GFE Example: VB for Non-CS Students

J. Philip East, philip.east@uni.edu
(Developed for a GFE workshop with Andy Bern & Ben Schafer)

This document describes a possible[1], partial[2] sample of applying Grading For Equity (GFE) principles to a course for non-majors in Visual Basic Programming. The general goal of the course was to provide students sufficient programming capability to write a program to solve a problem from their personal lives/contexts if they chose to (and, thus, to make it more likely they might choose to). We[3] determined that our 3-credit course should focus on problems rather than the language features course often taught to CS majors. Please keep in mind that this is not meant to be prescriptive and that there are almost certainly alternatives embodying GFE that would work as well or better than those identified.

Course Outcomes

We believe outcomes are capabilities, not knowledge. Knowledge alone is trivia--it becomes useful or important only when it is put to use. Knowledge is a means to an end, not an end in itself. Outcomes need to be stated in terms of what the learner will be able to do in relation to a course related task. For this course those are programming related tasks.

We worked backwards from the general course goal to develop the more specific course outcomes. We decided that developing student capabilities for that goal would require a course project after students had learned to use the programming IDE and the basics of programming. We also thought that many problems students might encounter would involve files. The general course outcome was that

> Students should be able to produce a program whose solution required the use of the basics of programming, data collections, and files.

We had the perspective that those basics required knowing the actions computers can carry out and ways the actions can be organized into programs. Additionally, we thought it important to present all coding activities in a problem-centric context. The course outcomes are provided below.

- **Programming Basics**
  - **System Utilization.** Use the Visual Basic IDE to create and test a program and submit the program to the instructor.
  - **Basic Actions.** Given the description of an IPO (input, process, output) problem, develop, test, and submit a program solution for various types of data

[1] The general structure described here is from an actual course but it was taught before GFE was discovered. The description is, therefore, a modification of what was actually done. The grading scheme is also “based on a true story” but is a actual reflection of what was done..
[2] Most of the desired course outcomes are included but only a subset of assessments and learning activities are provided in an effort to illustrate possibilities,
[3] Stephen Hughes and J. Philip East. The original course grew out of Stephen’s idea of presenting a set of problems on the first day of class and saying that by the end of the course, students should be able to solve any of them. See http://www.cs.uni.edu/~east/PcPl/problemList_2.0.php for more information.

(numeric, string & control property) of various kinds (constants, literals, variables & common functions).
Several components of this outcome were identified.

- **Boolean Expressions.** Given a description of the need for a Boolean expression, provide a Boolean expression to achieve the goal.
  We felt that numeric expressions were familiar to students and that string expressions were a relatively natural extension. However, we also felt Boolean expressions were less familiar and they required knowledge of numeric and string expressions. Thus, we had a separate unit/outcome involving conditionals.
- **Selection.** Provide two or more alternatives for selection (*if*) statements to achieve a desired result.
  Providing alternative codings is thought to deepen student understanding of the relationship between the conditional expression and the control structure of the selection statement and how simpler conditionals can be used with more complicated control structures and vice versa. It should also enhance student problem solving.
- **Repetition.** Provide two or more alternative solutions for problems requiring repetition.

- **Putting It All Together.** Write a short program (one page or so) for a task using data (a data collection to or from file), actions (input, assignment, built-in functions & output), and control structures (sequence, selection & repetition).
- **Project.** Identify a problem requiring all key course elements (sequence, selection, repetition, modularization, and file usage) then plan, develop, implement, and evaluate a program to solve the problem.

Learning Activities & Assessments

Identifying course outcomes and developing learning activities and assessments is not a linear process. Outcomes **must** be assessed and some outcome candidates are not amenable to assessment as worded. Learning activity (lecture, demonstration, homework, etc.) is needed but sometimes it is difficult to imagine how some outcomes (as worded) might be taught and practiced. There was much revisiting of the statements of outcomes, assessments for them, and the planned learning activity--the outcomes above and assessments and learning activities below did not initially have the form they do now.

Programming Basics

We wanted to provide a great deal of practice activity to develop the desired capability. We also wanted to use something akin to mastery learning in grading but we didn't like the term "mastery" feeling it implied a deeper capability than we were expecting. We settled on the term "competency". We had begun using two of the GFE practices prior to encountering GFE--not "grading" homework and allowing retakes of assessments. For each unit we would perform

in-class demonstrations, provide large homework sets with answer keys, and use additional class time for questions and feedback. Assessment would be in the form of competency demonstrations, essentially 20-minute quizzes, that could be graded quickly. Each competency demo had three possible marks--competency demonstrated, competency not demonstrated, and uncertain. For the uncertain cases, we asked students to come meet with us to discuss their performance.

The competency demos had several parts. Some addressed component elements of the capability and an overall capability. Briefy, they were:

- **System Utilization.**
- **Basic Actions.**
    - Indicate outcome of expressions (various data types & kinds).
    - Produce expressions for indicated tasks (various data types & kinds).
    - Trace code for a small IPO problem identifying the resulting value(s).
    - Identify the general purpose of code for a small IPO problem.
    - Produce code to solve a small IPO problem (various data types & kinds)
- **Boolean expressions.**
    - Indicate value of given Boolean expressions for indicated values for variables.
    - Produce assignments statements to save the value of a Boolean expression.
    - Generate Boolean expressions for given tasks in varying contexts (assignment, output, in *if* statements
    - Describe the general outcome or result from a given Boolean expression (e.g., always true, always false, indicates a grade of B, etc.)
- **Selection.**
    - Indicate the result of (perhaps nested) if-then-else statements assuming indicated values for variables involved
    - Given selection statements to accomplish a given task, provide an alternative that achieves the same result--compound Boolean conditionals versus nested if-then-else statements (e.g., leap-year determination), independent if statements versus case-like if statements (e.g., grade determination), etc.
    - Produce if statements involving various types and kinds of data to accomplish various kinds of tasks (set variables or properties, report, set Boolean flags, etc.)
- **Repetition.**
    - Trace repetitive code using either counting or indefinite loops to indicate the result of execution.
    - General understanding, e.g., Discuss similarities and differences of counting and indefinite loops; Choose between counting and indefinite loops for a given task and explain the choice; Convert between counting and indefinite loops.
    - Provide code to accomplish a repetitive task

Sample competency demos can be seen at
http://www.cs.uni.edu/~east/GFE/sampleCompetencyDemos_VB/.

The overall grade for the competency demos would be one of five categories--woefully inadequate, marginal, okay, good, or excellent (I, M O, G, E).

Project

The project is worked on over a number of weeks, perhaps a third or more of the course. Some class time is used for additional programming feature discussion. During the last two or three weeks of the course, much class time would be spent working on the projects. The project serves multiple instructional purposes. It would provide students with the experience of identifying a problem of their own for which a program would be useful, hopefully enhancing the possibility that they would actually apply programming to problems in the future. We would use the project activity to show the process for program planning and development. The programming would provide a learning activity for putting all the programming pieces together to write a relatively simple program enhancing student confidence in their ability. Initially, the projects were done by individual students but in later courses they could be done individually in groups. Assessment in those cases would need to differ to be true to GFE.

The instructional activity for projects could be class discussion of the assignment--its goals and the process--and of exemplars from previous classes (or the instructor) and eventually class-time work on the projects. Various project submissions would be discussed in person with each student (or group) to provide feedback about the project, e.g., too involved/large, too small/easy, recommended modifications, suggestions about programming element use, etc. If desired individual students could resubmit for a second assessment and in some cases a third. Anticipated project submission elements and assessment criteria follow.

- **Project Identification and Planning**
  Each student or group would produce and submit a plan consisting of the elements described below. For individuals working in a group each would produce a draft of the plan that the group would consider and use in developing the group plan.
    - **Identify the project and its general goals/behavior**. Was sufficient detail provided to give an accurate understanding of what would happen?.
    - **Indicate how the project will incorporate expected programming features**. Were selection, repetition, modularization, and file usage all addressed and their use as described reasonable?
    - **Provide a general algorithm for the top level (or two) of the program.** Was the algorithm readily understandable? Was it complete? Was it close to being able to accomplish the described goals/behavior?

  Each element of the submission would be assessed using the 5-category scale identified above--I, M O, G, E. Care would be taken to assess ideas rather than their presentation as writing is not a course goal. An overall mark would be based on the three individual marks:
    - **E**. Three Es,
    - **G**. All Gs and Es; or one each of O, G & E.
    - **O**. All Os, Gs & Es; or at most one M with an offsetting G or E.
    - **M**. All Ms; or at most one I with an offsetting O or above.
    - **I**. Does not meet any criteria above.

  For individuals working in groups, group products would be assessed but not count in individual grades. Individuals would receive a draft on their draft of the plan. Individuals

and groups would meet with the instructor for project plan feedback. Submitted plans could be revised by individuals as a retake..

- **Project Report**
  Individual students will submit a report evaluating various aspects of the completed project as indicated below. The report can be revised and submitted for a retake.
    - **Project Assessment**. A self-assessment of the first three elements in "Project Implementation" above identifying any deficiencies.
    - **Project Alternatives**. Ideas for changes in the project, i.e., additional functionality, changes in existing functionality, changes in program implementation (code), etc.
    - **Personal Understanding**. An identification of the various functional parts of the submitted program and a self-assessment of student capability/understanding with respect to them. (Groups members indicate the parts they did or worked on.)

  The project implementation would be assessed (as discussed below) and used to inform this assessment. Each element would be assessed somewhat differently but using the course standard 5-category scale--I, M O, G, E They would be examined for completeness and appropriateness, e.g., alternatives are reasonable, assessment matches reality, etc. Care would be taken to assess ideas rather than their presentation as writing is not a course goal. An overall grade would be based on the three individual marks:
    - **E**. Three Es or two Es and one G.
    - **G**. All Gs and Es; or one each of O, G & E.
    - **O**. All Os, Gs & Es and at most one M with an offsetting G or E.
    - **M**. All Ms or above with at most one I.
    - **I**. Does not meet any criteria above.

- **Project Implementation and Student Understanding**
  Project code and a *readme* file indicating how to use it would be submitted and an appointment for each individual student scheduled with the instructor. Prior to the appointment, the program would be assessed and a plan for assessing student understanding of the code made. When meeting with the instructor the student would discuss/explain the code and respond to questions from the instructor. There is no opportunity for a retake on the project. The grading elements would be:
    - **Programming feature use**. Does the program use all the expected programming elements? Are they used appropriately? Are there few or no instances of features not used when they should have been (and instruction was timely and adequate)?
    - **Program correctness**. Did the program run as described in the *readme* file? Does it run without crashing? Does the program produce correct results or behavior?
    - **Requirements met**. Does the program do what was originally planned (or as revised)?

    - **Code understanding**. Can the student explain selected elements of the program and discuss choices and alternatives?

  Each element would be assessed as to how well it met expectations. The student would respond to questions from the instructor as to program operation. The course standard 5-category scale would be used to indicate the degree to which expectations were met. The code understanding is considered more important that other elements as indicated by the overall mark based on the five individual marks:
    - **E**. An E on code understanding with the rest Es and Gs.
    - **G**. An E or G on code understanding with the rest Gs or Es with at most one O.
    - **O**. An O or above on code understanding with at most one M offset by a G or above, no mark of I).
    - **M**. All Ms or above and at most one I.
    - **I**. Does not meet any criteria above.

The overall project grade would consider the three project product assessments equally with result shown below:

- **E**. All Es & Gs with more Es than Gs.
- **G**. All Gs or Es with at most one O.
- **O**. All Os or above except for one M.
- **M**. All Ms or above and at most one I.
- **I**. Does not meet any criteria above.

Putting It All Together

This assessment is essentially a final exam. In addition to assessing the students it is used as the primary course assessment instrument. Due to time constraints there is no opportunity for a retake. The final has two parts. The first is the 10-item quiz adapted from the benchmark exam of Simon et al[4]. The second is the coding of a relatively short program. that requires: selection, repetition, file usage, and an indexed data collection.

Benchmark

The benchmark exam has 10 questions. We divided them into three groups--four multiple choice (MC), three code tracing (CT), and three supply code (SC). Each would be marked as correct or incorrect. Using the course standard 5-category scale resulting grades would be on the number of items in each group that were correct, i.e.,:

- **E**. Three SC items & five of the other seven, or two SC items & six of the other seven.
- **G**. Two SC items and at least five of the other seven, or one SC item correct and all the other seven.
- **O**. At least one SC item and four of the other seven, or no SC item and all the other seven.

[4] Simon, Sheard, J., D'Souza, D, Klemperer, P., Porter, L. Sorva, J., Stegeman, M. & Zingaro, D. Benchmarking Introductory Programming Exams: Some Preliminary Results. *ICER'16*, September 8-12, 2016, Melbourne, Vic, Australia.

- **M**. At least four of the seven non-SC items.
- **I**. Does not meet any criteria above.

Programming Problem

The relatively short program would require: selection, repetition, file usage, and the use of an indexed data collection. Resulting code is assessed for semantics, not syntax. The general expectations are:

- File-related
    - File is properly opened and closed
    - Code for reading from and writing to the file is located appropriately (e.g., within repetition)
- Selection
    - Correct Boolean expression used
    - Appropriate selection structure used (if-then vs if-then-else vs ...)
    - **Appropriate** action(s) taken for each outcome
- Repetition
    - Correct continuation/halting condition
    - **Appropriate** actions repeated
- Sequencing actions
    - Appropriate actions taken before and after repetition
    - Correct sequence of actions performed within repetition--before, inside all blocks, and after selection
- Correct non-trivial use of an index into the collection

Each aspect of the code would be assessed as one of three possibilities--correct, close, or marginal. A 5-category grade for the overall problem would be produced as follows:

- **E**. At least four correct and the other close.
- **G**. At least three correct and the other two close.
- **O**. At least one correct and three of the others close.
- **M**. At least three close.
- **I**. Does not meet any criteria above.

Overall Grade For The Final

The grade on the putting-it-all-together assessment would combine the two parts as follows:

- **E**. Both Es.
- **G**. Both Es or Gs.
- **O**. Both Os or above.
- **M**. Both Ms or above; or one G or above and an I.
- **I**. Does not meet any criteria above.

Overall Grading Scheme

The overall grade would be determined by combining the three elements--programming basics, project, and putting it all together--as follows:

- **E**. All Es; or two Es & a G.
- **G**. All Gs or above; or E, G, & O.
- **O**. All Os or above; or at most one M offset by a G or E.
- **M**. All Ms or above, or at most one I offset by a G or E.
- **I**. Does not meet any criteria above.

The overall mark would translate directly into the A, B, C, D, F approach. Note that averaging is not used. One result of this is that an inadequate or "failing" mark will prohibit a final grade of C (that we think of as "okay" work).

Comments and Caveats

We believe this document provides an example of what might result from working to convert course from traditional grading practices to one that applies the tenets of Grading For Equity. Please note that the above suggestions have never been used in an actual class. They should be considered a possible starting point. They are meant to be descriptive of something that might be used, **not** prescriptive of anything.

Questions, comments, and discussion can be sent to Philip East (philip.east@uni.edu).