Does the sum of the parts equal the whole?

Errol Thompson
Massey University
Wellington
E.L.Thompson@massey.ac.nz

In assessing software development projects, we want to assess a range of characteristics of the finished product. These include the design, the code, and possibly of the process used. How does providing a grade for each of the desired characteristics compare with a setting a minimal standard for each characteristic? The holistic assessment strategy is discussed and compared to grading of the parts, each characteristic, independently. In doing so, we contend that the assessing by summing the parts is not equal to assessing the whole.

Keywords
assessment, criteria.

1. INTRODUCTION

The assessment of programming assignments focuses on the characteristics of the submitted program code and supporting documentation. What is not assessed is the process used to achieve the submitted result. It is assumed that a higher coding standard and documentation standard means a better process and cognitive skill set is being used.

When using an assessment of the parts or characteristics, it was observed that students were focussing on some criteria ahead of others. An often overlooked characteristic was the testing plan or strategy. The ignoring of this characteristic was often an indication that students had not carried out any formal testing. The failure to use a formal testing strategy was often indicated by the program failing when some forms of invalid data was submitted.

However, the general standard of coding and the level of functionality implemented often saw these students obtain a pass grade. From an assessment perspective, this was considered unacceptable. To illustrate the situation consider the following cases.

2. CASE DESCRIPTION

2.1 Charles, a confident programmer

Charles believes that he is a good programmer. He reads the specifications and sets about the coding process. He focuses on implementing the required functionality according to the specification and when he is uncertain of what the specification is saying, he makes his own decision without consultation with the customer (lecturer). He aims for and believes that he has close to 100% coverage of the required functionality.

After writing large chunks of functionality, he manually tested his code with primarily valid values. The occasional invalid values are used more by mistake than by intention. He documents some of his tests by listing the possible input values and expected outputs. All are for valid values. Many of these recorded tests are incomplete because he records them before submission of the assessment. The testing strategy is incomplete, has about 50% code coverage, and ignores the testing of portions of the required functionality. If asked, he says that those portions of code were so simple, he knew they would work.

The code quality closely follows coding standards. Internal and external documentation is minimal. Each piece of functionality is implemented independently of previous functionality with only the more obvious common code shared through the use of subroutines and functions. Where a piece of functionality involved interaction with the user, a form has been implemented for it with a menu form providing the linkage between forms. Minimal thought has been given to usability issues or the way that the user interacts with the system.

During assessment, the program is assessed as implementing 90% of the functionality but only 70% of the validation of input data. There is some use of procedures and functions in the code but still some degree of code duplication. The code quality is assessed at 70% and testing strategy at 40%. User interaction design is assessed at 20% primarily because each form behaves well although there are tab sequence problems. Quality of documentation is assessed at 20% and there is no reasoning for the design approach provided so this is assessed at 0%.

2.2 Cheryl, a less confident but meticulous programmer

Cheryl is not confident about her coding ability. She examines the specification and divides the requirements into similar sized chunks. She prioritises the chunks based on the dependencies in the requirements and after seeking clarification from the customer (lecturer). For the highest priority chunk, she carefully prepares test data and documents each test listing the input and the expected outputs. She records, for later documentation, design decisions along with the reasoning for the decision made.

After deciding on an architectural and user interaction design approach, she prepares an initial model using UML. She retains the UML diagram for inclusion either in her documentation or in her diary of the process that she has followed. Following the instruction from the master craftsperson (lecturer), she prepares a unit test strategy for the key class in her design based on her previously developed test data. She then codes the first test using a unit-testing framework. After writing the minimum code to pass the test, she then codes the next test and continues until she has implemented all the functionality for her unit test plan. She then moves to the next class in her design and continues the incremental coding process.

As she codes, she recognises common code portions and after ensuring that the currently implemented unit tests are passing, refactors her code to remove duplication and improve the code structure.

As she completes each chunk, she ensures that she runs her initial test plan as an acceptance test strategy and completes the documentation of the chunk including updating the related UML diagrams. At all stages in the coding, she is prepared to revise the previously completed code with the confidence that her automated tests will reveal any problems.

As she progresses her coding rate increases but she recognises that she will be struggling to implement enough functionality. She discusses this with the lecturer who acts as the customer for the project. The lecturer agrees to some changes in scope and she revises her plan for the next chunk.

Her process is slow but the quality of her code is high. As she comes to the due date, she realises that she has completed only 60% of the original functionality but what has been completed 75% of the revised functionality agreed to by the customer (lecturer).

During assessment, the assessor assigns a grade of 75% for functionality based on the revised requirements implemented. The data validation is assessed as 90% completed. The quality is assessed as close to 95% for the completed code and all completed code has automated unit and acceptance tests implemented. Where Cheryl was unable to implement automated tests, a testing strategy is provided in her documentation. Should the testing assessment be 100% or 70%? Her documentation is of a high standard. It includes design diagrams and the reasoning for her design decisions. It is assessed at 95%.

Because of her emphasis on seeking clarification and feedback from the customer, her user interaction design is of a high standard. For the functionality implemented, her interaction design is assessed as 90% but is that 90% of 70%?

3. ASSESSMENT STRATEGIES

Two assessment strategies are applied to these cases. The first assigns a portion of the grade to each criterion based on the importance of and / or the amount of effort required to complete the criteria (assessment by parts, see Appendix A). This could be considered as a traditional assessment strategy. The second strategy assigns a set of criteria for grade bands (holistic assessment, see Appendix B). A grade band incorporates a required minimum level for each of the criterion.

The grade band approach was derived from a holistic assessment approach based on the SOLO levels (Biggs, 1999; Biggs and Collis, 1982). The

SOLO levels focus on cognitive skill levels. These levels are:

- unistructural – focuses on one conceptual issue or only one feature is given serious consideration,
- multistructural – a range of conceptual issues or features are considered but not related
- relational – conceptual issues and features are being integrated and applied in meaningful ways, and
- extended abstract – relating conceptual issues and features to existing principles and possibly questioning existing principles.

The holistic assessment scale has endeavoured to map these levels onto what might be expected from a programming assignment. Some revision is still necessary and probably desirable. However, there is the question of hoe the higher level cognitive skills are presented in program code. This may be easier where more complex object-oriented thinking patterns are required for the assignment. This aspect is still open for exploration in the assessment strategy.

At what level are our two students assessed using the assessment by parts verses the holistic assessment strategy?

Using the assessment of the parts strategy, Charles scores 18 for functionality, 14 for data validation, 10.5 for code quality, 6 for testing strategy, 2 for user interaction design, 2 for documentation, and 0 for design reasoning. This gives Charles an overall grade of 52.5% (C+). Charles has passed.

Using the holistic assessment strategy, Charles is judged to fall within the 40-49% scale as he has not implemented any automated testing and has a low level of documentation. Because of the high percentage of functionality and code quality, he is given a 47% (D) grade for the assignment.

Cheryl's grade is more difficult to assess under the assessment of the parts strategy. The assessor decides to give her 15 for functionality, 18 for validation, 14.5 for the quality of her code, 14.5 for the testing strategy, 6.5 for interaction design, 9.5 for documentation, and 7 for design reasoning. This gives Cheryl an overall grade of 85% (A+).

For the holistic assessment strategy, Cheryl is assessed as falling within the 85-100% scale. Although her percentage completed is lower than Charles, She has completed enough work to show the quality of her work. The assessor sees that she has shown initiative in seeking clarification and in the integration of her code. The assessor also believes that she has presented sound reasoning for her design. The assessor assigns a grade of 87% (A+).

Charles made decisions on what he would include in his assignment. He didn't see some aspects of the assessment as important. As a result, he left things out. When the assessment strategy is by parts, he can still manage to obtain a pass provided he does well in the areas that he does complete. By contrast, he is penalised for ignoring those parts in the holistic assessment strategy. This shows a focus on single features or conceptual issues.

Cheryl in contrast has a less obvious change in her grade when the holistic assessment strategy is applied. This has occurred because she has consistently applied a sound process and utilised the full range of skills across in all the assessment criteria. Cheryl is showing a broader range of skills and an ability to integrate them to complete the required programming task. As well, she has presented her reasoning thus showing that she is moving into the fourth level of the SOLO levels.

We have retained a percentage complete measure in order to ensure that a reasonable quantity of the assessment is completed. However, it could be argued that if Cheryl had completed only half the quantity of work, she would still have operated at the extended abstract level. Although we acknowledge this possibility, we believe that there has to be some component of the assessment that relates to an ability to manage one's progress but at the same time, we did not want to overly penalise for not completing all of the functionality.

It is difficult with two examples to show the full impact of the two approaches to assessment. The assessment by parts strategy makes it appear as though the student can select which portions of the assignment to complete. The holistic assessment makes it clear that work must be presented that covers all areas of the assessment. Strength in coding ability does not make up for inadequate documentation or failure to implement a testing strategy. A balance and integration across all criteria is required to obtain a reasonable grade.

4. DISCUSSION

We have been applying the holistic assessment strategy to a number of different assessments over the papers that we have been teaching for over two years. Some students recognise the change in assessment strategy and provide work that utilises the full range of skills. Others still assume that they can avoid handing in work for some of the criteria. To try and overcome this problem, we have increasingly taken time to explain the assessment strategy to the students especially as the strategy is not in wide spread use on Information Systems papers.

From an assessor's perspective, we prefer the holistic approach to assessment. We find it easier to justify the grade assigned to a student regardless of the standard of work.

In assessing by parts, students could pick up good marks in one area and fail completely in another. Where process skills are important, this could mean that they could ignore completely one area of the process and still pass the assessment. An example of this is the student who does not submit a testing strategy or implement any automated tests but because they have a high level of coding skill, the quality of their work is high. This can occur even when the structure for the paper clearly highlights a test driven strategy. With a holistic assessment strategy, we can insist on a minimum level of performance in each criterion in order to gain a particular grade level (i.e. a certain percentage of the tests must be automated). In areas where full compliance may be difficult, the percentage required for a grade band can allow for the degree of difficulty.

Another assessment benefit is that we are spending less time analysing the assignment detail. We no longer have to look for those minute problems for which we can remove a mark. Instead, we focus on the whole assignment and assess it within broader ranges. The time to assess is shortened and the grades when moderated are still consistent.

5. CONCLUSION

We still have some concerns about whether we are fully recognising the cognitive skill level appropriately for each assessment level. What features of a programming assignment show an integration of ideas? In the examples provided, it may seem obvious. Cheryl has produced an integrated solution and applied a range of skills to achieve that solution. Did we pick that up form looking at her presented assignment or was it that we knew something about the way that she completed the task?

In developing the criteria presented here, we started with the SOLO levels and an example Biggs' used for assessing an essay. What we are assessing could also be regarded as a creative work (McBreen, 2001; Hunt and Thomas, 1999). We expect to continue to revise the criteria with experience.

A holistic approach to assessment ensures that students cover all the important skill areas of the assignment and not simply those that they are interested in. Missing one of the assessment criteria can have a major impact on the student grade. We believe that this is a positive aspect of this assessment strategy. Research is required to show whether the change in assessment strategy develops better software craftsmen.

REFERENCES

Biggs, J.B. (1999) "Teaching for quality learning at University", Buckingham: Open University Press.

Biggs, J.B. and Collis, K.F. (1982) "Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)", New York: Academic Press.

Hunt, A. and Thomas, D. (1999) "The pragmatic programmer: from journeyman to master", Boston: Addison Wesley.

McBreen, P. (2001) "Software craftsmanship: The new imperative", Boston: Addison-Wesley.

7 APPENDIX A: ASSESSMENT BY PARTS

Each of the following criteria will be assigned a mark within the indicated scale.

Criteria	Mark
Proportion of functionality completed in relation to requirements	20
Proportion of data validation of input data implemented	20
Quality of the submitted code	15
The quality and percentage coverage of the testing strategy	15
The quality of the user interaction design	10
The quality of the supporting documentation	10
Reasoning for design approach followed	10

8. APPENDIX B: HOLISTIC ASSESSMENT

The criteria listed below are designed to act as a guide for the assessing of your performance. Where practical, a grade indication is provided.

***Copying code* or *no understanding* of programming issues. 0-39 E**

- Application is unrelated to requirements
- Application attempts to copy example code with minimal changes
- Application delivers less than 30% of the required functionality as agreed during the project.
- No attempt has been made to apply programming standards or good application structures.
- Layout of forms and sequence of controls shows no understanding of the principles and standards.
- Less than 40% of data validation requirements are implemented appropriately.
- No testing strategy or documentation is provided.

Shows a *limited understanding* of programming and application development issues shown. 40-49 D

Some of the issues considered include:

- Application operates but has significant obvious problems.
- Application delivers 30-39% of required functionality as agreed during the project.
- Programming standards and application structures are not applied consistently.
- Layout of forms and sequence of controls on forms shows an understanding of the principles and standards but lacks consistency in application.
- Between 40-59% of data validations are implemented appropriately.
- A minimal testing strategy has been implemented or no automated testing is implemented.
- Minimal internal documentation is provided.

Is able to *complete a working* piece of code to a base standard. 50-59 C to C+

Some of the issues considered include:

- Application operates without obvious problems (i.e. does not crash when executed).
- Application delivers 40-59% of required functionality as agreed during the project
- Application inconsistently applies programming standards.
- Layout of forms and sequence of controls on forms is appropriate and abides by standards.
- Between 60-74% of data validations are implemented appropriately.
- A testing strategy for all implemented functionality is defined and up to 49% is implemented using automated techniques.
- Application is documented internally to ease understanding.

Is able to *complete a working* piece of code to a more advanced standard. 60-74 B to B+

Some of the issues considered include:

- Application operates without obvious problems (i.e. does not crash when executed).
- Application delivers 60 to 69% of required functionality as agreed during the project.
- Application consistently applies programming standards.
- Between 75-89% of data validations are implemented appropriately.
- A complete testing strategy for all implemented functionality is defined and 50 to 89% is implemented using automated techniques.
- Application is documented internally to ease understanding.

Is able to *apply* the programming concepts taught and consistently *uphold* the standards and structures from example code. 75 to 84 A- to A

As above plus:

- Application delivers required greater than 69% of the functionality as agreed during the project.
- User interaction design follows a consistent design structure.
- 90% or greater of all the necessary data validations are implemented and in appropriate places.
- Application structure is clean and matches standards.
- 90 % or more of the code is tested using an automated testing strategy.
- Application is documented externally to ease understanding.
- Documents own assessment of the quality of the code against suitable criteria.

***Shows initiative* to experiment with new ideas and is able to *present a meaningful argument* for a revised approach. 85-100 A+**

As above plus …

- Application design shows integration of task components
- Documents reasoning for choice of approach to application design and coding
- Documents task integration issues