Monique Dargis

TRANSITION

PROJECT VISION

This project will be a student test data storage system; specifically, it will be software used by tutors who specialize in teaching the ACT college entrance exam. A typical tutor may work with three hundred students over the course of the year, and each student typically takes at least three practice tests. So, the software will save time by generating scores from raw data that tutors enter. In addition, tutors will be able to quickly generate easy-to-understand score reports that they can share with students and parents. Tutors will also be able to keep notes on student progress to ensure that students are improving and to evaluate different teaching techniques. Overall, this is a system that will save tutors time and help them stay organized.

SCOPE

At this time, the software will not have the capability to store data for tests other than the ACT. The software should be used in conjunction with a scheduling tool such as Outlook Calendar or Google Calendar as it will not have the ability to remind tutors of future student appointments. The software will allow tutors to keep track of student and parent personal information, such as email addresses, but there will be no functionality for sending messages from within the system. Students will not be able to log in to the system to view their test data – instead, the tutor will need to send score reports to the students.

BUSINESS CASE

a) Total points: 23
   Cost: 23 * 4 hr/point = 92 hours.
   92 * $35 = $3,220 for estimated cost
   Tutors earn an average of $35 per hour after expenses, so this is the opportunity cost of devoting time to developing this software rather than spending time with students.

b) There is at least one existing system that allows tutors to enter test scores and generate reports for the ACT. Socrato[1] allows tutors to access banks of more than 20 ACT tests that they can administer to students. Tutors then upload raw student score data, the data is converted into an ACT score, and a score report is generated. However, the score report is not customizable and does not adhere to common human factors standards, with overly complicated graphs and a risk of information overload for students and parents trying to read the report. Furthermore, use of the software is quite expensive – a tutor would pay at least $5 per score report, which easily runs up to nearly $5,000 in expenses over the course of a year.

c) There are no open source projects which allow tutors to store test data and generate customized score reports.

d) The decision is to **build** the test storage data system. From an economic standpoint, the software will more than pay for itself in the first year of use. From a tutoring standpoint, the software will be superior to software that is already on the market because it will provide clearer score reports.

RISKS

I do not have any programming experience, so it will be a challenge to fully implement this software.

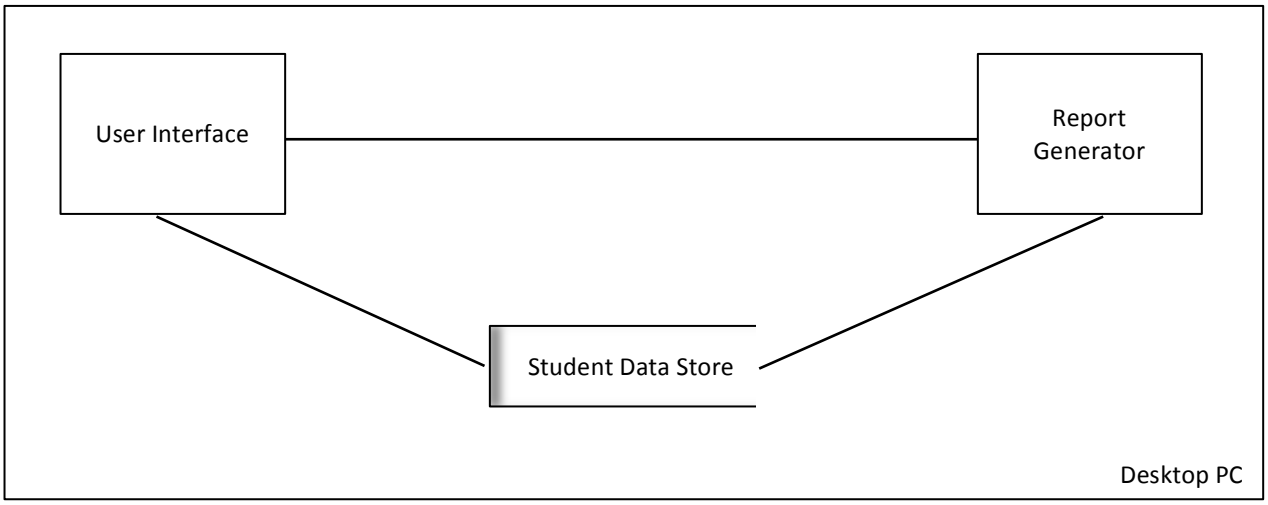
Integrity of data will be very important as even one error could significantly affect a student's test score and change the trajectory of the curriculum a test preparation tutor uses. The system will need to make it easy to enter test score data and easy to check that the raw data is correct before score reports are generated.

I will need to avoid making score reports too cluttered, while still conveying necessary information to students and parents. This is primarily a human factors issue, and although I have some experience in this area, I am certainly a beginner and will need to do a lot of research to select appropriate statistics and graphs to generate.

[1] http://socrato.com/socrato-for-tutors-and-schools/socrato-for-tutors/

CANDIDATE ARCHITECTURE

The system will run on a PC and student data will be stored somewhere that accepts data. The software will also generate a PDF score report that will not be saved in the system, so the tutor will need a place to store reports if there is a gap between the time a tutor generates a report and the time a tutor prints or sends the report via email



FURPS+

a) Features: All features designed by user stories and use cases.
b) Usability: The score reports should adhere to commonly-accepted best practices in human factors (ex. font should be readable, colors should be selected to avoid issues for those users with color-blindness)
c) Reliability: No special requirements at this time
d) Performance: No special requirements at this time
e) Supportability: No special requirements at this time
f) + No special requirements at this time

USER STORIES

- As a tutor, I would like to have student tests scored automatically so that I can save time I would have spent scoring tests by hand. (2 Points)
- As a tutor, I would like to be able to add new practice tests to the system so that I can offer these tests to future students. (4 Points)
- As a tutor, I would like to generate score reports so that I can share score information and progress with students and parents. (6 Points)
- As a tutor, I would like to store information about material covered and homework assigned during lessons so that I can plan future lessons more effectively. (2 Points)
- As a tutor, I would like to store personal information (ex address, phone number) about students and parents so that I can find this information easily when I need to communicate with students and parents. (1 Point)
- As a tutor, I would like to compare students to one another so that I can evaluate teaching techniques. (5 Points)
- As a tutor, I would like to generate statistics (ex mean score improvement) so that I can use this information in marketing materials. (3 Points)

GLOSSARY

- **ACT** – a college entrance exam given to high school students. The exam is made up of four required multiple-choice tests – English, Math, Reading, and Science – and one optional long-answer test – Writing. The four required tests have sub scores

out of 36; these scores are averaged to give a student's composite score out of 36. The exam is administered via paper and pencil. Results are a crucial part of a student's application to selective colleges, and poor scores can prevent an otherwise qualified applicant from gaining admission.

- **Composite score** – average of the four sub scores; reported as a value out of 36.
- **Outlook Calendar/Google Calendar** – software used, typically in conjunction with an email account, to schedule appointments and meetings.
- **Raw score** – the total number of correct answers in one of the four multiple-choice tests. Raw scores must be converted to a scaled score out of 36.
- **Score report** – for the ACT, a report that, at a minimum, contains a student's composite and sub scores for a single sitting of a test. More sophisticated reports may contain comparisons to previous tests that a student has taken and a question-by-question rundown of correct and incorrect answers and question types.
- **Sub score** – one of the four multiple choice test scores (English, Math, Reading, Science), converted from a raw score to a scaled score out of 36. Sub scores are averages to give a composite score.
- **Test date** – the ACT is given seven times per year, in September, October, December, February, April, June, and July.
- **Test form** – the version of the ACT given on a particular testing date. Each test form has a number/letter combination associated with it – ex. 72C.

Two Column Use Cases

User Case 1 – Add Student Test Results to System

A tutor will need to add a student's latest raw practice test data to the student's account via a scan of the student's answer sheet. The system will then calculate and store a test score, and ask the tutor if a student score report should be generated.

Actors: Tutor and System

Tutor	**System**	Questions to Answer	
1. Tutor logs in to system		tutor username, password	"login" scenario
	2. System displays tutor interface		
3. Tutor searches for student		partial or complete student name	"student search" scenario
	4. System returns possible matches for student		
5. Tutor selects correct student			
	6. System displays student profile page		
7. Tutor selects "add new test data"			
	8. System displays options for uploading student answer sheets		
9. Tutor selects method of uploading student answer sheet - scan		connect directly to scanner, or upload PDF from cellphone or desktop?	
	10. System connects to scanner		
	11. System displays scan options		
12. Tutor places answer sheet in scanner			"scan" scenario
13. Tutor clicks "scan"		ready to scan?	
	14. System scans answer sheet		
	15. System displays image of answer sheet		
16. Tutor accepts image of answer sheet		did answer sheet scan to system?	
	17. System displays stored test form options		
18. Tutor selects correct test form		which test form did student take?	
	18. System compares answer sheet to stored test form answers		
	19. System selects blank answers and answers for which system is less than 95% certain that the scan was read correctly		
	20. System displays possible errors to tutor		
21. Tutor confirms student answers by comparing possible errors to physical answer sheet		were blank answers really blank; are there any other errors in scan read?	
	22. System prompts for writing score		
23. Tutor enters writing score		writing score (calculated by tutor)	

	23. System calculates student's subtest and composite scores	
	24. System stores student's test score, raw test data, and copy of student answer sheet scan on student profile page	
	25. System displays student's test scores, including sub scores, writing score, and composite score	
	26. System asks tutor whether system should generate student score report	

USER CASE 2 – FIND STUDENT INFORMATION (WITH RAINY DAY SCENARIO)

A tutor will need to find a student's profile in the system. In this scenario, the tutor finds that the student has no record in the system and must add the student.

Actors: Tutor and System.

Tutor	**System**	Questions to Answer
1. extend "login"		
2. extend "student search"		
3. Tutor selects "add new student"		
	4. System displays new student form	
5. Tutor adds student profile information		student name/phone number/email/address; parent name/phone number/email/address, any available test scores, any general educational notes
	6. System displays new student profile	

USER CASE 3 – ADD NEW TEST FORM TO SYSTEM

A tutor will need to add a new test form to the system so that it can be used for future students.

Actors: Tutor and System.

Tutor	**System**	Questions to Answer
1. extend "login"		
2. Tutor selects "add new test form"		
	3. System displays add new test form interface	
4. Tutor names test form		test form name
	4. extend "scan"	
	5. System displays manual data entry interface	
6. Tutor manually enters answers to test form		answer for every data item
	7. System compares manual data entry answers to scanned test form answers	
	8. System selects answers that do not match	
	9. System displays possible errors to tutor	
10. Tutor confirms correct answers by comparing errors to physical answer sheet		which answer is correct?
	11. System prompts tutor to enter scale score conversion table	
12. Tutor enters scale score conversion table		for each scale score 1-36 for English, Mathematics, Reading, and Science, which raw scores correspond?
	13. System prompts tutor to confirm scale score conversion table data	
14. Tutor compares scale score conversion table data to original scale score conversion table and makes any adjustments		for each scale score 1-36 for English, Mathematics, Reading, and Science, does scale score conversion table data in system match original scale score conversion table?
	15. System displays all entered information about test form, including name, correct answers, and scale score conversion table	
16. Tutor confirms data is correct		are test form name, correct answers, and scale score conversion table correct?
17. Tutor saves test form		

PLAY

Setting: A software team (Proud, Best, and Gift) are trying to figure out how their student test data storage system will generate statistics on student progress for use by a tutor. In this fast-paced and engaged company, many other members of the development team walk by on a regular basis.

Proud: OK, so we've walked through how we're going to have tutors enter their student data. We have a rough idea of the steps to calculate a student's score. But what happens when a tutor wants to compare student progress?

Best: **Proud,** why would a tutor want to do that? Shouldn't she just worry about whether individual students are improving?

Gift: Well, if I could interject…

Proud: Go ahead, **Gift**.

Gift: **Best**, speaking as a former tutor, it is often hard to separate out the efficacy of a particular tutoring technique from an individual student's performance because different students have different levels of comfort with techniques and different work ethics. If I wanted to spend time teaching guessing techniques, for instance, it would be nice to be able to see whether that's a worthwhile technique for my students overall, even if each individual student doesn't necessarily benefit from that particular technique.

Best: Oh, that makes sense. You'd want to be able to kind of flatten out individual differences, then, to see whether the technique really works?

Gift: Exactly!

Proud: OK, so when tutors enter information on a particular student session, they are given a list of tags they can attach that have techniques they taught. Our system needs to pick up all of the students with a particular tag and show their average score improvement.

Gift: Well, we also need a control group to compare our students to, right? Otherwise, how can we tell whether the technique had any success?

Best: I think **Gift** is right. I would go even one step further – should we compare the time at which the technique was introduced? Maybe students who learn the technique earlier do better than students who learn the technique later….

Proud: **Best,** that's a great idea, but I think we're getting a little outside of the scope of our project. We'd need a statistician to help us with that programming! Let's stick to average score improvement by student, comparing between students who were taught a particular technique and students who weren't taught that technique. We'll use **Gift**'s example of teaching guessing techniques.

Best and **Gift:** Sounds good!

Proud: Let's start from the beginning. **Gift,** since you used to tutor, why don't you pretend to be our *user.*

Gift: I'm up for it!

Proud: **Best**, why don't you be the *interface* – the place where the tutor gets to send requests and receive information back?

Best: I think I can handle that! What about you, **Proud**?

Proud: I'm going to act as *facilitator* – I'll make sure everyone's messages get to the right place at the right time. OK, *user*, let's get started!

User: Well, first step would be to access the system, right? Can you help me with that, *facilitator?*

Facilitator: *Interface*, that's you. Please display the login page for our user.

Interface: OK, displaying fields for username and password.

User: I'm just going enter my username and password…OK, I put my info in and pressed 'enter.'

Facilitator: We need someone to check your credentials. Hey **Boom**, will you be our *security*?

Boom: Sounds fun! What should I check for?

Facilitator: Make sure this *user*'s username and password match the information we have stored in the system.

Security: Computing…computing…DOES NOT COMPUTE!

Facilitator: **Boom!** Cut it out! We're trying to take a *successful* path through the system. How can we do that if our user can't log in?

Security: Oh, fine, I'll let her in…I guess the username and password checks out, *facilitator*.

Facilitator: Great! *Interface*, can you display the tutor interface?

Interface: Displaying tutor interface. Do you see the option for statistics?

User: Yeah, it's right there! I'm going to click on that.

Facilitator: Make sure that she tells you whether she wants individual student statistics or population statistics, *interface*.

Interface: Did you hear that, *user*? Which type of statistics do you want?

User: Um, we're comparing across students, so…population statistics!

Interface: Got that, *facilitator*?

Facilitator: Yup, I'll send that information along to…hey, **Ice**, will you help us out? We need someone to act as our *retriever* – the person in the system who calls up the student data and has it ready for calculations.

Ice: Sure, happy to help out! Do you need a *calculator*, too? My brother **Float** is great at statistics.

Float: Don't drag me in to this!

Ice: Come on, **Float**, we can't do it without you…

Float: Well, if you really need me…..

Facilitator: OK, so **Ice** will be our *retriever* and **Float** will be our *calculator*. *Retriever*, will you pull up the student data? I need to tell our *interface* which tags to display. At this stage, can you just give me all of the possible tags?

Retriever: Here's a list of all of the tags associated with the student data, *Facilitator*.

Facilitator: *Interface*, please display all of these as options for the *user*. Let's see which one she wants to use in our statistical analysis.

User: OK, I can see that the *interface* is displaying all of the tags I've used for students. I'm going to select the tag labeled "guessing techniques."

Facilitator: Did you get that, *interface*? We've got to move on to the next part of the analysis – selecting which statistic to generate.

Interface: Which options do you want me to give her?

Facilitator: How about mean, standard deviation, median, mode, range – your typical arithmetically-generated statistics.

Interface: OK, displaying those options for the *user*.

User: Well, I want mean…but how do I make sure I'm getting the mean of the composite score and not one of the sub scores?

Facilitator: Oh, good point! We should probably have you select that option before we pick the exact statistic…let's back up to the moment after the *user* selected the "guessing techniques" tag. *Interface*, this time please display five options – composite score, English score, Math score, Reading score, and Science score.

Interface: Displaying the different test score options for the user.

User: Cool! I'm going to pick "composite score."

Facilitator: Great. We already figured out the selection of the exact statistic to generate, so let's just include that here and not belabor the point…the last thing that we need our *user* to tell us is exactly how to split up the data. We want to compare average score improvement of students who were taught the technique to the average score improvement of those students who weren't taught the technique.

User: Could we just make the system compute that by default whenever we are dealing with population statistics and tags? That comparison is going to be most meaningful for a tutor in most situations…we could set it up so that's the default, and the tutor has to choose to compute something different if they want something else.

Facilitator: That makes sense to me. Any objections? *((no one speaks…))* Great. I think we're ready to roll! OK, a lot of things have to happen at once, so listen carefully. *Interface*, will you send the *user*'s selections to the *calculator?*

Interface: *Calculator*, we want to compute population statistics – mean improvement of composite score, based on whether students are tagged with the "guessing technique" tag. Do the default scenario – compare mean improvement of students with the tag to mean improvement of students without the tag.

Calculator: Well, I'd love to, but…

Ice: **Float**, are you serious? You can*not* back out now!

Calculator: Well, I don't see how I'm supposed to calculate statistics when you haven't even sent me any student data!

Facilitator: Ugh! My bad! *Retriever*, you've still got all that student data, right? Please send it over to your brother.

Retriever: Here you go, *Calculator* – all ready for you to work your math mojo.

Calculator: Computing means…got it, *Facilitator*.

Facilitator: Are you ready to wow our *user*, *interface*?

Interface: Give me those means!

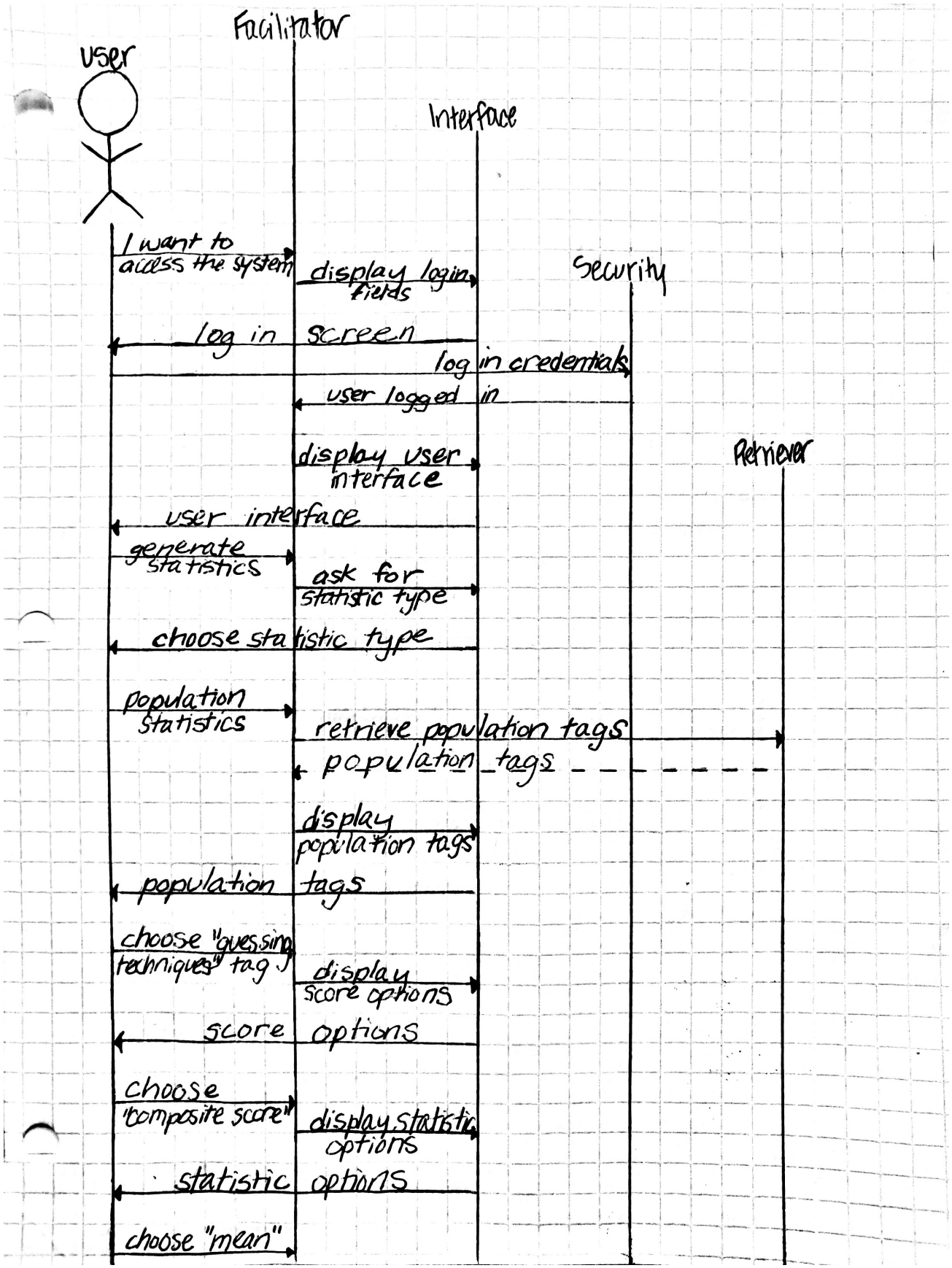
Facilitator: Here you go. What do you think, *user*?

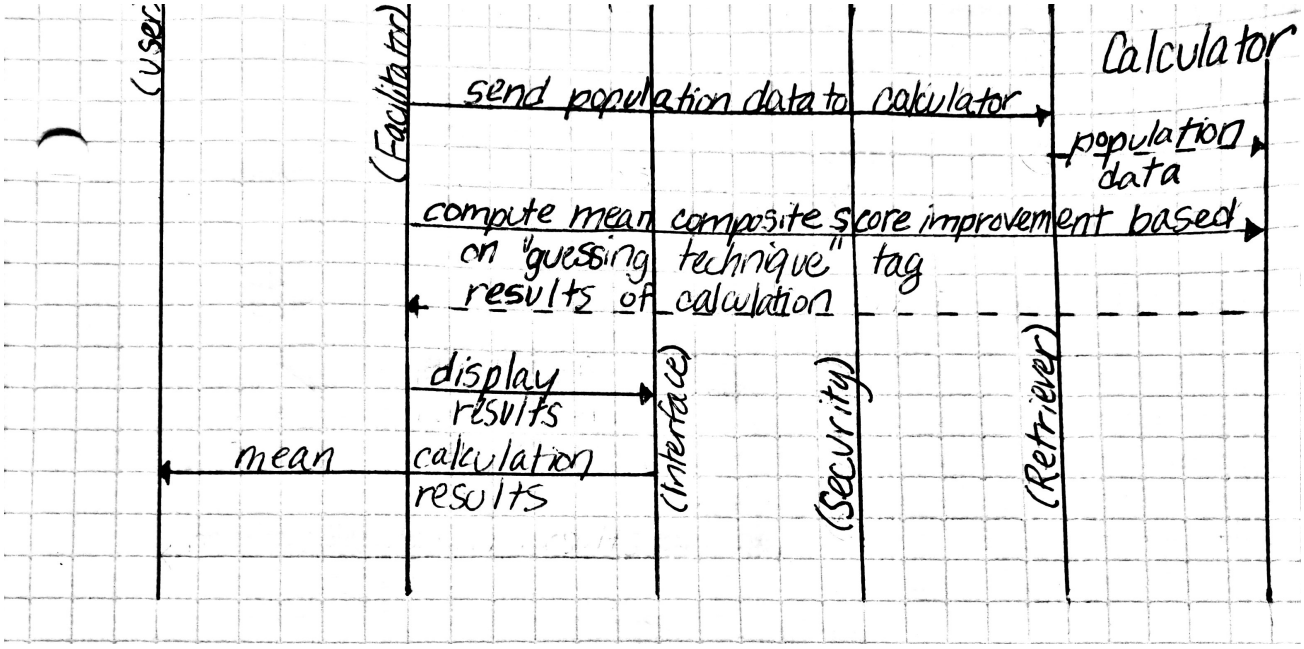
User: Wow, it's so clear! I can see right here on the interface that the students who were taught the technique had a higher average score improvement than student who weren't taught the technique. I guess there's some evidence that the technique is, overall, a good one to teach!

Proud: There you have it, folks! That's exactly how our system will work.

--fin--

Sequence Diagram





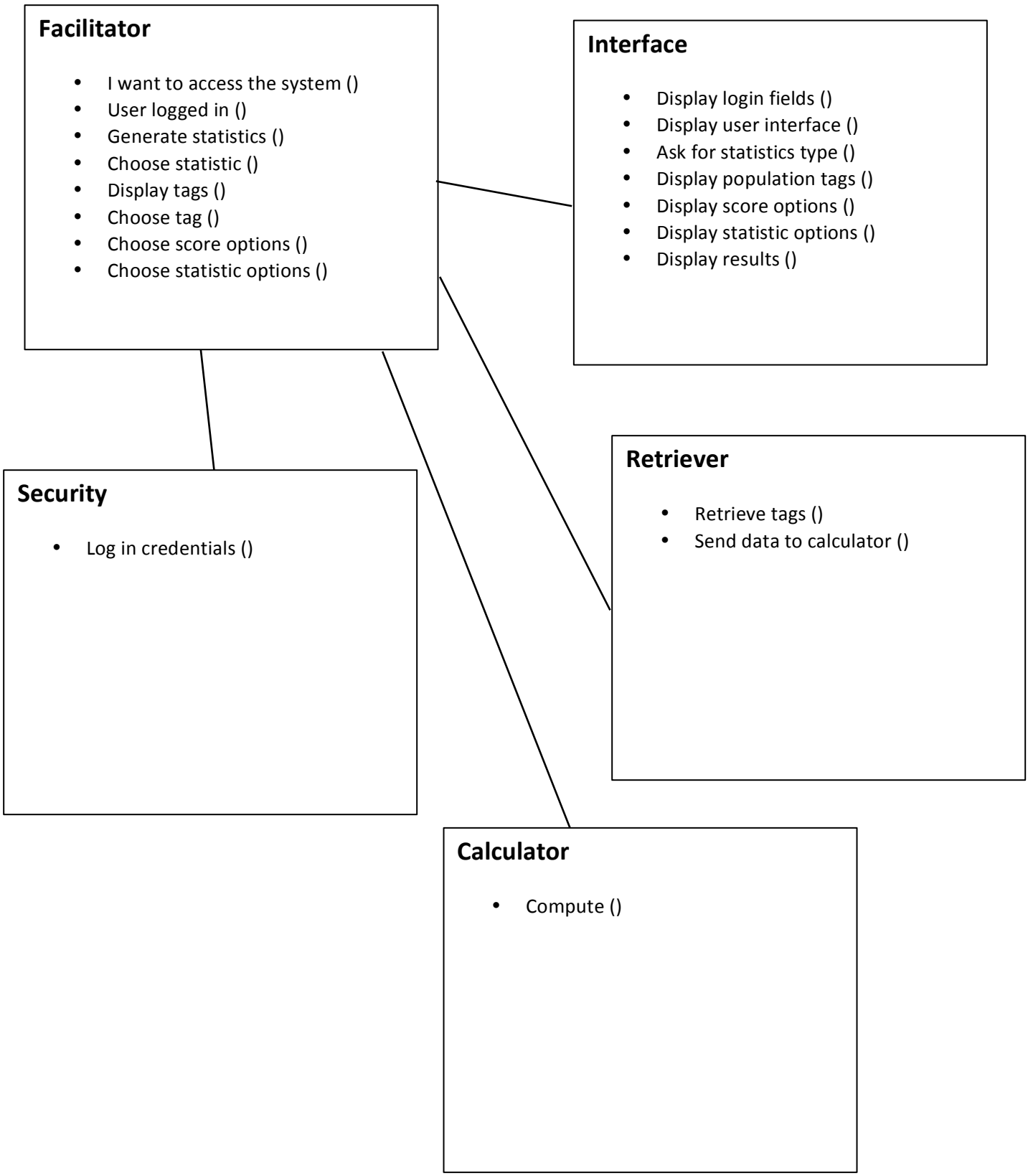
ALARM LEVELS

Notable, Financial and Emotional Trouble

The software may malfunction in several ways, all of which could have financial and/or emotional consequences for tutor and students. Probably the most serious failure of the software would occur if it fails to report a student's practice test score correctly. If the score is incorrectly reported as being too low, there could be an emotional impact on the student, as it can be very disheartening to work hard but still not see a score improvement. If, on the other hand, a student's score is incorrectly reported as being too high, the student may have a false sense of confidence, leading them to prepare inadequately for the real test. Incorrect score reporting could also influence the tutor to change the student's individualized curriculum in a way that actually isn't warranted by the student's true score. Further, if the student or a parent later discovers that the student's score was reported incorrectly, it could make the tutor look unprofessional and affect the tutor's ability to get further work as most tutors rely on referrals from previous students.

The other major way that the software could fail is if it is not able to generate accurate statistics based on tags within the student population. The primary use of the statistics is to help a tutor choose a student's individualized curriculum, and incorrect information could lead to the tutor making less-than-optimal curriculum decisions. If this were to happen, the student may not reach their full potential in terms of test scores. This could have, again, emotional repercussions for the student as many students feel their self-worth is at least partly wrapped up in their test scores. More importantly, though, failing to achieve a high test score could prevent the student from gaining admission to the university of their choice or qualifying for academic scholarships, which could have both emotional and financial consequences for the student's future. Also, the student may be less likely to refer other students to the tutor, which could again affect the tutor's bottom line.

CLASS DIAGRAM



TEST PLANNING ESSAY

This product has been designed using the Rational Unified Process, but I think that rapid prototyping (adding in the timeboxing technique from the agile approach) is the most appropriate life-cycle model. The developer and user of the product are one and the same, so understanding the requirements should be relatively simple. The developer/user should be able to move into actual coding fairly quickly, and the rapid prototyping technique would allow the software to be used in the field within weeks. This would be a benefit to the software engineering process as the user could engage in black box testing on a regular basis using current student answer sheets. The tutor would also be able to elicit feedback from students and parents on the presentation of the score report. Every three or so weeks, the tutor would release a new feature or version of the software until it is perfected. The students would be able to see improvements to score reports in real time. However, the tutor would have to be careful during this time to check and double-check the accuracy of the scores reported before sharing them with a student.

As detailed in the two-column use cases, the system will have some built-in measures to ensure that the data fed into the system by the tutor is accurate. The tutor will be required to manually enter and scan answers to new test forms, and any answers on scanned answer sheets that are less than 95% certain are flagged by the system and the tutor is required to manually verify that the data is correct. However, there should also be extensive testing done before the product is delivered in its final form. At a minimum, the software will need "95% certainty" defined, and the easiest way to do this is by using real student answer sheets to calibrate the system. The user should have access to many answer sheets and accurate scores from former and current students who have taken hand-graded practice tests.

In an ideal world, there should also be a fair amount of code review taking place; however, this would be difficult with a one-person team as it is not a good practice to be the only person reviewing your own work. It may be that, if the software expands beyond a single developer/user, an open-source life cycle approach should be taken to crowd-source fault detection. I do not think correctness proofing would be necessary because failure of the software is not life-threatening. It would also be difficult to not only convince open-source user to submit correctness proofs but also to find someone skilled enough to check their logic.