MXC - 0250

Worcester Polytechnic Institute
Computer Science Department

Learning Skills Academy Automated Math Testing Project

A Major Qualifying Project

Submitted in partial fulfillment of the requirements for
Degree of Bachelor of Science in Computer Science

April 2010

____

Craig Laprade
Rachel McMannus

____

Professor Michael J. Ciaraldi, Advisor

Abstract

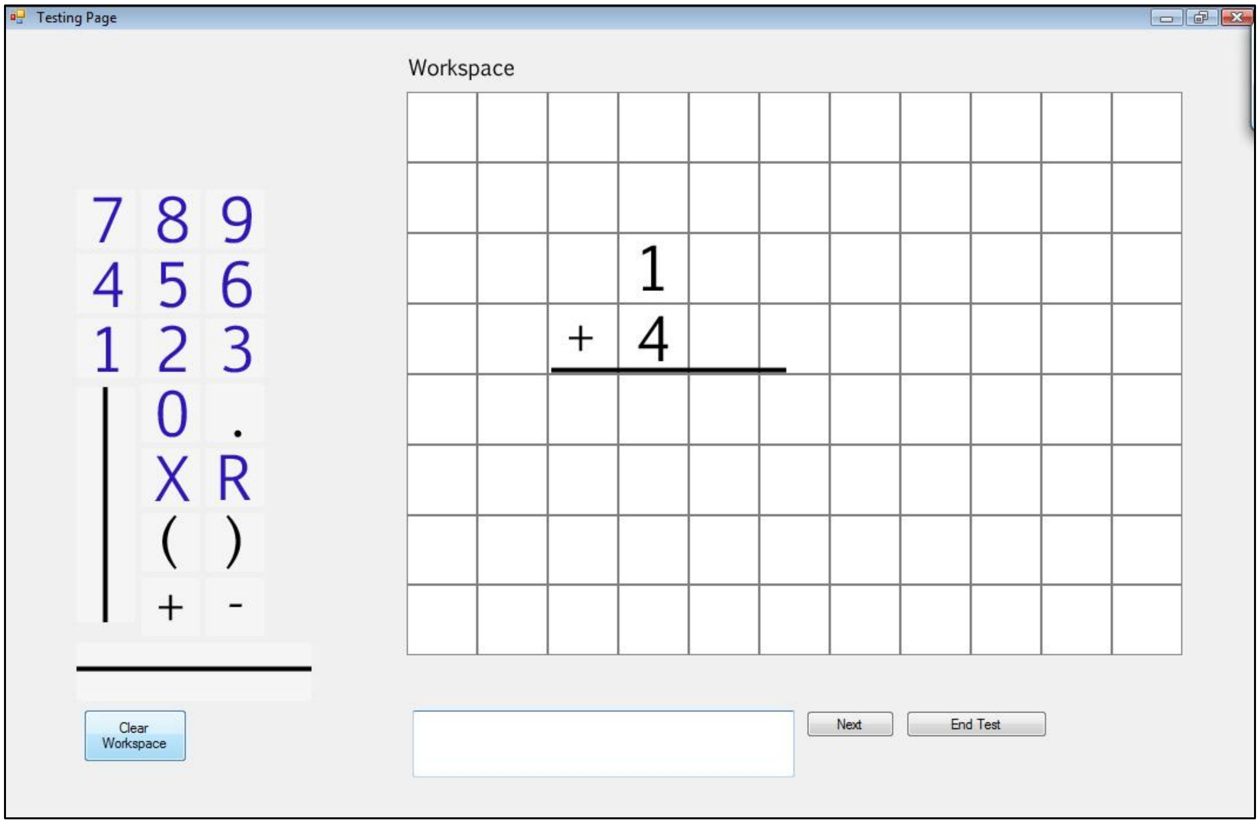
The Goal of this project was to create a program capable of effectively and efficiently measure a student's math and number skills at the Learning Skills Academy. The development of the software started by developing proofs of concepts in areas of perceived complexity, progressed to UI and background development, and concluded with UI testing and modification. The result was a program that met all client needs as well as established a readily accessible framework for extension.

Table of Contents

Introduction

The Learning Skills Academy Automated Math Testing Project was a successful attempt to generate a piece of software that would readily integrate into the LSA's existing computer system and replace an antiquated paper testing system that measures a student's mathematical Aptitude. The existing paper testing procedure involved a complicated set of levels in various categories to accurately test a student's ability to comprehend the various facets of mathematics. The LSA Automated Testing Program is able to increase the amount of testing that can be done at any time, the accuracy of record keeping, and decrease the amount to overhead an instructor at the LSA must do in order to test a student.

The program works by having the student, as identified by their Windows networked Username, determine which test they need to take through a simple drop-down menu. Once the student has selected their desired test, they are brought into the test where a series of 6 problems will be displayed to them as seen in the following diagram.



On the testing page, the student has the opportunity to work out the problem by dragging numbers and symbols into the workspace. This recreates the process of a pencil and paper, but has the advantage of number and symbol objects that can be moved and moved again, as well as deleted, all much faster than on a physical medium. Once the student discerns the answer, they enter it into the box at the bottom, a feature mandated by the LSA. When a student completes a test they are scored and the test data is stored on the network where the instructors can access and easily view their test results.

In this report, the circumstances that led to the need for the program will be explored, followed in detail by how the project team was able to meet and exceed the LSA's requirements. The report will finish with a conclusion about the final product and the possibilities for extension.

Background

The Learning Skills Academy

The Learning Skills Academy (LSA) is a private, non-profit school in Rye, New Hampshire for students in grades 4-12 that offers specialized education for students with average intelligence and significant learning disabilities or language impairments. Their mission statement is "… to enrich the lives of students by igniting and nurturing individuals' academic, social, and emotional potentials in a dynamic community which honors divergent learning styles" ("About Us").

LSA students come from New Hampshire, Massachusetts, and Maine, sometimes spending more than an hour in transport each way. Many of the students have been traumatized in their previous educational environments. Traditional schools that are unequipped to identify or overcome learning disabilities may unfairly label students as having low IQs, or behavior problems. Often their solutions either make the student a social outcast and/or hold them to increasingly high standards without providing the support they need to succeed.

Without exception, the students who enroll in LSA flourish –both academically and socially. Sometimes for the first time in their lives, the students look forward to coming to school (McManus). They no longer suffer from stomach pains and high anxiety.

LSA is unique because of its innovative teaching approach. The specific needs of every student are assessed in order to tailor a particular program. Language-based instruction is centric at the Learning Skills Academy. Students spend class time in small groups of 2-6 students and at least one instructor, if not receiving one-on-one instruction. Twice a month the students receive adventure-based learning to build confidence and teamwork. Their progress is continually assessed and regularly reported to the parents.

Traditional styles of instruction are not used at LSA. Concepts are emphasized, rather than blind memorization, in a step-by-step process.

Unfortunately, in exchange for adapting their teaching methods, LSA instructors have had to give up much of the commercially available teaching aids. Workbooks, instructional software, and test sheets don't match their needs.

Mathematics Instruction

A math disability, also known as dyscalculia, can be responsible for a person having trouble learning concepts like quantity, place value, and time. It can also make it difficult to memorize math facts, organize numbers, or understand how problems are organized on the page.

The Learning Skills Academy currently uses a system of instruction that divides mathematical skills into a series of very specific levels and categories. For example, level 1 of the addition category adds two single-digit numbers. Level 2 addition adds a single-digit number and a zero, because the zero concept can often confuse students with a math disability. Although they may study several categories simultaneously, students do not advance to the next level until they demonstrate a mastery of their current level.

In addition to basic mathematic skills, the teaching aims to provide the students with a “number sense” such as the actual meaning of a decimal place (a factor of 10 variation) or the actual non-arithmetic meaning of a multiplication problem.

For example, a student that has strong grasp on the principles behind arithmetic operations would see that 2.5 * 14 is two groups of 14, and half a group of 14 (7). In that way they are taught the concept, as opposed to simply multiplying 14 by 2 and 5 separately, adding the products, and then determining the position of the decimal point based on the number of digits to the right of the decimal point in the original problem.

This approach is designed to nurture an approach towards treating math as a pragmatic tool that is understood in its operational meaning, as opposed to being a procedural task.

Tests are used to examine and classify a students' progress. They must score at least a 97% on 3 of 5 tests to pass. A test should have 6 questions, but it's not necessary.

This process is clearly effective, but it is time consuming -both in instructor preparation and instructor record keeping. The tests all have to be individually designed, because no commercially available product allows the instructor enough control over the types of problems. Additionally, the fluidity of the process was less than optimal as it required the instructor to create volumes of new tests any time a modification, even a minute one, was made to the format.

The Project

The Learning Skills Academy approached us with a need for a software application to conduct automated and intuitive testing of the basic math skills, utilizing the unique process designed by the school. Of the product, LSA initially requested the following:

The program would generate 6-problem tests for any selected level and category. The student, through the use of a workspace, could work out the solution to a problem, and then confirm their answer. Both the answer and the workspace would be saved, for assessment and reflection. A profile for each student would be kept, containing their progress. Graphical and numerical descriptions of their individual tests and the 5 most recent tests of a level would be generated at command. The results had to be printable, in order to present to the parents during progress reports. Our own research provided many other requirements, which were addressed during analysis and development.

Methodology

Methodology Introduction

The Methodology section is intended to grant the reader a detailed view of what went into making the LSA Automated Testing Program. The section goes into the process of every major decision.   The section starts with our methods for analyzing the problem at hand and what needed to be done in order to satisfy the client's requirements. The methodology section then proceed to the Proofs of Concept that where required in order to determine the feasibility of implementing certain technologies and procedures into the program. With the back-end already being explored, the section then goes into the methods of User Interface development, from paper sketches to functioning UI that has already been merged with a more veteran back-end. The section concludes with a description of actual User testing at the LSA, as well as the results from the network testing that occurred concurrently to the user testing while at the LSA.

Initial problem analysis

Identifying Users

There were two end users expected to need consideration in this project: the student user and the instructor user.

The student user is a male or female student of the middle or high school grades. He is very comfortable using computers and is familiar with Windows and Microsoft software. He is capable of adapting easily to any product that follows Windows and Microsoft norms.   The student user may be color blind. He will have at least a third-grade reading level, a short attention span, prefer to actively manipulate his environment, and become frustrated quickly upon too much negative feedback.   He will also react with excess motivation to demonstrable success.

The instructor user is moderately comfortable using computers. He is familiar with Windows and Microsoft software, as well as many of the norms. He is very familiar with Excel. He is reluctant to give up paper testing altogether, because he wants to be able to see the tests on command, and be the final arbiter of grades.

Testing Levels

The complete list of testing levels, organized by category is under Annex E, provided by the LSA. Math problems are presented vertically, except for the division and fraction categories. The larger may appear on top or bottom of the presentation of the problem.

We first sought clarification of any levels that were ambiguous.

- When a category has levels "with zeroes," or "with carrying," any levels that do not specify such do NOT have zeros or carrying.
- Levels with zeros may include in one OR both of the numbers.
- There should never be more than nine digits in a problem, for an category or level. Fractions will never be more than 2 digits.
- Never divide by zero.
- "Easy numbers" are defined as multiples of 10 and 5.
- "Ugly numbers" are defined as everything not a multiple of 10 or 5.
- "Renaming a fraction" is the action of reducing a fraction to its simplest form
- If a decimal is less than 1, the program doesn't automatically put a zero before the decimal point.

Then we identified any levels that would require extraordinary functionality from our testing page.

- A student may need to move the displayed problem in order to align the decimals
- A remainder may have to be submitted.
- Fractions, Level 1, will need a picture displayed instead of an array of numbers.
- Division problems may have to be displayed in either of two different ways depending on the student's decision while taking a test. (LSA teaches it in two different formats.)
- A student may need to fill a grid square with repetitive borrowing or carrying in a problem.

The LSA did not have a concept in mind for how to deal with extraordinary functionality. They were not sure how it should be displayed, nor did they have a preferred method of interaction with the computer in any example. We brainstormed several solutions and presented the likely ones to the LSA. The school offered a few tentative opinions and we began testing to identify the best solutions.

Proofs of Concept

In the early stages of development, several of the design specifications called for technical solutions that could be solved by existing technologies. The challenge in this would be to properly utilize the existing technology and integrate it into our application in as robust a way as possible. Upon further review of the plan of attack, it was determined that proofs of concept would be required for each of these technical solutions in order to determine the technologies possibilities, and then to develop a working prototype for the development of a fully integrated module. The development of a module allows any element of the program to utilize it while only having to include the module in its scope.

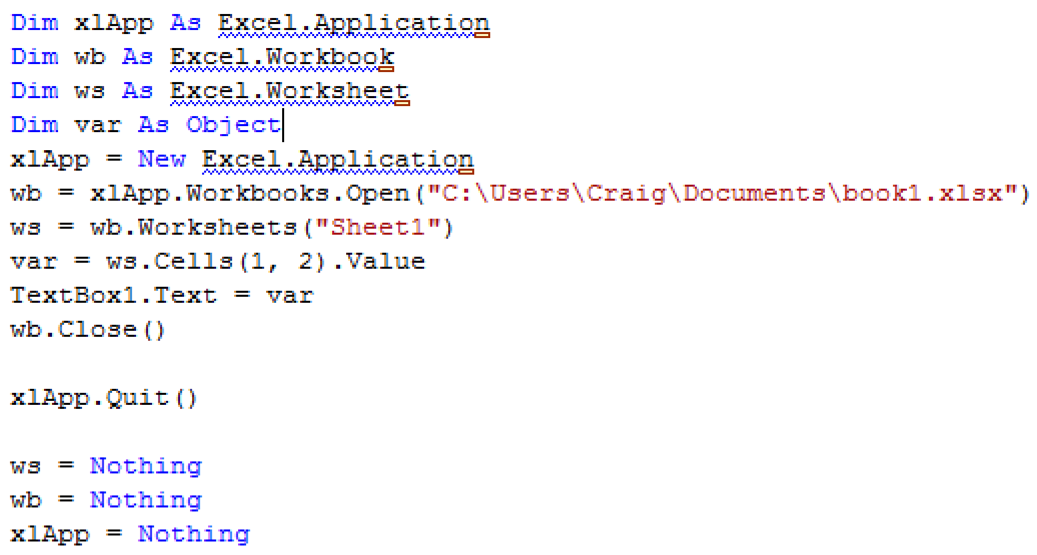
In brief, a proof of concept (PoC) is the demonstration of a principle whose purpose is to verify that some concept or theory is capable of being useful, or is at least feasible to implement. In the development of the LSA math program, this consisted of selecting a technology or technique that was deemed advantageous to use within the context of the program. This involved researching its usage in all pertinent contexts, and then generating a very primitive segment of code in Visual Basic, that would demonstrate that a certain technology or technique could be implemented in the immediate environment that the final product would run.

The primary proofs of concept that were generated early on in the development of the LSA Math program where: Visual Basic integration with Microsoft Office, Visual Basic utilization of Windows services and events, and XML generation and parsing within Visual Basic.

VB integration with MS Office

In the preliminary stages of development, it was concluded that the interface by which instructors entered and modified the problem set should be as easy to use as possible. With a majority of the instructors at the LSA having extensive experience in most Microsoft Office products, it was concluded that an Excel spreadsheet would be the best manner in which to have the instructors enter the problem data. This presented the need to read from an Excel file. In order to prove the feasibility of this, a proof of concept must be able to: open an Excel file, read specific contents out while disregarding any superfluous or erroneous entries, and then translate the cell values into an actual object complete with a full set of properly formatted and filled internal values.

The proof was developed by generating a simple form that allowed the user to read from a specific cell, and then modify the contents of a text box, and have the contained string written back to the origin cell on the specified Excel sheet. A few key concepts needed to be actualized in order for this to work properly. First, the *MS office Excel object Library 12.0* had to be properly imported into the namespace of the program and then each of the respective Excel objects (Workbook, sheet, page) needed to be instantiated as abstract object classes. Next, each of these instances of an abstract class needed to populate by linking them to an actual Excel document as seen below.



```
Dim xlApp As Excel.Application
Dim wb As Excel.Workbook
Dim ws As Excel.Worksheet
Dim var As Object
xlApp = New Excel.Application
wb = xlApp.Workbooks.Open("C:\Users\Craig\Documents\book1.xlsx")
ws = wb.Worksheets("Sheet1")
var = ws.Cells(1, 2).Value
TextBox1.Text = var
wb.Close()

xlApp.Quit()

ws = Nothing
wb = Nothing
xlApp = Nothing
```

Last, the value of an individual cell had to be extracted from a range of cells; this was accomplished essentially by saving the toString of the cell to a VB string, which could be implicitly cast into any other primitive.

The proof of concept proved to properly demonstrate two-way communication between the VB program and an Excel document via the *MS office Excel object Library 12.0* and a background instance of Excel.exe.

Visual Basic utilization of Windows services and events

The LSA math program had been initially designed to read a student's data off of a flash drive. Although this was later scrapped in favor of a networked system, it required a proof of concept in order to be seriously considered a viable option for data storage. This would require that the program recognizes that a flash drive is inserted, validate the data on it as being a student's testing data, and then read and write to the drive with any data mutations that would naturally occur as a result of the student taking a test. This presented several problems. If the there where two flash drives inserted into the computer, both with valid testing data, which drive was to be used? When a flash drive is inserted, how are we supposed to recognize that, and even if we can, how can we tell what the drive letter is? The solution would be determined in the second proof of concept.

The proof of concept had to be able to detect a flash drive, determine the drive letter, determine whether or not it was the last flash drive inserted, and then perform a file validation on a file on the flash drive. The flash drive proof of concept did exactly that. The program used a library that was able to monitor Windows events. When a drive mount event was detected, it would copy a state string from the Windows kernel and then, via a parsing algorithm, extract the last drive letter to be mounted. Then, the program detected and validated a text file written on the flash drive by ensuring that it was named properly, and that its timestamp was at an earlier date, and that it contained all the necessary fields in order to instantiate a experimentally created VB object.

During the testing of the program in the actual deployment environment, it would be determined that students should not have to log into the program since they have already logged onto Windows. Upon further analysis of this, it was concluded that since the student had already been authenticated by their Windows credentials, it was perfectly acceptable to manage the student's profile as a subset of their Windows identity. In order to accomplish this, the *Visual Basic utilization of Windows services and events* was revisited. This time, the username was discovered to be a global variable available within the namespace of any running program within Windows, once the program was verified to be running locally, the fact that the user was logged in was all that was required to automatically access the students testing data a proceed with testing.

XML generation and parsing within Visual Basic

During our initial meeting with the LSA staff, they had not explicitly specified a manner in which they wanted the student's data, as well as the problem set, stored. As described previously, the problem set was resolved to be stored within a Microsoft Excel file, but the student data was left as a decision to the programmers. In an effort to use concepts that the teachers and student would readily grasp, it was initially proposed that flash drives would be the medium by which student data was transferred from the student computer, and onto the instructor's machine. This idea afforded a stable storage medium as well as a physical means by which to control the student's data, but it still required the design team to develop a storage scheme or format by which to store the student data, the result was XML. The solution of using XML provided to be so robust, that when the program's student data storage location changed to a remote location, accessed via the local network, the XML module did not need to be touched and a wrapper only added to provide automatic location, validation, and revision control for the remote file.

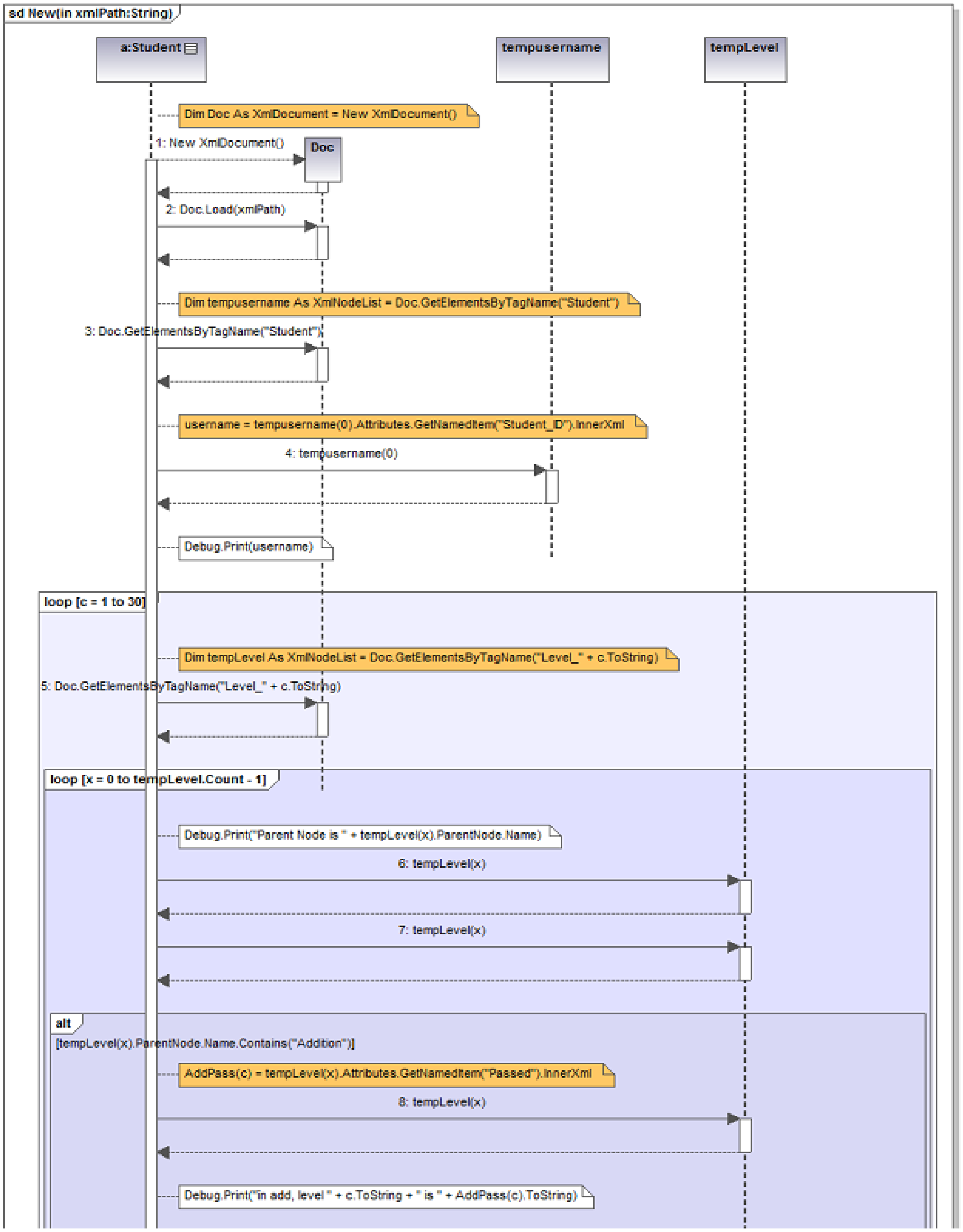
Extensible Markup Language (XML) is essentially a system by schema to make rigorously formatted electronic documents in a hierarchical node structure. XML affords the programmer the ability to make their own custom schema and have it read remotely, and have a file automatically parsed there by, or a programmer can use the default schema and build their own parser.   An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<family>
  <name gender="Male">
    <firstname>Tom</firstname>
    <lastname>Smith</lastname>
  </name>
  <name gender="Female">
    <firstname>Dale</firstname>
    <lastname>Smith</lastname>
  </name>
</family>
```

In this particular example, the node “family” has two sub nodes with attributes: gender = Male, firstname.value = Tom, lastname.value = Smith, and gender = Female, firstname.value =Dale, lastname.value = Smith. Essentially, the XML document stated that the family has two members, Tom and Dale Smith.

In Visual Basic, an XML file can be read in as a unique object that is essentially a tree of nodes. The nodes can then be traversed and elements and attributes of each node can be extracted via their names or values. In order to make a fully convincing proof of concept for XML utilization in Visual Basic, it would have to read in a XML file from a specified location, generate a VB object out of the XML data, and then modify the object, and write the object back to the XML file. The primary complication for the completion of this proof came from the multiple ways in which VB allows you to cast a XML object. Eventually, the DOM (Document Object Model) was selected with only the use of nodes, as opposed to the VB.type sub node and the VB.type fragment, both of these latter examples proved to be too cumbersome to reap any benefit from their used.

When the XML proof of concept was complete it had already developed into approximately half of the code necessary for the final XML subsystem. It was capable of reading in an XML file, validating its completeness, determining if it was newer or older than the data already in memory and subsequently update the proper copy, perform the required modifications, and then write the object data back to the XML file. The following diagram shows how the program can make a Student.vb object from an XML file.


sd New(in xmlPath:String)
a:Student
tempusername
templevel
Doc
1: New XmlDocument()
2: Doc.Load(xmlPath)
3: Doc.GetElementsByTagName("Student")
4: tempusername(0)
loop [c = 1 to 30]
5: Doc.GetElementsByTagName("Level_" + c.ToString)
loop [x = 0 to tempLevel.Count - 1]
6: tempLevel(x)
7: tempLevel(x)
alt
[tempLevel(x).ParentNode.Name.Contains("Addition")]
8: tempLevel(x)

```
Dim Doc As XmlDocument = New XmlDocument()
Dim tempusername As XmlNodeList = Doc.GetElementsByTagName("Student")
username = tempusername(0).Attributes.GetNamedItem("Student_ID").InnerXml
Debug.Print(username)
Dim tempLevel As XmlNodeList = Doc.GetElementsByTagName("Level_" + c.ToString)
Debug.Print("Parent Node is " + tempLevel(x).ParentNode.Name)
AddPass(c) = tempLevel(x).Attributes.GetNamedItem("Passed").InnerXml
Debug.Print("in add, level " + c.ToString + " is " + AddPass(c).ToString)
```

Figure 1

The preceding diagram shows how one facet of the XML subsystem takes in an xml file of a student's profile and produces a STUDENT.vb object.

UI Development

Development of the user interface (UI) began almost as soon as we accepted the project. Because the school was initially undecided on many of the details of the user interface, we used testing to determine their needs. In order to keep testing effective, we had to continuously develop and update a practical user interface.

Primarily we were concerned with the interaction between the student user and the testing screen, where they would work out the solutions to the test problems. We also had to address ease-of-use for the instructor, as the program's existence began with a desire to simplify the instructor's processes of student testing. To make the entire program effective, we additionally had to perfect the flow of user activity from login to testing to review and profile reflection.

The heuristic values of highest import to the Learning Skills Academy were:

- ✓ Ease-of-use: At no time should the program create additional difficulties for the students. The process should be as comfortable as the tried-'n-true paper method, but simpler.
- ✓ Flexibility: It should allow for the variety of diverse problem-solving methods taught to LSA students.
- ✓ User control: The program should never rail-road the user, but allow any task to be initiated at any point. The user should be able to easily "undo" an action, or leave a task without having to go through an extended dialogue.
- ✓ Age-appropriate aesthetics: Middle and high school students needing to re-learn math skills traditionally taught in elementary school may already be defensive. The program should protect their egos and reflect the adult approach they desire.

- ✓ Effectiveness: The program should test the students of each of the declared levels, provide accurate assessments, and compile all the graphs and statistics the instructor might want.

After meeting with representatives from the LSA school, and discussing their hopes for the project, we included the following values in our development of the user interface (always subordinating them to the LSA's primary concerns):

- ✓ Error prevention: The program should prevent the users from accidentally altering the program itself. It should anticipate the possibility of erroneous input and immediate disallow it, automatically correct it, or provide a submission process that negates the possibility.
- ✓ Help and documentation: Since the project was to be completed in the spring of 2010, the developers would not be available after that to provide support services. The students and instructors had to be able to fully understand the product on their own, or find the answers easily in our documentation.
- ✓ Recognition rather than recall: The program should be intuitive at all times, requiring a purely negligible adjustment period for first-time users.
- ✓ Security: The students should not be able to update their profiles or statistics outside of the intended testing. Students should not be able to alter others' profiles or statistics.
- ✓ Extensible: The program should provide for potential future versions, including an algebra level, a SMART board interface, or a portable product.

The Interface for Student Users

The testing page had to as closely resemble a paper test as possible. All of the student's normal actions in problem solving had to be readily available on the testing screen. From the beginning we provided a gridded workspace, with one problem displayed at its center at a time. Too much back-and-forth between the mouse and the keyboard can be tedious and time-consuming; any numbers or shapes required by the user were brought to the workspace using a drag-and-drop process. The required images are displayed beside the workspace. User-placed images are blue, to be distinct from the generated black problem.

Originally, the images dragged into the workspace snapped to a grid when they were dropped. The program then recognized the numerical value associated with the image. By anticipating the steps a student would use in solving a problem, the program would be able to "expect" certain values within the grid and identify where mistakes were made. The snap kept the workspace organized and standardized.

Later testing showed us that the design was limiting. For example, when the question of where a decimal on a grid was put to the LSA, no definite answer could be provided. Students instinctively placed the decimal to the immediate right or left of a number, on top of the grid line between two numbers, or in its own grid column. The school had no reason to teach one specific format over the others. There was also a problem of image size, especially in reference to mixed fractions, carried-over digits, and replacements for numbers that were crossed out.

In the end we determined that the workspace should allow for dynamic manipulation. It doesn't assign any numeric value to the work the student does; instead the user can treat the workspace as scratch paper. The grid is only an image that the numbers and shapes can be dragged on top of. In this way, the user can place the decimal wherever he chooses.

All of the images brought to the workspace are able to be re-selected, moved, deleted, nudged, rotated, and resized. Any of the manipulations aside from re-selection, and drag-and-drop, required key-strokes from the keyboard.

The Interface for Instructor Users

The LSA math instructor currently uses Excel sheets to design paper tests. It was the natural solution to use Excel as the primary bank of test problems. If the levels ever needed to be changed, the template would be familiar.

Testing showed us that the instructor, upon viewing the results of a student's test, wanted to have the authority to override the computer's conclusions. In one example, the student correctly solved a math problem, but submitted a typo as the answer. The instructor wanted to save the solution to that problem as "correct" and immediately show the updated results to the student. In the final design the instructor could change the recorded answer for a problem by clicking a button beside the displayed results and submitting a password in the subsequent window.

It was originally our intention to carry the program on the students' individual flashdrives, rather than the school's computer network. The original design required that the flashdrives be submitted to the instructor – either to update his master bank of student profiles, or so that he could view the profiles individually. The first design had the advantage of being highly portable; students would be able to take the program with them for homework. It also didn't depend on the anecdotally-unreliable school network.

Later the desire for easy profile updates that allowed the instructor to view all of his students' profiles at any time altered the original plans.

Flow of user activity

A secure log in was questioned many times. First, the physical security was provided by storing the program on individual flash drives. Then, when we determined that the best route for data storage was to use the school's network, LSA suggested that the atmosphere of trust and personal responsibility shared by the students wouldn't require them to memorize a particular password for the math program. Our concern that a student might take tests –even by accident- under the wrong profile kept the issue unresolved for some time.

The final design automatically identifies the student when the program is started. It searches the networked file system for the user logged in to the PC in use. The user has the opportunity to log out of a profile name, if it is the wrong one by chance. However the user is not required to memorize a password, nor are they given the opportunity to log in as another student by mistake.

Any screen the user views gives the user the opportunity to view the profile screen, log out, or select a new test. However the first screen they are presented with is the Select Test screen. Should they choose a test category and level, they may move to the testing page and begin. Completion of the test provides a results screen, reviewing the test recently completed. Afterwards the student is redirected to their profile page to see the overall testing results, now updated.

UI Testing

User testing was a continuous process throughout analysis, research, and development.

Our initial project requirements were entirely insufficient from a software design standpoint. The LSA had no prior experience with declaring project requirements; the details they DID have in the client conceptual model had to be coached out of them. Other details that were not included had to be identified and, by a process of narrowing options, a decision determined.

To do this, we used an abstracted testing method. First on paper sketches, we showed them what the current model might look like. Then we asked them to perform a user's actions while we observed their assumptions of how the program might function. When we, from a coding standpoint, were aware of more methods of functionality than the LSA was conscious of, we neutrally provided the different methods for them to evaluate.

Beta testing was very similar since we were attempting to ensure that the final product would be intuitive, easy to use, and require a negligible adjustment period. We offered a range of possible actions to effect a user's goal and asked the testees to do a task, without giving them any clues outside of the interface that might influence their actions.

When such testing resulted in unique actions for every person tested (for example the decimal place deposited in different positions on the grid), we considered providing a variety of options. When the testing provided a consistent result (for example the mechanics of the drag-and-drop system being second nature), we solidified our design choice. When the testing returned completely unexpected results (for example students submitting the answer "sixteen" written in long hand rather than "16" numerically), we took the opportunity to provide more instruction on the screen. If a user searched for something that wasn't provided, we added it to the next round of testing.

Final testing took place with users who had roughly ten minutes of familiarity with the program (either from previous testing, or from a mix of observation and active exploration). We gave them the sole instruction to take a test from a given category and level. Then we observed their actions. None had any trouble navigating the screens or manipulating the workspace, so we judged the success based on the work and answers that the instructor expected from each student.

Networking

As previously mentioned, it was determined that it was most advantageous for the Student profile, including testing data, as well as the problem set to be stored on the local network and read at each instance that the data was required. In regards to the problem set, this allowed the instructor to make a revision to the Excel file and as long as this was done prior to the student selecting the individual test that they want to take, the student's test would reflect these changes. This was the best choice as far as the instructor interface is concerned, because with the use of a desktop shortcut, the instructor would appear to be editing a simple Excel file on their local machine, something that they have done hundreds of times. In regards to the student profile, the hosting of the data on the network afforded increased persistence of data due to the backups maintained by the network contractor. More importantly than stable storage, the networking of the student profile allowed the student to use the program on any computer throughout the LSA and have the experience be seamless since all their data virtually localized (appears to be on the local disk) when the logged on. The basic flow of the instructor controlled Excel problem set file can be seen below:

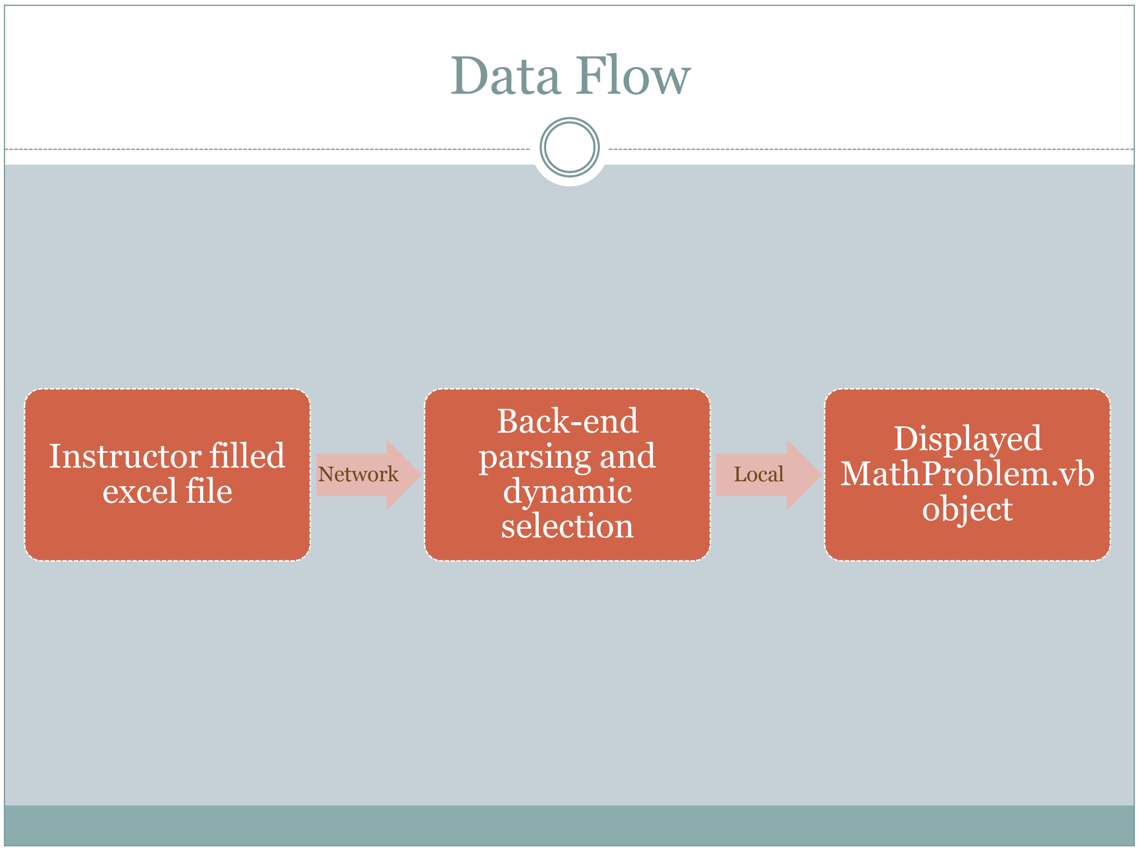

Data Flow
Instructor filled excel file
Network
Back-end parsing and dynamic selection
Local
Displayed MathProblem.vb object

Figure 2

During early client consultations, the LSA relayed the information that each student had a remote “S:\” drive, identical in practice to the “Toaster” drive that is used at WPI. This information was taken from the meeting and implemented on a test machine running a simulation of the client’s operating environment: Windows 7 64-bit, 500GB HDD, 4GB RAM, 2.56Ghz Intel Dual Core Processor, and a software replicated 10GB S:\ drive. The test machine was also setup to have several login usernames/passwords that had access to the same S:\ drive. Through experimental development it was discerned that by saving the XML file of the student’s profile with a time stamp and unique test data on the S:\ drive under a file name derived from their Windows username, all predictable file conflicts and versioning problems across the network could either be averted, or resolved upon detection. In order to maintain the consistency of the remote file system, automatic file validation was instantiated via a 3-point check: Size, Time, and Content. The remote files must have a valid timestamp in the past, they must not be too small or too large, and they must contain all the necessary fields required for their operation.

A working version on the test platform was generated and was taken for testing in the actual environment for testing.

When the testing of the networked components started, the signs were promising. The program was able to detect the logged in user and see the S:\ drive. The problem came when the program attempted to write to the S:\ drive. When the program was run from the VB debugger, it was discovered that it was a write permission error. After some coordination with the LSA faculty, the test student profile that was being used on that particular machine was given Admin privileges, yet the problem persisted. Suspecting that the problem was localized to the testing only student profile, a regular student was asked to log-in and attempt to run the program, the student was given the same error message. After asking the student to attempt to write to their S:\ drive, it was discovered that the entire student body was not able to write to their S:\ drive. Upon further investigation, the student revealed that the files that they wrote a week earlier to the "Documents" folder of another computer (a local folder on most PCs), were accessible on a different matching. After further questioning of the faculty, it was concluded that in an effort to stop students from saving on the local HDD, the company that they contract their network administration out to, had made each student profile networked, so that all files contained under the directory of STUDENTNAME were loaded from the network every time the student logged in, and dynamically updated as the student modified the contents. This made the S:\ drive obsolete, so they, without telling anyone at the LSA, made the drive read-only so students could read their old documents.

Once the new network structure was discovered, the program was modified on site, and in less than an hour, a working version was implemented on the LSA network and more than one sternly worded email was sent to the LSA's networking administrators from the LSA administration. From that point forward, all network testing, involving the student profile, was able to be replicated on any Windows 7 machine and with any login.

Although the newly discovered structure of the network allowed the student profile to be treated as a local identity, even more so than a mounted drive, it still remained that the Problem set would have to be loaded from a remote location each time the student started a test in order to insure that the problems where the most recent revision as specified by the instructor. This was accomplished with much less confusion than the student profile. It consisted of allowing all students on the network to read a single directory on the instructor's networked drive, this directory contained a single file, LSA.xlsx. This file, with only a need for minimal verification, would be read into the program and utilized, as per the procedure laid out in the preceding diagram. The result was an LSA Math Testing Program that was fully integrated into their network and did not rely on any physically managed medium for data management.

Client UI Testing

Client Testing of the product (not the testing designed to discover the client mental model) occurred at regular intervals throughout the development process. We visited the school no more than once a week to use the math instructor's free period. The tests were prepared in advance, with specific goals in mind.

For example, one day of testing was designed to answer the following:

1. Where do students think the decimal belongs in a grid format?

2. Will students recognize the purpose of the vertical bar button?
3. If many images are overlapping in the workspace, how difficult will it be to select one image?
4. Do students prefer a button that says "Next" or "Submit"?
5. Is the space provided in the basic-sized workspace sufficient for division problems?
6. Will using the CTRL+Arrow keys to complete an action be too difficult? Would it be better to pursue a mouse click instead?
7. Is the scale of the grid and images too large? Does it look immature?

Each time we pulled a pair of students that had never seen our program to test. The instructor would brief the students on our goal to digitalize the math tests, and explain the process of user testing. Emphasis was made to the students that their actual math skills were not being tested at the time. And in fact, the students weren't being tested, the program was being tested. They were asked to think about the program critically when they used it, and imagine having to use it all the time during school. They were also encouraged not to hold back any ideas, complaints, or criticism to spare our feelings.

The students were then brought one-at-a-time before a laptop with a medium-to-large screen, provided a wireless mouse, and asked to begin the testing prepared for that day. While they tested, one member of the team took copious notes regarding the student's actions as well as the prepared questions. (All testing information was recorded by student name and date, to facilitate review at a later time.) If the student became confused or appeared to be frustrated, they were given an explanation or prompt regarding the issue. The process was unobtrusively observed by the instructor and school director.

In the previous example, the answers to questions 1 – 5 were observed by letting a student take a prepared test that included both a decimal problem and a division problem. Data regarding question 6 was gathered by giving the student a description of the current controls available (including CTRL+L or R arrow, to rotate an image) and asking the student to try using them. The student was than given a chance to volunteer any observations about the "look" of the program. If they didn't provide an opinion of the scale of the grid square or images, question 7 was asked directly.

The second student, who until then waited in another room, followed the same process.

When both were done, they were given the opportunity to explore the program at their leisure. Many used the opportunity to ask questions or volunteer opinions that they felt uncomfortable saying during an apparently "professional" testing session. Notes were recorded then also.

On most occasions, at least one student who had tested with us previously stepped in. They performed the day's testing as well. They were then given the same chance to explore the program, as we encouraged them to compare the new version to the last one they saw.

We used these sessions to make observations outside of our prepared concerns, often leading to future avenues of testing. On the day we tested the provided example, we noted that the submission field was initially overlooked by two students. They didn't see it and didn't realize they were meant to supply the answer in it. This led us to 1) try changing the size and placement of the field and 2) try providing a directive blurb.   That same day we also noticed that the first thing every person did when given the chance, was to use the workspace to create art out of the images they could drag onto the screen. That led to the consideration of putting a timer on the testing page.

Once the students were done, the instructor took a turn interacting with the program. He tested the student-specific parts as well as his own. He then gave his own opinions about the program, and offered further observations of the student testing that we may have missed because of our lesser knowledge of the students themselves. (For example, one student who suffered from significant confusion over a division problem we tested, hadn't realized the program offered the opportunity to solve the problem in the way he was used to. What we had witnessed was a small display of panic disrupting all of his previous knowledge of the program.)

Final Analysis

The final analysis is an analytical analysis of the Learning Skills Academy Automated Math Testing Program's overall effectiveness. It is meant to discern the overall impact of the piece of software in high enough detail that causal relationships can be established. This was done by reviewing all test data throughout the development of the program and determining to what extent each of the clients objectives were met.

The first segment of functionality to be examined is the XML and Excel networking. The networked transmission of data greatly simplifies the LSA's staff's ability to manage and control the testing process, as well as the great deal of data it generates. In actual testing, the networked component of the program used an extremely small amount of bandwidth (the files are less than 100kb in most instances), which led to an undetectable amount of latency when the program required real-time access to network resources. In addition to being efficient, the networked component was robust. In actual operation, it was, and still is, able to detect and recover from attempting to access any corrupt or missing files. In short, the back-end of the LSA Automated

Math Testing Program functions well within the limitations of the machines it is run on and performs its processes in a robust and effective manner.

The back-end was a success for the simple reason that it supported the UI, and had an extremely small failure rate after initial testing, but the actual User Interface is in a whole different category. The UI does not have the luxury of being graded on a Pass/Fail standard. The UI has the potential of being unusable, but still theoretically capable of offering required UI functionality, or it can so intuitive that documentation is a waste of energy. The final UI in the LSA Automated Math Testing program proved to be closer to the latter. When student were presented with the final product and told to take a test, they were able to navigate the program with no help, and after a few refinements, very few test subjects were able to reach a point where they required instructor assistance. As derived from early deployment results, the UI is designed well enough that one instructor can manage a class of 6 LSA students, some testing and some not, without the need for assistance -something that is not possible with any other teaching or testing aid on the market.

Conclusion

The LSA Automated Math Testing Program was an exercise in pragmatic software development. The programmers met with the client, determined what their needs were, recognized what the intermediate tasks were, developed a beta, tested the beta, and then integrated and deployed the software; fulfilling each of the client's needs.

When the project was initially accepted, the LSA required a program that would take over their pencil and paper math testing procedure. After several emails and phone calls, the testing procedure was documented well enough to create the Use Cases (see: Appendix A) for the finite states of a testing page. From this point forward, the programmers were designing the piece of software, not on what had initially been laid out in a proposal, but by what would actually fill the need of the client. This is the primary defining characteristic of the LSA project; it was a real-world exercise in client-need driven software development.

As previously stated in the Final Analysis, the program met the requirements of client and is being adopted at a rapid rate with full deployment expected by fall 2010. This state was not met by making exactly what the client said they wanted; if that had been the case, a test-based keyboard driven solution that runs from a floppy disk would be the final result. Instead, the final product is a networked and extensible, touch-screen-compatible, and efficient solution. During the course of the project, this was observed to be due to the fact that the LSA administration, as with most average end-users, had no experience in the process of developing software; they were only able to think in very large and abstracted pieces. For example, the initial project idea from the client was to make digital flashcards. This concept was extremely vague and provided to be a source of ambiguity. From this point, it was the programmer's duty to deduce what the client wanted to gain from the use of the program. After presenting the client with enough potential use scenarios and recording their desired outcome, a logical framework was established that allowed for the freedom of design while working towards the client's needs.

Through the development of the LSA Automated Math Testing Program, UI design and testing was always at the forefront of the task list. UI development is a highly analytical process that involves looking at years of form design research, generating metrics for the interface, modifying the interface in order to improve those metrics, yet there is never a substitute for actual user testing in the deployment environment with the target user base. The constant testing with the actual users at the LSA was a key action that led to the overall success of the UI. By involving the end-user at each stage of UI development, any poor design choices were avoided as early as possible, and beneficial features should be conceived more easily with the actual user working with the program and the programmers at the same time. The result of the UI testing regime was an effective an efficient UI that met the LSA's needs with minimal designer-client conflict.

In conclusion, the LSA Automated Math Testing project was an exercise in pragmatic, client-need driven programming. The programmers were able to take an ambiguous set of ideas from the client and formulate a logical flow of user interaction from that, and then develop the entire program to support and implement this UI flow. UI and back-end testing was conducted on-site at the Learning Skills Academy during the every stage, resulting in an effective and efficient UI. These elements culminated in a software application that fully met the requirements of the client, the Learning Skills Academy.

Future Work

Throughout the development of the LSA math program, extensibility and adaptability were always a goal when engineering any given code module. Although all code was designed in as abstract and modular a manner as possible, two specific features, to be added in the first update, were considered at all times: Touch screen and SMART board compatibility, and the portability of the system to any network and even its portability on any removable media. The prior would seem to be a natural step with touch UIs becoming the norm in most new technology, but that was not the primary reason for its consideration. The students at the LSA are there because traditional teaching techniques were incapable to helping them learn, often due to concepts that are too far abstracted. Their experience is similar to a freshman Computer Science student at WPI being expected to know how to program an Artificial Intelligence in Scheme for their first assignment ever in the Computer Science program. Touch interfaces offer a degree of tangibility to any problem that can help the student deal with abstract concepts in a concrete manner. Secondly, the portability of the program to work almost anywhere allows the program to be used at any institution with a short setup process. This step is natural as the program would then be allowed to become a production level tool for specialized instruction. Coupled with network portability, is medium portability. Allowing a student to take the program home on a flash drive affords the LSA the opportunity to take the math learning, and subsequent testing experience out of the library and classroom, and into the student's home.

Touch screen interfaces have been used in education since their inception. They were initially used to help instruct students who did not have fine enough motor skills in order to operate a pen or pencil. At the LSA, SMART boards, projector screens with a built in capacitive touch layer, have been a topic of purchase for the past year. After a demo of one of our early program prototypes was done on a touch screen, the LSA administration immediately saw the worth of being able to use the software as a classroom teaching aid, and not just a testing tool. Ironically, the only feature of the program that requires a keyboard was specified by the client. The LSA administration required that the program forces a student to type in their answer in an answer box after deriving the answer in the workspace. This was done to ensure the student's understanding of the material as well as their attention to detail, an essential life skill. In order to modify the program to be fully touch UI friendly, the answer box would have to be removed, and the answer would have to be automatically recognized in the workspace; this can be done with the addition of a object.equals function on each mouse release of a number in the workspace.

Extensibility would require a more technical solution, but nonetheless it would be accomplished with tools that have already been implemented within the code. In order to make the program work on any network, the paths of the individual student's data would have to be discerned, as well as the location of the problem set. This could easily be solved by a helper program that prompts the network administrator for all of this information and then stored it in a location that each client machine can access. Then all the client machines would have to do, is be told this location via a configuration file the first time they are ran, and the program would then be ready to run on the new network. The second facet of portability that can be readily implemented is the operation of the program from a flash drive. The program in its current state can do this, but will throw an error when it detects that it is not on the network. In order to circumvent this, the program would need to detect when it is not on the network, and then load all values from a local copy that was updated the last time it was on the network. Subsequently, all test data accumulated while away from the network would be uploaded to the main database once the student inserted their flash drive into a school computer.

Extensibility and adaptability are two features that any software engineering team must keep in mind, if not focus on, while designing any piece of software. The LSA Math program has been developed in such a manner that it is inherently extensible and adaptable buy creating abstracted and modularized code that is already designed to accept several version 2.0 add-ons without restructuring of any facet of functionality.

Bibliography

"About Us." *Learning Skills Academy*. 2010. Web, < http://www.learningskillsacademy.org/home.asp?Page=65&Parent=1>.

Jacques, Mathieu. "An Extensible Math Expression Parser with Plug-ins - CodeProject." *Your Development Resource - CodeProject*. 25q Apr. 2005. Web. 29 Nov. 2009. <http://www.codeproject.com/KB/recipes/MathieuMathParser.aspx>.

McManus, Lisa. Personal interview. 24 Dec. 2009.

---. Letter to the author. 20 Apr. 2010

MICROSOFT. "How to Automate Excel from Visual Basic .NET to Fill or to Obtain Data in a Range by Using Arrays." *Microsoft Support*. 2009. Web. <http://support.microsoft.com/kb/302094>.

MICROSOFT. "MSXML." *MSDN: Microsoft Development, MSDN Subscriptions, Resources, and More*. 2010. Web. <http://msdn.microsoft.com/en-us/library/ms763742%28v=VS.85%29.aspx>.

Petroutsos, Evangelos, and Mark Ridgeway. *Mastering Microsoft Visual Basic 2008*. Indianapolis, Ind.: Wiley/Sybex Pub., 2008. Print.

Rothous, Doug. "The Visual Basic Team : VB XML Cookbook, Recipe 1: XML Transformations Using XML Literals (Doug Rothaus)." *MSDN Blogs*. 1 Feb. 2008. Web.

<http://blogs.msdn.com/vbteam/archive/2008/02/21/vb-xml-cookbook-recipe-1-xml-transformations-using-xml-literals-replacing-xsl-for-each-doug-rothaus.aspx>.

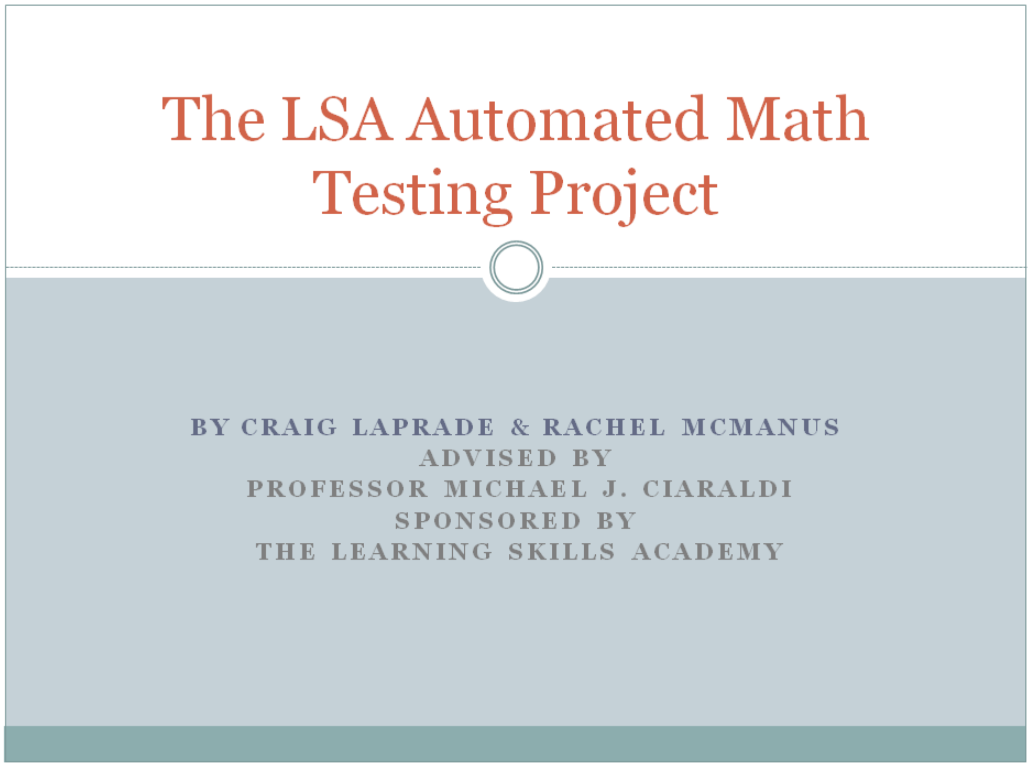
Stone, Deborah L. *User Interface Design and Evaluation*. Amsterdam: Elsevier, 2005. Print.

Walkenback, John. "Excel - a VBA Function to Extract Text from a String - Using MID and FIND to Remove a Word from a Sentence." *Help and Tips Using Excel Spreadsheets and Access Databases*. 2004. Web. <http://www.meadinkent.co.uk/xlextracttext.htm>.

Annexes

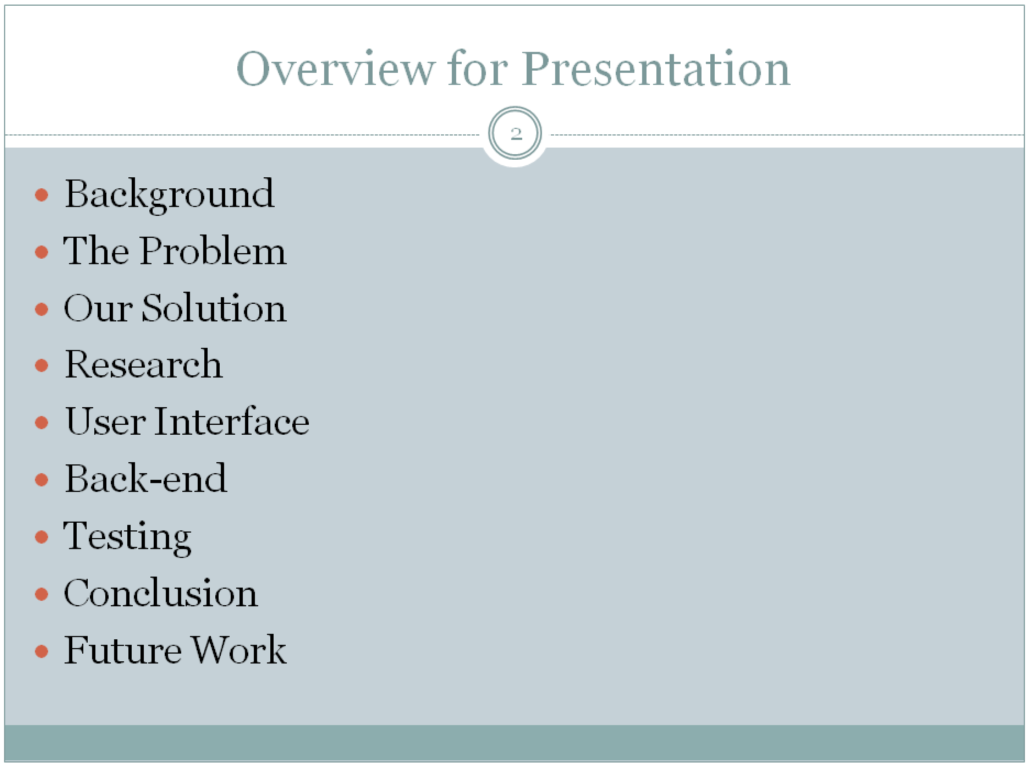
Annex A:

Slides from Project Presentation:



The LSA Automated Math Testing Project

BY CRAIG LAPRADE & RACHEL MCMANUS
ADVISED BY
PROFESSOR MICHAEL J. CIARALDI
SPONSORED BY
THE LEARNING SKILLS ACADEMY



Overview for Presentation

2

- Background
- The Problem
- Our Solution
- Research
- User Interface
- Back-end
- Testing
- Conclusion
- Future Work

Background

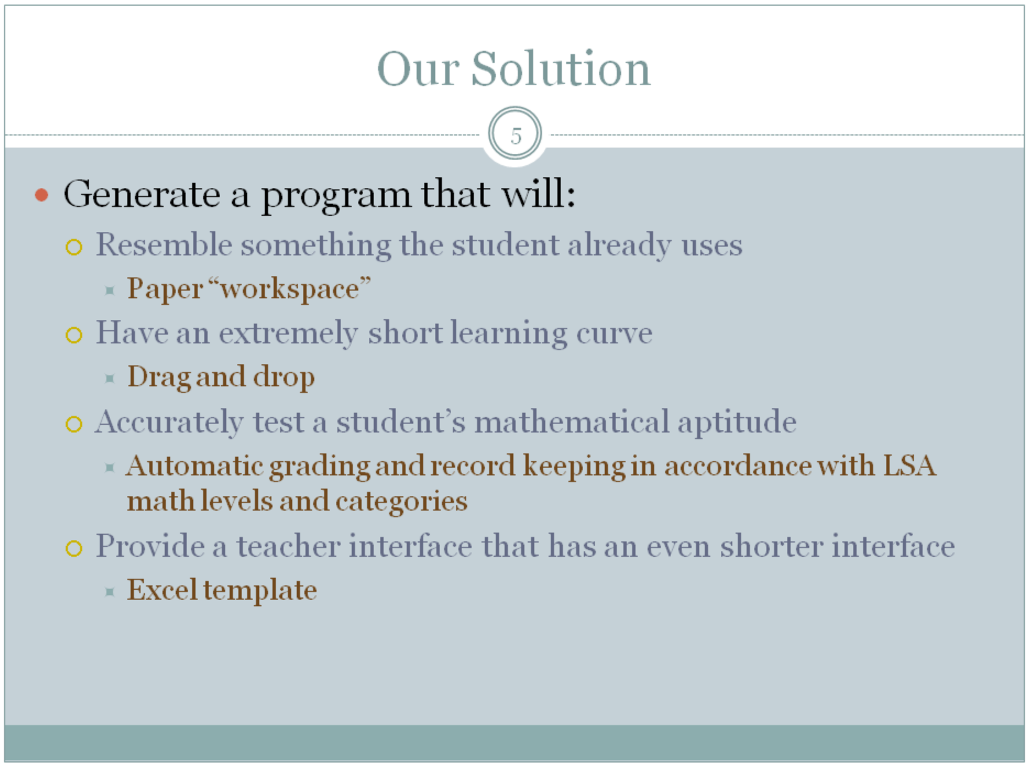
3

- What is the Learning Skills Academy
  - Private Middle and High School for students with learning disabilities
  - Located in Rye, NH
  - Has a highly specialized and unique approach and teaching style
  - Approximately 45 students and 20 teachers

The Problem

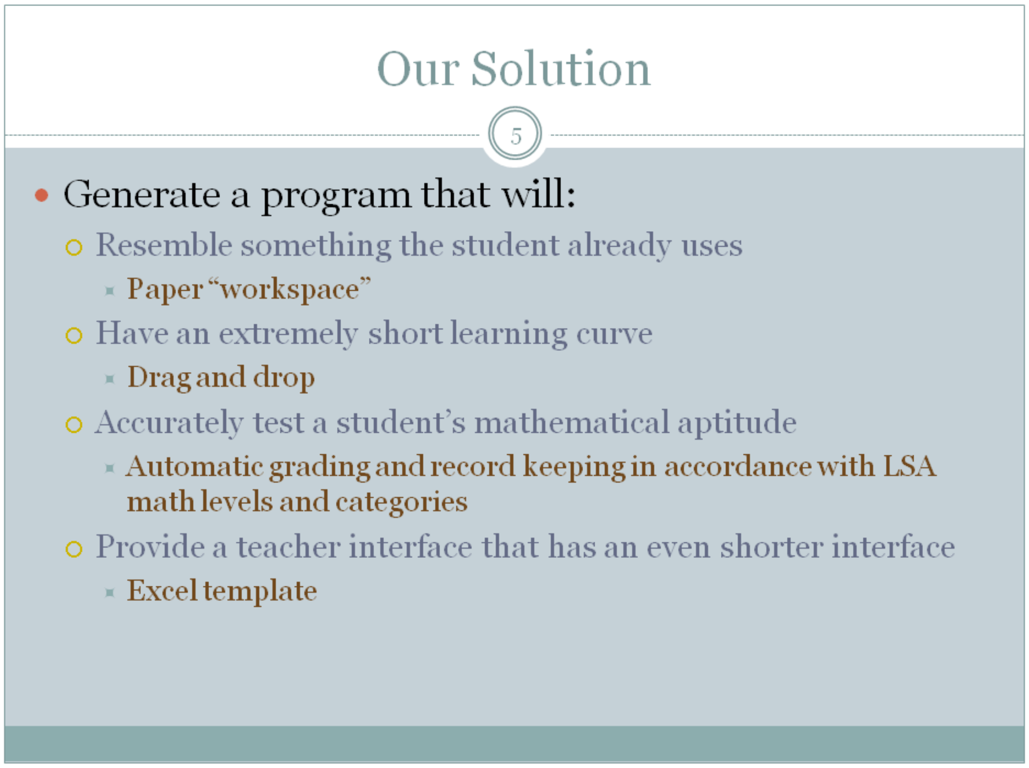
4

- LSA used to test a student's math proficiency with numerous paper tests
  - Monotonous
  - Time consuming
  - Repetitive
  - Perfect candidate for computerization
- Problem
  - The LSA's highly unique teaching style cannot conform to many software applications' "rubber stamp" approach to testing



Our Solution

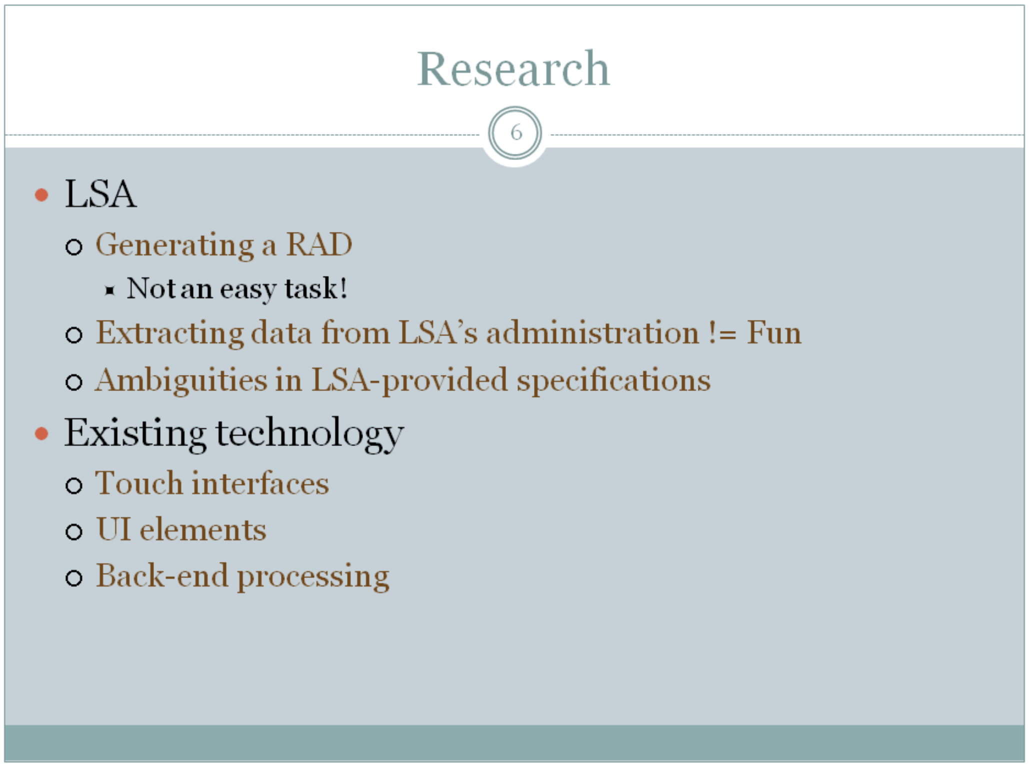
- Generate a program that will:
  - Resemble something the student already uses
    - Paper "workspace"
  - Have an extremely short learning curve
    - Drag and drop
  - Accurately test a student's mathematical aptitude
    - Automatic grading and record keeping in accordance with LSA math levels and categories
  - Provide a teacher interface that has an even shorter interface
    - Excel template



Our Solution

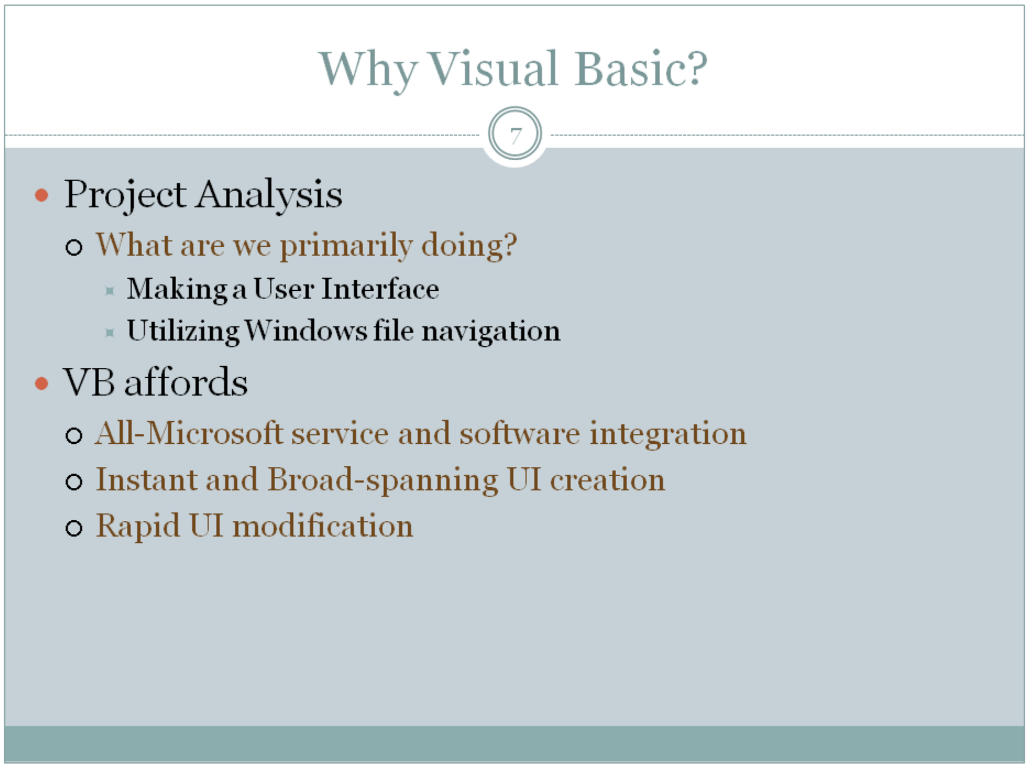
5

- Generate a program that will:
  - Resemble something the student already uses
    - Paper "workspace"
  - Have an extremely short learning curve
    - Drag and drop
  - Accurately test a student's mathematical aptitude
    - Automatic grading and record keeping in accordance with LSA math levels and categories
  - Provide a teacher interface that has an even shorter interface
    - Excel template



Research

- LSA
  - Generating a RAD
    - Not an easy task!
  - Extracting data from LSA's administration != Fun
  - Ambiguities in LSA-provided specifications
- Existing technology
  - Touch interfaces
  - UI elements
  - Back-end processing



Why Visual Basic?

- Project Analysis
  - What are we primarily doing?
    - Making a User Interface
    - Utilizing Windows file navigation
- VB affords
  - All-Microsoft service and software integration
  - Instant and Broad-spanning UI creation
  - Rapid UI modification

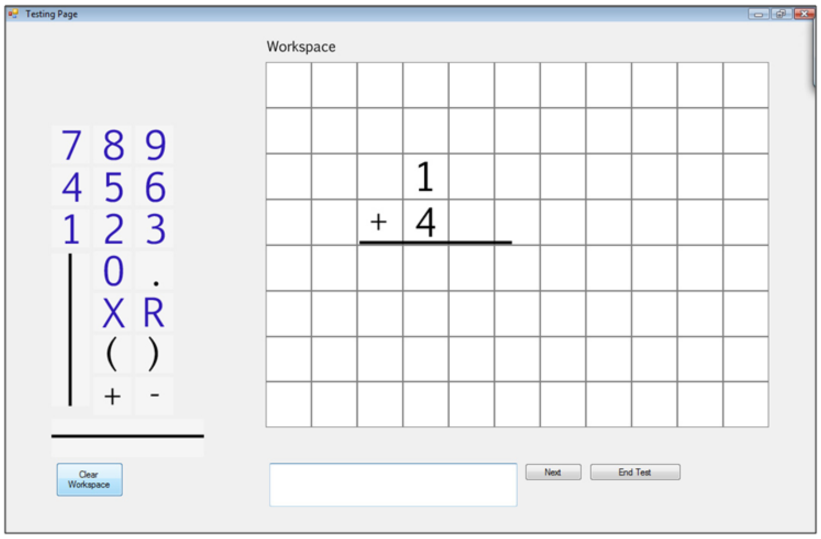
User Interface

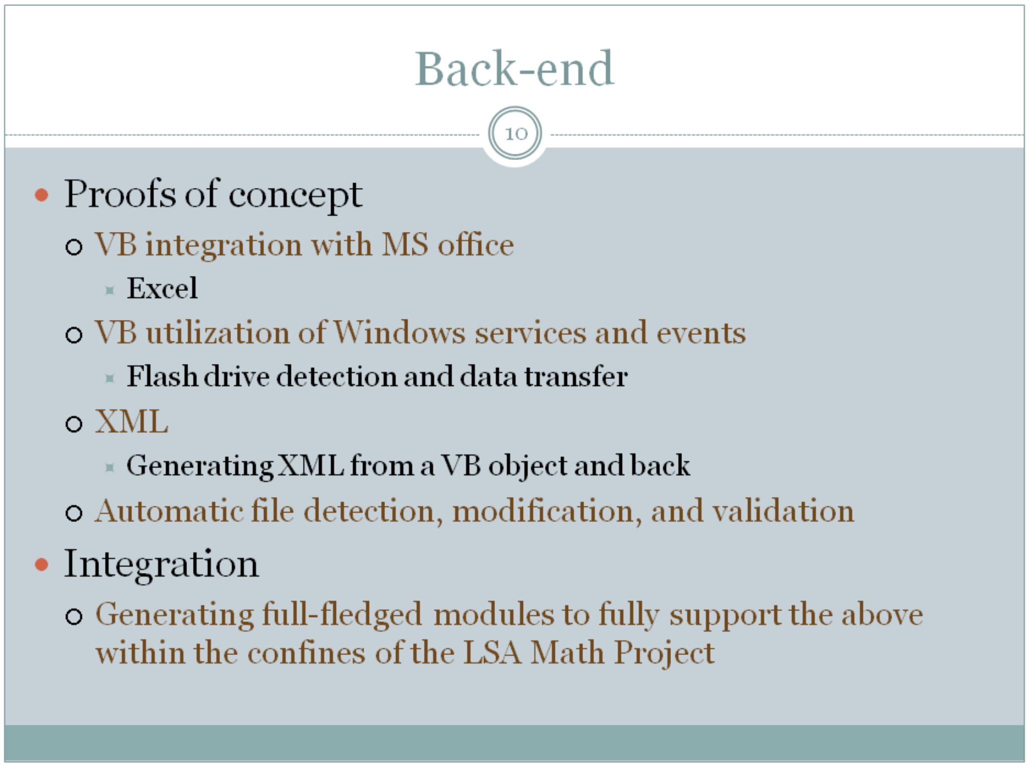
8

- Testing Page
  - Closely resembling a paper test
  - Easy for the student to manipulate
  - Ensure that the UI did not hinder the students ability to take the test
  - Resembles the teacher's style of teaching
    - Required support for several additional actions
      - Two different presentations of the division problem
      - Manual submission of the answer
- UI had to be easily manipulated by the instructor
  - Instructor-corrected answers
  - Teacher-defined problems via spreadsheet

Testing Page

9

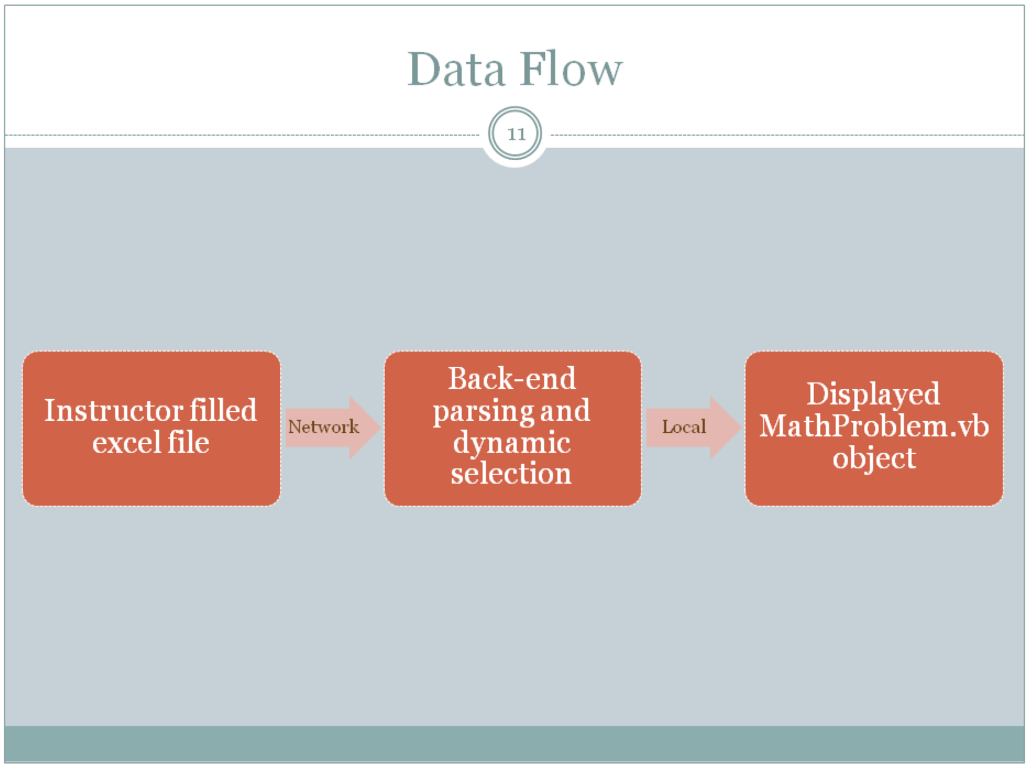




Back-end

10

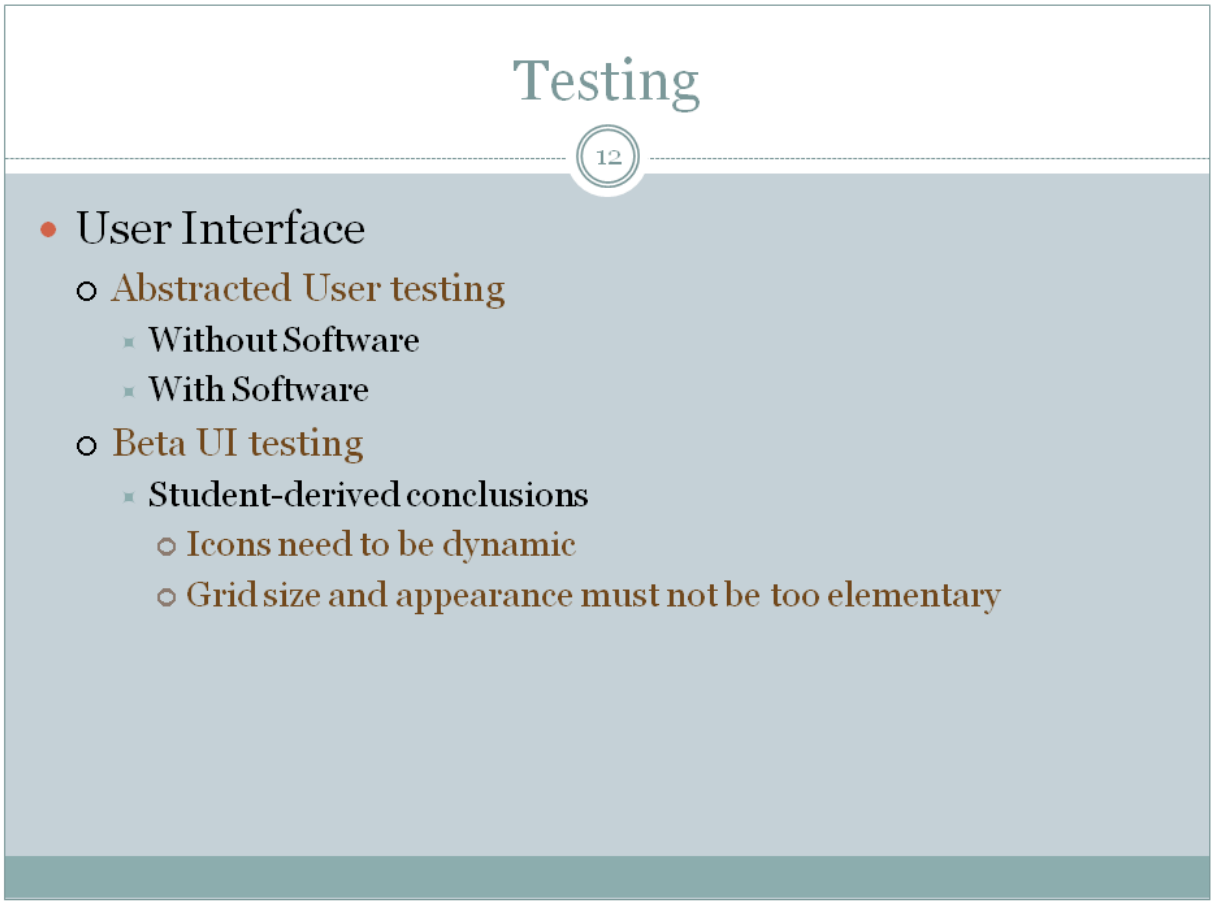
- Proofs of concept
  - VB integration with MS office
    - Excel
  - VB utilization of Windows services and events
    - Flash drive detection and data transfer
  - XML
    - Generating XML from a VB object and back
  - Automatic file detection, modification, and validation
- Integration
  - Generating full-fledged modules to fully support the above within the confines of the LSA Math Project



Data Flow

11

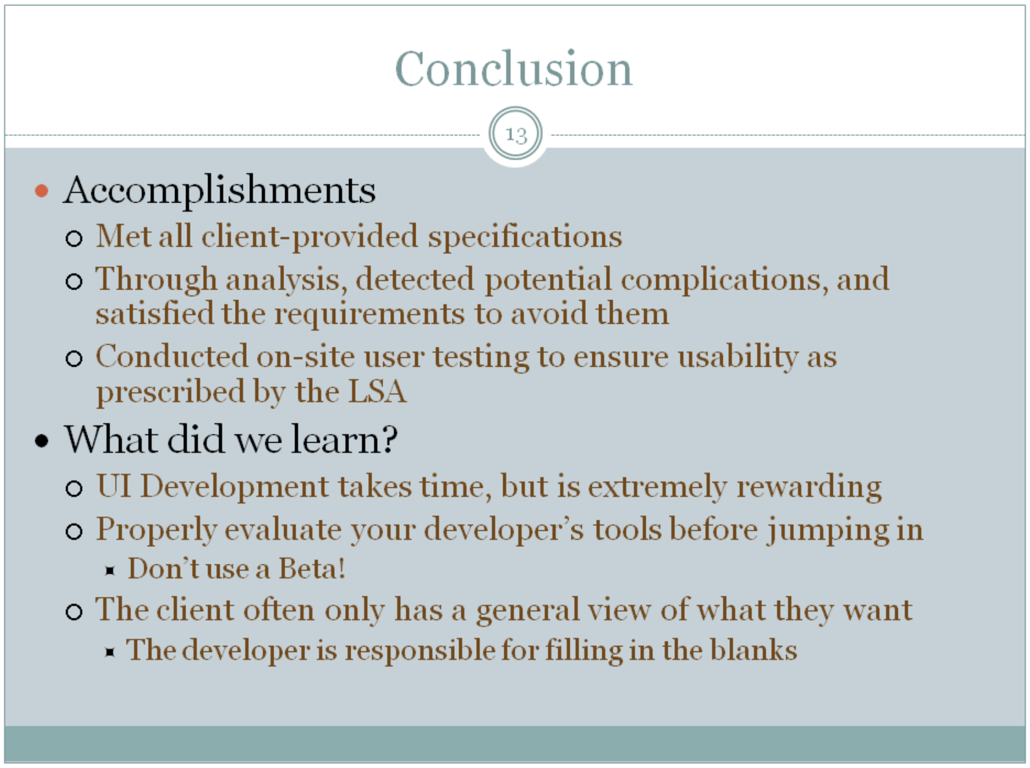
Instructor filled excel file
Network
Back-end parsing and dynamic selection
Local
Displayed MathProblem.vb object



Testing

12

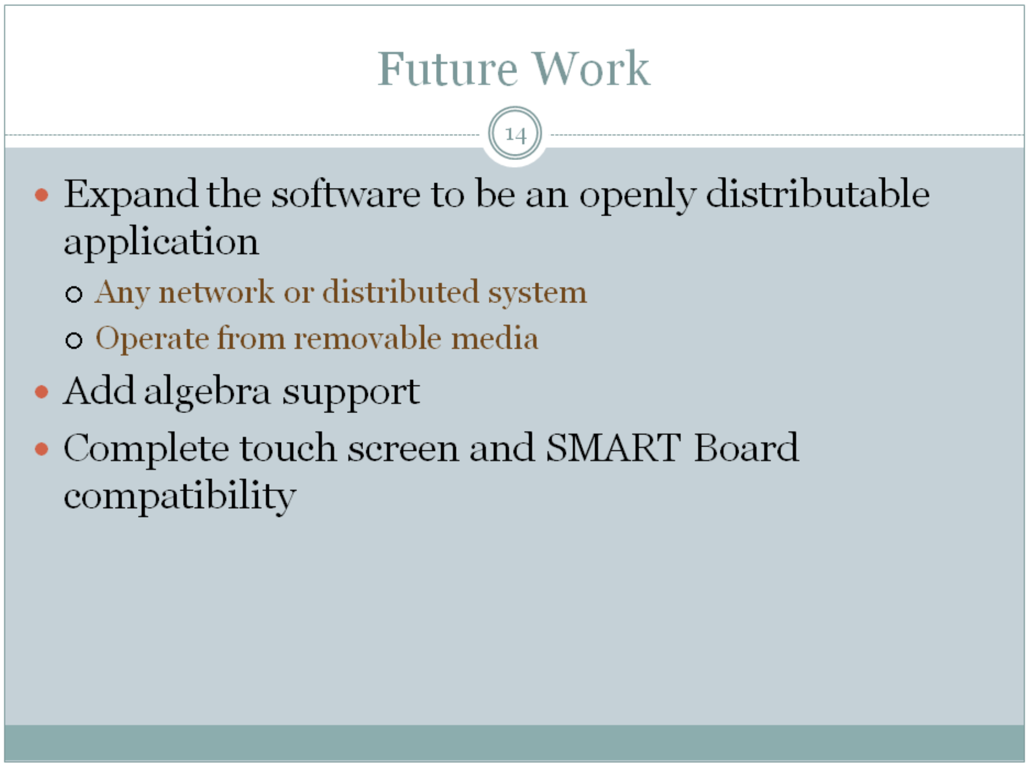
- User Interface
  - Abstracted User testing
    - Without Software
    - With Software
  - Beta UI testing
    - Student-derived conclusions
      - Icons need to be dynamic
      - Grid size and appearance must not be too elementary



Conclusion

13

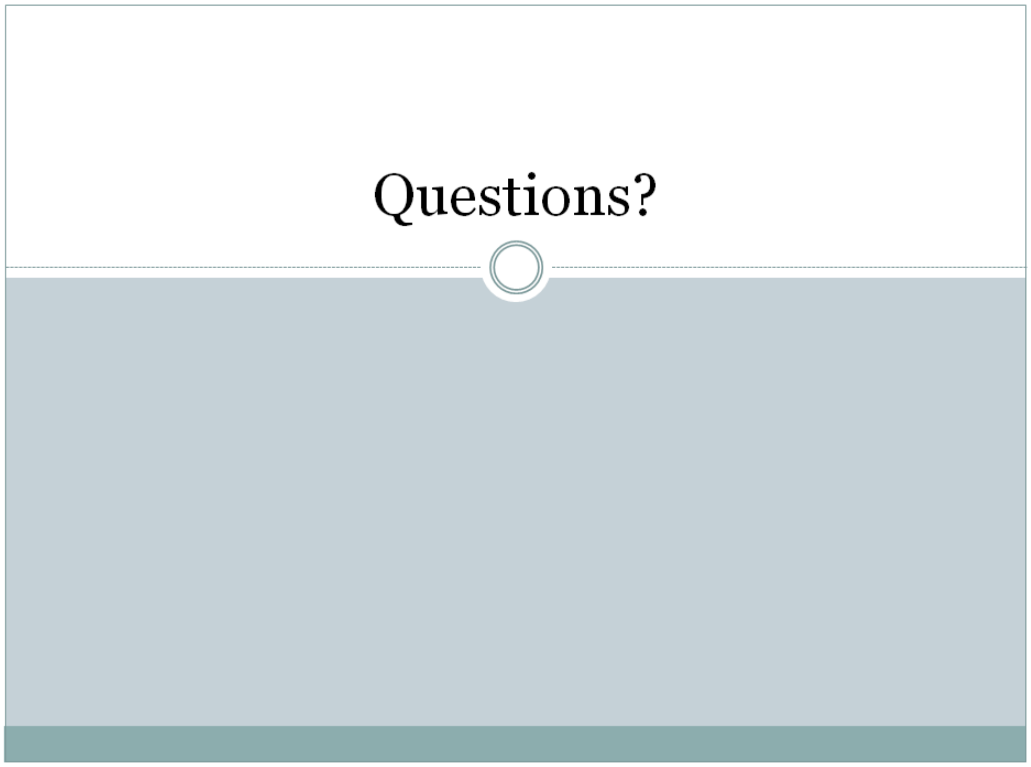
- Accomplishments
  - Met all client-provided specifications
  - Through analysis, detected potential complications, and satisfied the requirements to avoid them
  - Conducted on-site user testing to ensure usability as prescribed by the LSA
- What did we learn?
  - UI Development takes time, but is extremely rewarding
  - Properly evaluate your developer's tools before jumping in
    - Don't use a Beta!
  - The client often only has a general view of what they want
    - The developer is responsible for filling in the blanks



Future Work

14

- Expand the software to be an openly distributable application
  - Any network or distributed system
  - Operate from removable media
- Add algebra support
- Complete touch screen and SMART Board compatibility



Questions?

Annex B:

Addition of MTParser DLL:

In order to run the LSA Automated Math Testing Program, the included MTParserCOM.dll must be installed on the local machine. The most secure way of doing this is to copy the file to the desktop and attain its absolute path. Next, open a command prompt as an administrator and run the command *regsvr32* with the absolute path of the DLL as its one and only argument. This will allow the program to, without any specific reference to the .dll, run the included functions.

Running LSA.vb

In order to run the LSA Automated Math Testing Program from the source, you must have a Visual Basic compiler and editor, Microsoft Visual Studio 2008 is recommended. From within the editor, LSA.sln should be opened as a project. Once the project is opened the following options must be set in order to operate it properly. Under the file named "LSA" in the solution explorer, navigate via Compile -> Advanced Compile Options -> Target CPU, set this to x86 only. While still in "LSA" navigate to references. If "MTParserCOM 1.0 Type Library" is not listed, add it by clicking on add-> COM -> MTParserCOM 1.0 Type Library. This last step only works if you had added the MTParser DLL as described above. The program can now be run by clicking on the play arrow at the top of the screen.

Annex C:

Addition	
level 1	1d + 1d
level 2	1d + 1d with zeros
level 3	1d + 1d with carrying
level 4	2d + 1d
level 5	2d + 1d with zeros
level 6	2d + 1d with carrying
level 7	2d + 2d
level 8	2d + 2d with zeros
level 9	2d + 2d with carrying
level 10	2d + 2d with carrying > 1x
level 11	3d + 1d
level 12	3d + 1d with zeros
level 13	3d + 1d with carrying
level 14	3d + 2d
level 15	3d + 2d with zeros
level 16	3d + 2d with carrying
level 18	3d + 2d with carrying > 1x
level 19	3d + 3d
level 20	3d + 3d with zeros
level 21	3d + 3d with carrying
level 22	3d + 3d with carrying > 1x
level 23	4d + 3d
level 24	4d + 3d with zeros
level 25	4d + 3d with carrying
level 26	4d + 3d with carrying > 1x
level 27	4d + 4d with carrying > 1x

Subtraction	
level 1	1d - 1d
level 2	1d - 1d with zero
level 3	2d - 1d
level 4	2d - 1d with zero
level 5	2d - 1d with borrowing
level 6	2d - 2d
level 7	2d - 2d with zeros
level 8	2d - 2d with borrowing
level 9	2d - 2d with borrowing from zero
level 10	3d - 1d
level 11	3d - 1d with zeros
level 12	3d - 1d with borrowing
level 13	3d - 1d with borrowing from zero
level 14	3d - 2d
level 15	3d - 2d with zeros
level 16	3d - 2d with borrowing
level 18	3d - 2d with borrowing from zero
level 19	3d - 2d with borrowing > once
level 20	3d - 3d
level 21	3d - 3d with zeros
level 22	3d - 3d with borrowing
level 23	3d - 3d with borrowing from zero
level 24	3d - 3d with borrowing > once
level 25	4d - 3d
level 26	4d - 3d with zeros
level 27	4d - 3d with borrowing
level 28	4d - 3d with borrowing from zero
level 29	4d - 3d with borrowing > once
level 30	4d - 4d with borrowing > once

Multiplication	
level 1	1d x 1d
level 2	2d x 1d
level 3	2d x 1d with carrying
level 4	2d x 2d
level 5	2d x 2d with carrying
level 6	3d x 1d
level 7	3d x 1d with carrying
level 8	3d x 2d
level 9	3d x 2d with carrying
level 10	3d x 2d with carrying > once
level 11	3d x 3d
level 12	4d x 3d

Division	
level 1	1d / 1d
level 2	1d / 1d with remainder
level 3	2d / 1d
level 4	2d / 1d with remainder
level 5	2d / 1d with regrouping
level 6	2d / 1d with zero
level 7	3d / 1d
level 8	3d / 1d with remainder
level 9	3d / 1d with regrouping
level 10	3d / 1d with zero
level 11	4d / 1d
level 12	2d / 2d (easy numbers)
level 13	2d / 2d (ugly numbers)
level 14	3d / 2d (easy numbers)
level 15	3d / 2d (ugly numbers)
level 16	4d / 2d (easy numbers)
level 18	4d / 2d (ugly numbers)
level 19	4d / 3d

Fractions	
level 1	ID fraction in picture
level 2	solve fraction equivalent to 1/2
level 3	solve equivalent fractions
level 4	simplify
level 5	convert to greater terms
level 6	convert mixed numbers to fraction
level 7	convert improper fraction to mixed number
level 8	add like fractions
level 9	subtract like fractions
level 10	ID LCDs
level 11	add unlike fractions
level 12	subtract unlike fractions
level 13	add mixed numbers without renaming
level 14	subtract mixed numbers without renaming
level 15	add mixed numbers with renaming
level 16	subtract mixed numbers with renaming
level 17	multiply 2 fractions
level 18	reduce before multiplying
level 19	multiply fraction and whole number
level 20	multiply mixed numbers
level 21	make reciprocal fractions
level 22	divide 2 fractions
level 23	divide fractions and whole numbers
level 24	divide mixed numbers
level 25	convert fractions to decimals
level 26	convert fractions to percents
level 27	order fractions with like denominators
level 28	order fractions with unlike denominators

Decimals	
level 1	align decimal point
level 2	insert zeroes to the right of the decimal number
level 3	add without carrying
level 4	subtract without borrowing
level 5	add with carrying
level 6	subtract with borrowing
level 7	multiply with one decimal point in the bottom number.
level 8	multiply with one decimal point in the top number.
level 9	multiply with 2 decimal points
level 10	divide with decimal point in dividend
level 11	divide with decimal points in dividend and divisor
level 12	convert decimal to fraction
level 13	convert decimal to percent