parseInt(servingsStr);
            }
            if (newServings != servings)
            {
                for (Ingredient i : ingredients)
                {
                    double newQuant = i.getQuantity();
                    newQuant*=newServings;
                    newQuant/=servings;
                    i.setQuantity(newQuant);
                }
            setIngredients();
            }
        }
        catch (NumberFormatException nfe)
        {
            // Basically don't change anything
        }
        setServings(newServings);
    }//GEN-LAST:event_servingsButtonActionPerformed

    private void
getRecipeButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_getRecipeButtonActionPerformed
        // Get the selected recipe from the database
        String filename = (String) recipeList.getSelectedItem();
        readXmlFile(dataPath + "/" + filename);
        setServings(servings);
        setIngredients(ingredients);
        setDirections(directions);
        setImage(image);
        imagePanel.setVisible(true);
        if (!image.equals(""))
        {
            ImageIcon icon = new ImageIcon("./photos/" + image);
            imageLabel.setIcon(icon);
            imageLabel.setText("");
            imageLabel.setVisible(true);
            imageList.setSelectedItem(image);
        }
        else
        {
            imageLabel.setIcon(null);
            imageLabel.setText("");
            imageLabel.setVisible(false);
        }
    }//GEN-LAST:event_getRecipeButtonActionPerformed

    private void
editRecipeButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_editRecipeButtonActionPerformed
        String title = (String) recipeList.getSelectedItem();
        directions = directionsBox.getText();
        servings = Integer.parseInt(servingsBox.getText());
        String ing = ingredientListBox.getText();
        updateIngredients(ing);
        createXml(dataPath + "/" + title);
        addItemSafely(title);
    }//GEN-LAST:event_editRecipeButtonActionPerformed

    private void
deleteRecipeButtonActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_deleteRecipeButtonActionPerformed

        String title = (String) recipeList.getSelectedItem();
        File file = new File(dataPath + "/" + title);

        if(file.exists())
        {
            file.delete();
            recipeList.removeItem(title);
            recipeList.setSelectedIndex(0);
            ingredientListBox.setText("");
            servingsBox.setText("");
            directionsBox.setText("");
        }
    }//GEN-LAST:event_deleteRecipeButtonActionPerformed

    private void imageListActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_imageListActionPerformed
        setImage((String) imageList.getSelectedItem());
        imagePanel.setVisible(true);
        ImageIcon icon = new ImageIcon("./photos/" + image);
        imageLabel.setIcon(icon);
        imageLabel.setText("");
        imageLabel.setVisible(true);
    }//GEN-LAST:event_imageListActionPerformed

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_jButton1ActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton deleteRecipeButton;
    private javax.swing.JTextArea directionsBox;
    private javax.swing.JLabel directionsLabel;
    private javax.swing.JScrollPane directionsPane;
    private javax.swing.JButton editRecipeButton;
    private javax.swing.JButton getRecipeButton;
    private javax.swing.JLabel imageLabel;
    private javax.swing.JComboBox imageList;
    private javax.swing.JLabel imageListLabel;
    private javax.swing.JPanel imagePanel;
    private javax.swing.JTextArea ingredientListBox;
    private javax.swing.JLabel ingredientListLabel;
    private javax.swing.JScrollPane ingredientListPane;
    private javax.swing.JButton jButton1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JComboBox recipeList;
    private javax.swing.JTextField servingsBox;
    private javax.swing.JButton servingsButton;
    // End of variables declaration//GEN-END:variables

    // Accessor methods for the various GUI components
    // We need these so that the regression tests can use them.
    // Otherwise, you couldn’t keep them private.
    public javax.swing.JButton getDeleteRecipeButton()
    {
        return deleteRecipeButton;
    }

    public javax.swing.JTextArea getDirectionsBox()
    {
        return directionsBox;
    }

    public javax.swing.JButton getEditRecipeButton()
    {
        return editRecipeButton;
    }

    public javax.swing.JButton getGetRecipeButton()
    {
        return getRecipeButton;
    }

    public javax.swing.JComboBox getImageList()
    {
        return imageList;
    }

    public javax.swing.JTextArea getIngredientListBox()
    {
        return ingredientListBox;
    }

    public javax.swing.JComboBox getRecipeList()
    {
        return recipeList;
    }

    public javax.swing.JTextField getServingsBox()
    {
        return servingsBox;
    }

    public javax.swing.JButton getServingsButton()
```

```
    {
        return servingsButton;
    }
}
```

```
/*
 * This is the main RecipeBox class that does the execution
 */
package my.recipebox;

/**
 * @author davidjansing
 */
public class RecipeBox {

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {

        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
                 javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel
                        (info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(RecipeBoxUI.class.getName()).log
     (java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(RecipeBoxUI.class.getName()).log
     (java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(RecipeBoxUI.class.getName()).log
     (java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(RecipeBoxUI.class.getName()).log
     (java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

/* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new RecipeBoxUI().setVisible(true);
            }
        });
    }
}
```

References:

1. Elfriede Dustin, Jeff Rashka, John Paul, *Automated Software Testing, Introduction, Management, and Performance*, pp. 9, 120-133, Addison Wesley, 1999

2. David McClure, *Why Do We Trust Automated Tests?* – http://dclure.org/essays/why-do-we-trust-automated-tests (2013)

3. Cem Kaner, *Avoiding Shelfware: A Managers' View of Automated GUI Testing*, http://www.kaner.com/pdfs/shelfwar.pdf, 2002

4. Laurie Williams, Gunnar Kudrhavets, Nachiappan Nagappan, *On the Effectiveness of Init Test Automation at Microsoft*, Department of Computer Science, North Carolina State University (2009)

5. Exforsis, *Automated Testing Advantages, Disadvantages and Guidelines*, http://www.exforsys.com/tutorials/testing/automated-testing-advantages-disadvantages-and-guidelines.html, 2005

6. Quality First Software (homepage) http://www.qfs.de/

7. T-Plan (homepage), http://www.t-plan.com/

8. Robert C. Martin, *Clean Code*, Prentice Hall, 2009

9. Overview of QF-Test, http://www.qfs.de/en/qftest/index.html

10. Gassy Assassa, Hassan Mathkour, Bander Al-Ghafees, *Automated Software Testing in Educational Environment: A Design of Testing Framework for Extreme Programming*, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia (2006)

11. Rudolf Ramler and Klaus Wolfmaier, *Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost*, Software Competence Center Hagenberg GmbH, Hagenberg, Austria (2006)

12. Dudekula Mohammed Rafi, Katam Reddy Kiran Moses, Kai Petersen, *Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey*, Blekinge Institute of Technology/Erikson AB School of Computing, Karlskrona, Sweden, 2012

13. Rachel S. Smith, *Writing a Requirements Document: Workshop Materials*, http://www.cdl.edu/cdl_resources/writing-requirements, Center for Distributed Learning, California State University

14. Software Testing Help (Commercial Website), *Writing Test Cases From SRS Document*, http://www.softwaretestinghelp.com/writing-test-cases-from-srs-software-testing-qa-training-day-4

15. Shuang Wang, Jeff Offutt, *Comparison of Unit-Level Automated Test Generation Tools*, George Mason University, Fairfax, VA (2009)

16. Mark Fewster, Dorothy Graham, *Software Test Automation*, Addison-Wesley, pp 234-3, 267-8, 326-341, 519-20, 527-8 (1999)

17. Stefan Berner, Roland Weber, Rudolf Keller, *Observations and Lessons Learned from Automated Testing*, Zuelke Engineering, AG, Zurich Switzerland (2005)

18. Matthias Mueller, Frank Padberg, *About the Return on Investment of Test-Driven Development*, University of Karlsruhe, Germany.

19. *Test Automation Tool comparison – HP UFT/QTP vs. Selenium*, Aspire Systems, Inc., San Jose, California

20. *Designing a Test Environment*, Microsoft Corporation, https://technet.microsoft.com/en-us/library/cc778654%28v=ws.10%29.aspx (2003)

21. Martin Fowler, *TestCoverage*, http://martinfowler.com/bliki/TestCoverage.html (2012)

22. Yvette Francino, *What is the difference between code coverage and test coverage?*, TechTarget, http://searchsoftwarequality.techtarget.com/answer/What-is-the-difference-between-code-coverage-and-test-coverage (2010)

23. Trevor Atkins, *COTS Tools vs. Scripting Languages for Test Automation, Thinking Through Testing*, http://thinktesting.com/articles/cots-tools-vs-scripting-languages-for-test-automation (2002)

24. George Candea, Stefan Bucur, Cristian Zamfir, *Automated Software Testing as a Service*, School of Computer and Communication Services, EPFL, Lausanne, Switzerland (2010)

25. Gustavo de Oliveira, Alexandre Duarte, *A framework for Automated Software Testing on the Cloud*, Federal University of Paraiba, Paraiba, Brazil (2010)

26. 3/8/15 – James Bach, *Agile Test Automation*, http://www.satisfice.com/articles/agileauto-paper.pdf

27. Voas, J. and Miller, K., "Software testability, the new verification" http://www.rstcorp.com/papers/chrono-1995.html , IEEE Software, p 3, (May 1995)

28. Burton, Ross, Subverting Java Access Protection for Unit Testing, http://www.onjava.com (2003)

29. *Oracle VM VirtualBox,* http://www.oracle.com/us/technologies/virtualization/virtualbox/overview/index.html, Oracle Corporation

30. Belorustes, Vladimir, *Efficient Configuration Test Automation using Virtual Machines,* Plaxo, Inc.

31. Yahong Sun, Edward Jones, *Specification-Driven Automated Testing of GUI-Based Java Programs*, Meditronic, Inc, and Florida A&M University (2004)

32. ConAgra Foods, *Pasta e Fagioli*, http://www.hunts.com/recipes-Pasta-e-Fagioli-4568

[33] *IEEE Standard Glossary of Software Engineering Technology ANSI/IEEE 610.12*, IEEE Press (1990).

[34] Beust, Cedric, TestNG documentation, http://testng.org/doc/index.html, (2004-2015)