Version 1.0 0712

General Certificate of Education (A-level)
June 2012

Computing COMP4

(Specification 2510)

Unit 4: The Computing Practical Project

Report on the Examination

Further copies of this Report on the Examination are available from: aqa.org.uk

Set and published by the Assessment and Qualifications Alliance.

The Assessment and Qualifications Alliance (AQA) is a company limited by guarantee registered in England and Wales (company number 3644723) and a registered charity (registered charity number 1073334).
Registered address: AQA, Devas Street, Manchester M15 6EX.

General

Schools/colleges that entered students in 2012 should read this Report in conjunction with the specific feedback sent to them on the publication of results. If you do not receive this feedback, please consult your Exams Officer in the first instance before contacting the AQA Qualifications Developer. The comments below highlight the observations of Senior Moderators during this year's exam and should be used with the subject specification, assessment criteria and the COMP4 Advice and Information Booklet for Teachers' Standardising to ensure that school/college assessment is accurate. For the 2012 exam, the latter can be found in the Secure Key Materials section of e-AQA together with exemplar projects and commentaries. For the 2013 exam, similar information will appear on the Teacher Online Standardising (T-OLS) website which will be available via **e-AQA**. For the 2013 and subsequent exams, T-OLS will replace the face to face meetings for the standardising of teachers.

There were no changes to the total mark compared to last year. However, for the 2012 exam, the mark for the Quality of Written Communication was incorporated into the User Manual section (this was a regulatory requirement) and the assessment criteria for this composite section was rewritten. This change to the COMP4 assessment criteria was further clarified at the series of Standardising Meetings for the 2012 exam which were held during autumn 2011.

A copy of the revised COMP4 criteria appeared on the Noticeboard of the GCE Computing pages on the AQA website in August 2011. All changes, including changes to the subject content, appeared in the specification on the AQA Website by 1 September 2011 and letters were sent to all existing schools/colleges via the Exams Officer.

For the 2011 exam, the possibility of assessing the Complexity at the Analysis stage at a different level from all of the other sections had been introduced. This possibility was for when the initial problem was undeniably 'Complex', but where the student's implementation had fallen far short of meeting the stated objectives. More schools/colleges should have used this approach than actually did in 2012.

However, the most important task still remains the need to identify correctly the complexity level of the problem and its solution from the outset at the Task setting / Pre-analysis stage.

This is the third year that the COMP4 specification has operated. Most schools/colleges, including new ones, had taken on the requirements of the specification and managed to get their students to program a solution at the level of which they were capable. Again, there were some cases where it appeared that much time had been spent on coding because of the complexity of the problem and the Analysis and Design sections were again probably written post-implementation when time was running out. It also appeared that the User Manual and Appraisal sections were sometimes casualties of poor time management by the student. Appraisal was sometimes ignored or very poorly written.

Generally, more schools/colleges had marked within AQA tolerance than last year. Some schools/colleges were very close to the standard. Schools/colleges, where assessors had included a detailed marking commentary, annotated the students' projects or added detailed comments to the **Project Log**, were generally very close to the AQA standard. These projects were much easier to moderate as it was clear where marks had been awarded. Internal standardisation, when required, was generally well carried out.

Some schools/colleges, however, had significantly over-marked their students' projects. This generally was due to an over-estimation of the complexity of the project. Many schools/colleges either did not state and justify the project complexity or simply ringed the complexity level on the **Candidate Record Form** without any explanation. These schools/colleges were extremely difficult to moderate and required each project in the sample to be marked afresh. Schools/colleges not stating and justifying the project complexity level significantly risked the chances of being out of tolerance. In some cases, the project complexity should have been reassessed following the Analysis section and, following that section, marked at a lower level throughout.

Few schools/colleges showed evidence of using the "**COMP4: Definition of problem types for projects**" document published in the Teacher Resource Bank (TRB) area of the AQA GCE Computing website to justify complexity. Those that did use it generally assessed the project complexity level accurately and consequently marked well within tolerance.

For a fully coded solution, even if students have used Data structures including Lists, User-defined Records/Types, Array of Records etc which do appear in Table 1 in the section of **FC2 -** Use of sophisticated features of programming language in the **Definition** document, schools/colleges need to look **first** at sections 2.1 to 2.4 to obtain a holistic best fit and **then** to look at relevant sections in Table 1. For example, if the work is thought to be "Complex" overall using the best fit approach and bullet point 2 in section 2.1 is being considered, then if the bullet point "*Use of sophisticated features of programming language/complexity of programming language, eg sophisticated data structures, runtime created objects, user-defined OOP classes*" is being justified, the Local Assessor then needs to look at table 1 where principle FC2 explains further what this means. Just because user defined records or arrays of records have been implemented, the method of solution does not confirm complexity, a holistic mapping needs to take place starting with sections 2.1 to 2.4 and the problem area **before** moving onto Table 1.

A **Guide to Good Practice** was also published in the TRB area of the AQA GCE Computing website in autumn 2011. It stated that if, schools/colleges wished, their students may ignore the sequential order of the headings in the specification when developing their system. For example, the system does not have to be fully analysed and designed before coding can commence. However, the project documentation must still be presented for assessment in the order indicated on the **Project Log**. This approach strongly supports prototyping. A **sample project diary** is also included in the TRB which schools/colleges may also find useful.

Some school/colleges' reports were difficult to moderate as, not only did their students not follow the order of headings in the specification, but they also used a number of appendices for evidence that should have been placed in the main narrative of the report. However, it is not appropriate to give students a word processed report template with notes and hints included, especially when the students leave these items in their reports and/or leave in subheadings that are not appropriate to their actual project eg 'E-R Models for Database Projects' in the Analysis section for a non-data processing project.

Schools/colleges should **not** encourage their entire cohort to produce very similar systems be they CAL or Ordering/Invoicing systems, with just minor differences. This approach tends to produce great similarities in code and documentation. A variety of projects is expected from individual schools/colleges.

Some ingenious non-data processing projects were seen this year including real world Complex problems involving IP Networking, Control Robotics and Mobile Phone applications including ones which were part of a suite of programs which used the phone application for data capture and a further program to process and report and display the collected data. Simulations were also popular where software was being developed for the Computing Teacher who was acting as End User. This approach is perfectly acceptable. There was a wide range of topics presented this year, including a significant number of games, many of which were appropriate and can prove highly motivating to the students. However, data processing projects are still most welcome and often make it easier for students to find genuine end users.

Despite the potential for high scoring Complex part-packaged/part-coded solutions, the better projects this year still seemed to be the ones where students had fully coded the solutions, but some very good solutions were also seen with genuine client-server applications. Most schools/colleges have moved still further away this year from customised Microsoft Access projects with minimal coding as was intended when AQA introduced this specification.

Administration

Schools/colleges that accurately complete the **Candidate Record Form** (CRF), the **Project Log** (PL) and the **Centre Declaration Sheet** (CDS) can really assist in the moderation process. Moderators always seek to agree with the Local Assessor, but this is very difficult to do if all that is provided are the section marks and final total on the CRF and perhaps a perceived level of complexity.

The **Candidate Record Form** (CRF) needs to be fully completed, including signatures on page 1 and completion of Section A on page 2. It is permissible for a school/college to provide its own marking grid for the separate assessment criteria marks and total mark in place of Section B on page 2 of the

CRF. Some teachers do not indicate their perceived level of complexity in the 'Concluding comments' section on page 2 of the CRF; this does not help their students. The Local Assessor (not the student) must normally select **one** level of complexity and justify it by reference to the criteria described in the **'COMP4: Definition of problem types for projects'** document eg, "This student has used an algorithm which was more complex than O(n squared)," or, "They have used principle FC2 – user defined data structures and classes[1]," etc. This justification is expected to be written here if it is not clearly explained elsewhere on school/college-designed documentation (if this has been done, please refer the Moderator to it). It is not appropriate only to make statements such as, "The student worked really hard on this project and produced some complex code."

Schools/colleges that have devised their own comment sheets are also to be commended. However, these still need to be attached to the CRF for the student concerned. Also it is still a **mandatory** Ofqual requirement that the signatures required on the CRF are indeed present.

If a school/college has judged the Analysis to show that the project complexity is potentially at a particular level, but the student's implementation fell far short of this, then this split level must be indicated in the Concluding Comments section on page 2 of the CRF with possibly more details given elsewhere. Further advice on reassessing complexity is given in Section 6 of the Definition of problem types for projects document.

If a separate school/college marking grid has been used, comments on the Project Log do not need to be written in full, but the student does need to tick the boxes and give the page number(s). The Local Assessor also needs to tick the boxes that they think are covered, possibly correcting the student page numbers as they go. Many schools/colleges used the PL very effectively to explain their rationale for how they had awarded their marks.

The CRF and PL **must** be the current versions for the exam series. A very small number of schools/colleges had not realised that the mark scheme had changed and separately assessed the Quality of Written Communication across the whole project. Moderators then had to remark this work.

It is an Ofqual requirement that the CDS is completed, even if the school/college enters only a few students and it is obvious that there was only one assessor. A number of schools/colleges failed to send CDSs. The sheet for 2012 had been updated so that they did not have to be countersigned by the Head of School/College. Some schools/colleges also had incorrect additions on their Centre Record Forms.

There were some very effective examples of script annotation by some Local Assessors. It is useful to indicate the location of particularly important sections of code elsewhere eg on the PL or school/college marking grid (if used) as well as just on the particular page of code.

Schools/colleges need to be aware that legacy exam CPT3 contexts must **not** be used for A2 projects because AQA completed the Analysis section for CPT3 assignments.

Projects need to be bound as a booklet, ideally with one or two treasury tags or spiral binding to enable them to be read easily. Some projects arrived in a completely inappropriate state eg as loose sheets in document folders or in very large ring binders. Moderators need to be able to scan back and forth when moderating work and dealing with a pile of loose sheets hinders this process. Schools/colleges which spiral bound students' work and then separately paper clipped the CRF, PL and any school/college marking grid to the front, rather than binding them really aided the moderation process.

A significant number of schools/colleges sent work late, without the prior approval of AQA. This strategy risks the late publication of schools/colleges' own students' results which may have serious consequences personally for them during the UCAS Clearing process. Schools/colleges are reminded that it is **their** responsibility to get the projects to the Moderator by the deadline of 15 May.

Projects must be sent to the Moderator using only Royal Mail **first class post** which does not require a signature upon delivery, retaining proof of postage. Projects sent by recorded delivery or Parcel Force (which require a signature upon delivery) risk being lost or returned to the sender if the

[1] A table in the 'Definition' document.

Moderator does not have the time or the means to travel to the (sometimes distant) distribution depot to pick up a heavy parcel. Unless you are an overseas school/college, please do not send projects using a method that requires a signature.

Software

The majority of students used Pascal/Delphi or a version of Visual Basic/VB.Net. Other languages seen included VBA, C # or C++, Objective C, Lua, Java, Python, ASP, PHP, HTML and Actionscript 3. HTML and CSS may have formed part of the solutions presented, but they needed to be combined with one or other programming languages. There was a rise in the number of students using Python this year.

It is not appropriate to use GameMaker software to create games. As it says in their advertising, "GameMaker allows you to make exciting computer games without the need to write a single line of code." Extreme care also needs to be taken with Greenfoot.

Students completing a project entirely using a high level language accounted for most of the very high scoring projects, but a number of very good client-server systems were also seen involving PHP and MySQL for example. Although a part-packaged/part-coded approach using Delphi or VB combined with MS Access or SQL Server certainly has the potential to produce high marks, few of these were seen.

In the small number of examples seen, there continues to be a problem with coursework written in Actionscript this year. Although the implementation may function well, usually there was very little student-written code with animation being carried out by Actionscript, not the student. The later versions of this language support OOP and sophisticated programming constructs and these need to be used to demonstrate high technical competence.

PHP based projects can create problems with regards to authenticating code and assessing complexity. This makes assessment difficult for schools/colleges where the student had taught his or her self how to code in PHP and the Local Assessor was in no position to judge the level of technical competence achieved. PHP-based projects are to be encouraged but care must be taken to assess these appropriately. There are many code templates on the Web for students to use. Students can be credited for adapting these to the needs of their project but they must acknowledge their source. Judgment of the degree of technical competence should be dependent on the complexity of the original source and the need for the student to understand the source in order to adapt it to their needs, as well as the degree of adaptation. Selecting complex libraries/code templates is itself complex if the learning curve is steep and high technical skill is needed to use the libraries/code templates, eg OpenGL graphics.

Project development

Low scoring students still tended to submit simple data processing projects with little complexity, but most of these were assessed appropriately by the schools/colleges.

There continues to be far too many students who pursue a project that merely used either a fully coded or a part-packaged/part-coded solution to enter data, save it, update it and to produce simple reports with little or no data transformation. If it is a part-packaged/part-coded solution using MS Access for example, all validation must be carried out in the code, not by using the Masking/Validation built into the package. QBE must not be used to design and run queries; all queries need to be hard coded in the programming language. Similarly, the Report Wizard should not be used. However, some of these techniques might be useful in an earlier prototyping stage. Regardless of the programming language used (even if it was a fully coded solution), this 'Data Strategy' type of project is not complex enough for A2 and cannot score a good mark however well the report is written and however hard the student worked to produce it.

Project reports

The majority of the high scoring students used a programming language or a combination of a package and substantial programming to demonstrate good coding skills. They also produced well-

tested, well-documented, effective solutions to real problems, with corroborated end user feedback. Particularly good practice was seen where the agreed objectives had been signed and dated by the end user.

Students who followed the reporting style of the specification and followed this up by completing the log sheets fully usually justified their assessment. The majority of schools/colleges used the Candidate Record Forms and Project Logs or their own school/college-designed documentation to record a qualitative indication of students' achievements, rather than to record how much effort a student had put into their work.

Most students achieving a high mark addressed fully all the items listed in the specification **in the context of their projects**. Some headings are not relevant to a particular project or just require a brief comment as to why they are not appropriate. The project report is not the place for general theory. As in previous years, those who scored the highest marks included well-reasoned and justified explanations of all aspects that are listed as indicators in each section of the specification and, in particular, good programming techniques which, together with a high performance level in the other sections, tended to be the features that most distinguished the high scoring students from the others.

It would assist the Moderators greatly if items are addressed by students in the same order as the specification and with the same headings, especially when essential items appear in one or more appendices without accurate references as to where these items can be found in the main body of the report. Sampling as described in the subject specification reduces bulk. There should not be lots of appendices or projects submitted as seven or eight individually treasury tagged booklets.

Analysis

It cannot be over emphasised just how important this section is to set the scene and confirm the complexity of the project. A number of students provided a proper, structured investigation using one or more formal techniques including interviews, authenticated questionnaires, observation and the use of analysed source documents. It is not appropriate for the student to assume that their personal view of a problem provides sufficient detail. This happened in the case of a number of Complex projects scoring high marks for Technical Solution. What is required is a careful analysis with appropriate evidence of the current and the proposed systems.

This section was generally handled well by the majority of students. Common deficiencies included assuming far too much knowledge of the reader when describing the problem background, omitting a detailed description of the current system and not including a realistic evaluation of potential solutions to the problem.

Not all students provided a structured investigation using one or more formal techniques. Some relied on their personal view of a problem. What is required is a carefully composed interview, evidence of the current system and an analysis of any questionnaires used.

The best examples of good practice contained a numbered, comprehensive list of highly detailed specific objectives constructed from more than one interview with the end user. The specific objectives were then improved, clarified and checked for completeness after an initial prototyping stage involving feedback from the end user. The finally agreed set of specific objectives were agreed, signed off and dated by the genuine end user. The best examples of projects also contained an outline of the proposed solution using dataflow diagrams and a careful consideration of the proposed and alternative solutions. The latter is an aid to confirming the complexity level of the problem.

It should be noted that analysis, design, implementation can be carried out in prototyping mode from day one of the project leading to a more and more complete picture of what needs to be done, ie the final analysis, and a clearer idea of how to do it, ie the final design. The structure of the write up follows the Waterfall model, specified in the specification, but the detail of each stage can be achieved very effectively by following a prototyping approach. Staging the write ups of each stage of analysis and design so that sufficient prototyping can be carried out involving the end user often leads to a better understanding than an approach in which no design/coding begins until the analysis has been signed off.

The analysis data dictionary still remains a problem area with many students ignoring it or just copying the one presented in Design. The analysis data dictionary captures the data items and their properties that appear on the artifacts of the current system, eg a paper based booking form or an order form and that which will be required in the proposed system. The properties will include such items as length, range and size of fields, data types of fields and example values. The data types may come from any of the formal methods employed and will be as perceived by the end user, eg a whole number, a decimal number with two decimal places; not the data types that will be used in the chosen programming language eg Long Integer, Float(5,2) which will be stated in the design data dictionary.

Numerous students totally ignored the section on data volumes, even though it was relevant in their case because they were tackling a data processing problem either fully coded or alternatively part-packaged/part-coded. In some cases, the data volumes may be relevant in determining the level of complexity of the project. If it is not relevant for a particular problem, students are expected briefly to justify this reason for its exclusion.

Not all students produced DFDs for the existing system even if they did for the proposed system. Students should be encouraged to use standard notation and labelling, especially for DFDs and ERDs. Non-standard data flow diagrams and those without data on flows were credited far too frequently by Local Assessors.Too frequently there was no data labelling of the flows where these were not obvious, but sometimes actions were given. An adaptation of DFDs in which the datastores perform the role of event/message stores and the processes perform the role of actions that respond to specific events/messages is perfectly acceptable.

Many students produced clear and measurable objectives that were specific to their projects, rather than generalised, 'text book' type objectives such as 'user friendly' or 'system must load in a certain number of seconds' (which are not actually wrong in themselves). 'SMART' objectives as defined in the current specification enable students to achieve higher marks in both this and the Appraisal section. An example of a specific objective is, "The system must allow the user to enter the starting x, y coordinates of the object specified in specific objective 12." An example of a constraint specific objective is, "The system must be able to project through a data projector of maximum resolution 800 x 600." The objectives of some students often indicated little processing, but schools/colleges still claimed these to be 'Complex'.

The complexity of a problem can be assessed by examining the specific objectives, the dataflow diagrams of the proposed system, the data model where the latter is appropriate, an outline of the proposed solution and by discussion with the student. This becomes more difficult if the student fails to provide sufficient detail. In this case, examination of the design and implementation should throw light on the complexity level.

The process can be helped by an initial prototype and identification of the critical path for the project, ie how difficult is the most difficult path in the project? For example, synchronising files on two geographically separated computers connected by the Internet. The critical path in this case is reading the properties of a remote file, ie data and time of creation and transferring a copy of the most up-to-date file to the other computer. If this is clearly stated as a specific objective(s) then it becomes easier to assess the project's complexity because the steps that contribute to the critical path are sufficiently articulated for an assessor in the school/college to focus on the problem(s) that must be solved for the project to succeed.

Many identified end users (client - users) seemed to have very little influence on the specific objectives and could provide little or no meaningful feedback in the later Appraisal section. A few students got their end user to sign off and date the agreed objectives. This is good practice. Students producing clear and measurable objectives that were specific to their project allowed the complexity of the project to be clearly identified. Vague, non-specific, non-quantifiable 'specific' objectives are unhelpful at A2 level as they would be at any level seeking to meet specific goals.

'SMART' objectives enable students to achieve higher marks in both this and the Appraisal section. The basic purpose of their system should be included in their understanding of the general objectives eg, "To produce a system which helps teach the Advanced Level Mathematics Topics of Matrix Transformations."

Please note that the awardable mark for Analysis is linked to the level of complexity of a project. If a school/college determines the level of complexity as anything other than 'Complex', they cannot then award marks in Band 4 for Analysis.

After reading the Analysis section, the conclusion should be: "Yes, I know what the aim and scope of this project is." If this is not the case, and/or all the relevant Analysis assessment criteria are not fully met, then the project cannot warrant a mark from the top of the range for the complexity level assigned to the project.

Design

Design is about how to solve the problem in general and in detail. The quality of the design is judged in two ways. Firstly, by the structure of the solution, ie how good is this structure? Secondly, by how well the design is described, ie could the description be used successfully to produce the solution? The first relates to planning and the second to how well the planning is reported.

Many students neglected the basic principle that this section should have all the details required to enable a competent third party to build the system. Some designs were well planned and showed a high level of understanding of the requirements of an A2 project. Most high scoring students show a good appreciation of data structures, file design and normalisation where this is required

In the design section the biggest single omission were algorithms for data transformation. Very few students provided algorithms that showed how data were to be processed or transformed. If these algorithms are omitted the design will not be feasible, severely limiting the maximum marks that can be awarded. Many schools/colleges missed this point when assessing this section.

Security, integrity and test strategy are problem areas because many students still do not address these in the context of their system, but give a generic overview or quote general theory.

The **Guide to Good Practice** mentioned earlier encourages the prototyping approach during the development of the whole project. Screen shots that look as if they were produced during implementation are now perfectly acceptable here but it would be good practice if they were identified as prototypes. Such prototypes can corroborate end user involvement by being signed off and dated.

In the past three years, we have been trying to widen the range of project types submitted. Traditionally, when the majority of projects were data processing projects, an emphasis on data validation and data security was deemed both relevant and important. However, a more important requirement is, "Does the solution/algorithm meet its specification under all circumstances?" For an algorithmic type of project (one that necessarily seeks out one or more suitable algorithm(s) to solve the problem) the emphasis should be on the 'correctness' and effectiveness of the algorithm.

Some designs were well planned and showed a high level of understanding of the requirements of an A2 project. Many students failed to score high marks by not conveying an understanding of the concepts of validation and security beyond a very superficial level, such as field length or presence checks. The high scoring students showed a good appreciation of data structures, object-oriented design of classes and objects, file design and normalisation where this was required for their solution. The normalisation process can be demonstrated by showing how the data model develops from the analysis data dictionary and the ERD shown in that section which may contain unresolved many to many relationships to 3rd normal form using standard notation, or by confirming that the entity descriptions derived from the E-R diagram are in 3rd normal form or BCNF using one of the tests, eg every non-key attribute is a fact about the key, the whole key and nothing but the key. Some students are still failing to show proof of normalisation and simply producing ERDs, often incompletely labelled, especially for the relationships where labelling would add clarity.

There seems to be a great reluctance to show the HCI design as clearly annotated sketches or by the use of a graphic package or the form designer related to the chosen programming language. This allows the student to give clear explanations as to the rationale behind the input and output screens, and how they meet the user needs. Chapters 7.2 and 7.6 of the AQA endorsed A2 text book[2] contain guidance on HCI. The HCI design should also convey guidance in pictorial, annotation and prose

[2] *'AQA Computing A2'* by Kevin Bond and Sylvia Langfield, published by Nelson Thornes ISBN 978-0-7487-8296-3

forms that will enable the HCI design to be implemented. For example, the rationale may cover the reason for choice of fonts, font sizes and colours for text, foreground and background as well as specifying these. Really good practice was demonstrated by the genuine end user signing and dating the agreed final designs. This provided evidence of end user involvement at various stages of the project. HCI design involves the end user and other potential users. HCI design is necessarily an iterative process. It is now acceptable to include screen shots of the implemented system created as part of the prototyping process

Very few students gave clear algorithms relating to the data transformation to be programmed by the student. These algorithms should be in a form that could be coded in any language and need to concentrate on the complex parts of the system that the student is going to code. Frequently, the 'sample' was of very basic functions. We are not very interested in the algorithms required for login, form navigation and password changes etc. Algorithms in pseudocode or structured English must be given if a high mark is to be scored. Those that were given were generally poor with the notable exception of some schools/colleges whose methodical approach made moderation of this section relatively simple.

Frequently 'algorithms' were post implementation code extracted from the full code listing, not in pseudocode or structured English as required. The algorithms presented in this section are one of the places that Moderators inspect closely at an early stage to confirm the potential level of complexity of the project. It is not appropriate to omit key design algorithms and forward reference those later given in the System Maintenance section because the purpose of the Design section is to allow a competent third party to implement the required system. Students were heavily penalised for doing this because the design is simply not feasible without design algorithms.

If students are going to need to embed SQL queries in their programming code because they will be creating part-packaged/part-coded solutions, it is appropriate to include the designs for these queries in the design section. Similarly, DDL script designs for table creation should be included here as appropriate.

Security, integrity and test strategy are problem areas as many students are still not addressing these in the context of their system, but presenting general theory rather than applying it to their particular projects where relevant.

Technical solution

In most cases students showed that they were competent programmers. A wide range of programming constructs were used and many students appropriately used objects in their programming. Very few students annotated their code fully. For projects in Visual Basic, Java and Python, this sometimes made assessing what was package-generated code and what was students' own code very difficult. This was especially the case with VB.Net.

A very small number of students did not actually submit any code at all.

High scoring students used advanced features appropriately. It is helpful if the complex sections of the code written by the student are highlighted. This may help confirm that the student understands what they have done.

Code obtained from another source must be acknowledged and cannot earn the student credit, but may form part of the solution.

The Computing specification does **not** require a separate ICT- style implementation guide for the Technical Solution. Evidence for the mark in this section comes from the Testing and System Maintenance sections and possibly the User Manual where screen captures may be the only evidence of the working system if other sections are not complete.

This section is also closely linked with the level of complexity as well as the level of technical competence displayed. Many students programmed a solution at the level at which they were capable, but again there was a great deal of evidence of much time being spent on coding and only when this was completed or when time was running out were the Analysis and Design sections finally attempted. The complexity demonstrated in the Technical Solution - eg appropriate use of OOP - is one way that

the complexity of the problem as defined at Analysis can be confirmed. A technical solution that has some embedded SQL or that creates objects at runtime, for example, does not in itself make the problem being solved complex. However, if at the Analysis stage doubt is present as to the appropriate level of complexity, then evidence from Design and Technical Solution stages should be used. A 'best fit' method of assessment as opposed to a 'tick box' approach is required during assessment.

The project is not made complex just because a student has used Java, C++ or Python for a fully coded data processing project involving minimal processing of a sample database. Similarly, writing an iPhone application using an application generator and incorporating published library code is not 'Complex'; it is the degree of technical competence of the student-written code and the creativity involved that is important.

A number of schools/colleges again allowed students to use the data source configuration wizard of VB.Net or the Express editions of the language to fill a dataset with data ie they used the TableAdapter feature. This is effectively a package-generated code to establish database connectivity and does **not** confirm complexity of the solution. There are other more appropriate non-wizard methods for database connectivity that should be used to gain high marks.

Only some Local Assessors made clear statements on the Project Log or other school/college-generated documentation such as, "A fully working robust system," "most processing objectives achieved," etc. Those that did significantly assisted the moderation process.

Many of the students who used a programming language produced well-structured listings. High scoring students used advanced features appropriately. It is helpful if such code actually written by the student is clearly highlighted by the Local Assessor and, in particular, the use of advanced features such as user defined data/record structures need to be clearly indicated.

There is no longer a specific reference to parameter passing in the specification; this is assumed as demonstrating high levels of technical competence where parameter passing is appropriate. Solutions are not Complex simply because parameter passing has been used, although this was used as the justification for perceived level by at least one school/college.

Some students achieved a high mark in this section and went on to gain a high mark overall, but others seemed to spend the majority of their time on their coding to the neglect of the other sections that account for the remaining 55 of the 75 marks available.

If a part-packaged/part-coded approach has been adopted, it is important that validation is coded by the student using the programming language and that SQL queries are fully embedded within the code as discussed in the Teachers' Standardising Meetings. Similarly, the reports also need to be programmed, not generated by wizards. A small minority of students again simply customised a database package with minimal amount of self-written code. This is not in keeping with the philosophy of the new specification. Creating a solution which links a spreadsheet package eg MS Excel with Delphi or VB still counts as a part-packaged/part-coded system and the appropriate assessment criteria need to be applied. In simple terms: 'Package' does not just mean a database.

System testing

The testing section was generally completed well, but the use of boundary testing is still contentious. Very few students stated they had done tests involving boundary (extreme) data and of those even fewer knew what such a test is and so gained little credit. Most students, and indeed whole schools/colleges, sometimes confused boundary data testing with erroneous data testing.

Testing is one of the few sections that is assessed without reference to the perceived level of complexity. Testing by many of the average and weak students was trivial. It is **not** appropriate to have multiple tests for login passwords and/or to prove that all the buttons open/close forms and/or reports. It needs to be emphasised that what is required is proof that the students' coding works to produce accurate results.

The higher scoring students showed evidence of carrying out a thorough test plan effectively by producing a table of results and printouts to prove that it was well annotated, not excessively cropped,

cross referenced and ideally authenticated by the assessor. There should be no more than two screens to an A4 page to make them clearly readable. Evidence needs to be in the testing section, rather than relegated to appendices, especially if it is not properly referenced.

It would be useful if students were able to reprint the column headings on subsequent pages of the test plan.
The higher scoring students showed evidence of carrying out a thorough test plan effectively by producing a table of expected results and screen captures to demonstrate that these results were achieved, the screen captures were well annotated and were cross referenced to the table entries; also they were **neither** heavily cropped **nor** simply labelled with a figure number. Ideally they might be corroborated by the Local Assessor or end user.

Few students used boundary testing effectively. In some cases boundary data was justifiably non-existent because of the type of problem being solved, but these instances need to be stated clearly and justified. Similarly, erroneous testing may not be possible because of the use of slider bars etc in the implementation. Sampling is needed for this section to avoid coursework of potentially massive length. Some students incorrectly interpreted extreme testing as extremely high or extremely low erroneous data. The following definition may prove helpful: "Boundary values are those that are just **inside**, **on** and just **outside** the range of allowed values". In this case, boundary testing maps onto extreme testing. High scoring students often clearly identified if the data was erroneous, extreme (boundary) or normal. A number of students interpreted 'extreme' data incorrectly ie they chose extremely large or small unlikely values.

In simple terms, students need to demonstrate that the really clever parts of their system worked correctly, not that they could trap an incorrect password, accept a correct password and change a password etc.

Maintenance

This section continues to be badly executed. All of the items listed in the subject specification need to be addressed fully so that what the student has done can be clearly seen.

Some schools/colleges used 'Prettyprinter' type software to produce the code listings and most students had used appropriate, self-documenting identifier names for variables constants, types and subprograms, but many students did not annotate their code. Few students could write algorithms or a suitable alternative, often confusing program code with pseudo-code and yet this was still credited by the school/college. Only a small number of high scoring students produced these at a level that would enable their systems to be effectively maintained.

Better students included subprogram, procedure and variable lists (including scope as appropriate). However, frequently the procedure and variable listings were completely absent. This would make maintenance of the system very difficult. However, in a high scoring project there may be too many procedures and variables to all be documented separately.

Unless the code listings exemplified a high degree of self documentation to a degree that minimised the need for substantive variable and procedure lists, students were penalised. A list of the main procedures and important variables, with some explanation of their importance and role, should be sufficient, backed up by well written code. Useful information about the main variables should also be contained in the design data dictionary.

Some students who had programmed in Java often used the Javadoc tool to create some of the documentation for this section, but others stated that it generated a lot of hard copy and included just a sample.

Not all students produced algorithms or a suitable alternative and only a small number of high scoring students produced them at a level that would enable **their** systems to be effectively maintained. If the algorithms originally proposed in Design are omitted or have been changed during Implementation, they need to be updated here. If they have not been changed, a clear reference back to the algorithms in the earlier Design section must be given. This is allowable; forward reference from the Design section to System Maintenance is not allowable.

Please do not include any package-generated code. This was especially relevant for those students who had used the data source configuration wizard of VB.Net or the Express editions of the language to fill a dataset with data. This is effectively package-generated code to establish database connectivity and does **not** confirm complexity of the solution.

A similar problem occurred with a school/college whose students had all used PyQT to generate the HCI required for their Python applications. Moderators are not expected to trawl through the submitted code to decide what was written by the student and what was effectively wizard generated. The student is expected to identify clearly all of the student-written code with a highlighter or similar. Another school/college whose students had used Javax Swing to create the HCI for a Java application had done this; it greatly helped the moderation process.

User manual including Quality of Written Communication (QWC)

This year the user manual section also included the marks for quality of written communication. This change did not seem to affect the marks that students achieved, except for the very few students who produced no user manual. Most students scored appropriately for QWC within this section.

The main omission from this section was the lack of error messages and/or troubleshooting and especially error recovery, which limited students' marks. A significant number of good students missed one or more of the table of contents, the introduction or the installation instructions and so limited their maximum marks.

Some students produced good user documentation. Many showed screen shots in the documentation, but too many of these were excessively cropped. This was particularly the case with error messages where the data causing the error needs to be clearly visible as well as the resulting message. Many failed to convince that the system worked, because there was very little data, or nonsense data in the system when testing. A number of students still failed to provide a manual to the complete system as the subject specification requires. A small number of schools/colleges failed to use the up to date assessment criteria that detail how to award the user manual mark according to level of complexity of the solution and the number of features on the QWC page that are addressed.

However, most schools/colleges demonstrated that they were able to operate the new assessment criteria correctly. Having first assessed the complexity of the project, the subject marks were assessed and then the QWC criteria were checked to determine the overall mark.

It is again disappointing that many students failed to use a word-processor appropriately. Many project reports were missing such basics as headers, footers, word-processor-generated page numbers and a table of contents. Page numbering was hand-written throughout the whole report in some cases. Minor spelling and grammatical mistakes that did not detract from the meaning were not penalised.

Not all students provided appropriate installation instructions for the different types of user/administrator of their system, especially for client-server systems.

Nevertheless, most students incorporated screen captures with appropriate explanations, but perhaps they should have considered the needs of their users in more depth. Some students had provided the manual for the user to assess during acceptance testing and this was signed off in some cases. This was again good practice.

Error messages linked to screen captures, trouble shooting and recovery procedures generally needed to be more extensive. Sometimes, there was very little data in the system when either testing it or writing the user manual. Recovery procedures, which are very important for data processing type projects, were frequently absent and this reduced the Moderator's marks in many cases.

Appraisal

Most students wrote honest evaluations of their performance against their objectives. The majority of students referred to the objectives set out in their Analysis section but not all fully evaluated **how**

these objectives had, or had not, been met. It is not appropriate to use a yes/no approach. Each objective needs to be fully evaluated to earn marks. It is particularly poor practice when the numbered objectives are copied and pasted from analysis to appraisal without editing the word-processor's auto numbering. Assuming that students had produced a detailed list of SMART objectives in agreement with their end user in the Analysis section, there was easy scope for comparison and discussion of objectives versus outcomes.

There were only a few schools/colleges that had students' work submitted with significant evidence of genuine, authenticated user feedback and because this forms the basis for analysis by the student and suggestions of further development, these marks were frequently not available. This feedback needs to take the form of a dated and signed letter on headed notepaper or a **genuine** email from the end user. Local Assessors should authenticate user feedback by communicating with the end user and signing and dating this section of the report, especially where it is obvious that the student has just typed this into the report.

The best feedback evidence had criticisms as well as praise for what had been achieved by the student so as to allow analysis of the feedback and derivation of possible improvements etc. Even if genuine feedback had been presented, many students still failed to analyse it and then use it as the basis for discussing possible future developments/improvements. Many possible extensions (if indeed present) appeared to be entirely student driven.

Statistical data and information on grade boundary ranges www.aqa.org.uk/over/stat.html

UMS conversion calculator www.aqa.org.uk/umsconversion