Guidelines for Conversion MSc Software Development Projects David Bernhard

Software development projects

This document is about software development projects, as opposed to theory or research projects that may involve some element of software development. In a research project for example, what matters is almost exclusively the result and the write-up thereof. In a software development project, the emphasis is on the quality of your software and this document explains what might be meant by 'quality' and how this might influence your marks.

As a rough guideline, imagine that you show your software (including source code) to a potential future employer as part of the recruitment process, to say "if you hire me then that is the kind of thing I can do". If they would clearly hire you, it is probably good quality software.

One way to break down software quality into a list of factors is the following:

1. It does something useful – it solves a real problem for someone.
2. It is mature, e.g. ready to use for real.
3. It does something technically difficult.
4. The code is well written, structured and tested.
5. Its user interface is well designed (and there is evidence of this).

Marking

In your project, you produce a 'masterpiece' that shows your abilities after two terms of taught material. You will learn new material during the project too, but you will do this learning individually and as needed, rather than following a list of topics set by your supervisor.

Your project will be marked primarily by your supervisor and a second marker. The main part of the marking will be the two markers reading and evaluating your thesis, which means that any claim you want to make in favour of your software must appear in your thesis – ideally with evidence. You will also demonstrate your software to both your markers, and they may look at the source code that you must submit along with your thesis, but the thesis itself is most important for your mark.

We will judge you to a large extent based on the features that you claim are important. For example, we would mark a project differently depending on which of the following you might claim:

- The software is easy to use for young children.
- The software is aimed at a technical audience to solve a complex problem.
- The software improves the efficiency of an existing solution (e.g. it is 10% faster).
- The software scales to millions of concurrent users.

To increase your chances of good marks, you should make sure that:

1. You make clear in your thesis what claims to quality you are making about your software.
2. You provide evidence for everything that you claim.
3. You are familiar with standards in the area that you are claiming for.

For example, if you claim usability then you should know a bit about the methods for evaluating this (as introduced in Research Skills).

Grade bands

Conversion grade bands are fail (below 50), pass (50–59), merit (60–69) and distinction (70+). In theory, the scale goes up to 100 but in practice marks above 80 are extremely rare and marks of 90 or higher are almost unheard of – we reserve these largely fictional marks in case a student ever makes a world-changing breakthrough during their project.

A mark in the low 60s indicates that a student has achieved the learning outcomes that we have set; marks below 60 indicate that a project was deficient in one or more areas (and not good enough in other areas to compensate for this). A mark of 45–49 is a signal that the project is not good enough to pass, but the markers believe that the student can improve this by resitting the project; a mark below 45 is a fail without resit. You should view a 70 mark as '100%', with everything above that indicating an excellent project.

One way of marking a project is to start with an initial guess of 60 and then go up or down while reading the thesis if the positive points exceed the negative ones, or vice versa. I will only recommend a distinction mark if there are no major negative points and there is a strong reason that speaks in favour of a distinction, i.e. I could complete the sentence "I am choosing to give this student a distinction because …".

Your degree will be awarded based on the minimum of (1) your average taught mark and (2) your project mark. For example, even if you have a taught average of 80, you need a 70 in your project to get a distinction – a 68 would be an MSc with Merit. Your markers are aware of this fact; although they do not look at your taught mark, they know that awarding a 68 rather than a 70 is deciding that this student is definitely not getting a distinction.

Thesis proforma

The usual structure of a thesis is as follows:

1. **Abstract** – maximum one page (max. 1/2 page preferred) that very briefly describes what your project is about. No technical detail, and as long as you get the key points across, shorter is almost always better.
2. **Introduction** – this is perhaps the most important section, which you should spend sufficient time on re-writing until you are really happy with it. After reading this section alone, a marker should know what your project is about and what to look out for in the rest of the thesis. In this section you do not go into too much technical detail except what is really essential for understanding your project. You do state the problem you are trying to solve and explain what you have done to solve it. In the introduction, you also state your claims to quality, and ideally explain how you are going to provide evidence for them.
3. **Background** – think of this as the state of the world before you started your project. In a research project this is an extremely important section as it places your work in context of existing literature and shows your awareness of all relevant previous work. In a software development project, the background section is less crucial, you can use it to introduce the problem you are trying to solve in more detail, mention technologies that you will use, or compare existing attempts to solve your problem (and explain why the problem is not fully solved yet).
4. **Body** – here you present everything necessary to understand your project. For example, the body could contain a "user manual" of your software with screenshots to evidence your

software's features. You might draw the marker's attention to design decisions or patterns used, explain your development process, or show the results of early evaluations and the conclusions you drew from them for later development. The important point here is to suitably highlight, that is draw the reader's attention to, points that you think are particularly important when marking your work – for example, what you consider to be the high-quality aspects of your software. As always, back up claims with evidence.

5. **Evaluation** – this is the section where you provide evidence for your claims to quality. Typically, this will include some form of study with members of the target user group, ideally including a proper statistical evaluation. You might want to start this section by repeating your claims, then going through them one by one and providing evidence, or referring back to evidence in previous sections.
6. **Conclusion** – (you can merge this section with evaluation if you want) here you can be a bit more informal again. End your project by summarising again what problem you set out to solve and how well you solved it, talk about what the challenges were during your project and what you would do differently if you started it again, or what you would do next if you had another few months to work on it.

Honesty

Not all projects will achieve everything they set out to do, and this is ok. If you planned to do three things but only achieved two of them, please be honest about this – it will get you a better mark than claiming all three, but your markers notice that the third claim is contrived. In general, claiming something incorrect and/or without evidence will lose you marks. Being honest about your project's weaknesses as well as its strengths might gain you extra marks.

Examples of quality

On the following pages you will see a more detailed description of what quality might mean for different aspects of your project, roughly mapped against different grade bands.

- The points under 'Fail' are negative examples, and you should make sure that you avoid all of them in your project. Although a small number of these will not immediately fail an otherwise strong project, you can imagine that every one of these points found in your project will reduce your mark by some amount.
- You should aim to satisfy a majority of the points for the grade band you are aiming at, e.g. if you are aiming for a Merit then you should meet all the 'Pass' positive requirements and most of the 'Merit' ones.
- Of course, the points for the aspect(s) that you are claiming in your thesis will carry the greatest weight.

Spelling, grammar, style and layout

You are not doing an English degree, and your English is not the most important part of your mark by far. In particular, there are lots of constructions that, while technically incorrect, are still perfectly understandable but just show that the writer is not a native speaker of English (often enough, the type of mistakes made even give a rough guide to which culture the writer is from). This is not a problem, and we will not mark you down for this.

However, the prerequisites for joining the CS conversion course include:

1. English language to level B2 (upper-intermediate independent user) on the CEFR scale.
2. A previous degree (upper second class honours or equivalent).
3. GCSE mathematics (A-level preferred).

Final theses that fall well below the requirements of any of these three points are likely to fail. In particular, we assume that your previous degree has prepared you for matters such as academic writing, citing and referencing, researching source material, and avoiding plagiarism. We teach you some of the most important points again in the Research Skills unit, and expect you to be able to do these things correctly in your thesis.

The main reason that a thesis can fail for matters of spelling and grammar is that the student's English is so poor that they do not manage to get their key points across. Using the wrong words or expressions that mean a different thing to what you intended to say can also contribute to failing the project, especially if you end up saying things that are factually incorrect as a result.

For style and layout, if you are not sure what to do then use Microsoft Word and do not mess with the default settings unless you know what you are doing. In particular, do not set the line spacing to double, even if you needed to do this for essays back when you were at school.

I recommend that conversion students use Word, not TeX, unless they are already TeX experts – there are lots of things you need to know to use TeX well including how to typeset punctuation, how to properly layout figures and tables etc. You risk losing time fighting the software that could be spent on your project. Also, Word has a spellchecker built in and you are expected to use one.

Spelling, grammar, style and layout	
Fail	– Frequent mistakes of the kind that could be caught by using Word's spell checker. – Repeated and major sloppy mistakes that even a beginner in English should know. → example: not starting sentences with capital letters. → example: not ending sentences with punctuation. → example: fragments rather than complete sentences. → example: colloquialisms, e.g. 'u r' for 'you are'. – Frequent elementary grammar mistakes e.g. its/it's, your/you're. – Incorrect definition and use of technical terms in ways that affect the project.
Pass	– Quality of English language that would be acceptable in a B2 or higher exam. – Document has been proofread.
Merit	– High quality English that flows naturally and has few if any spelling and grammar mistakes of the kind that would immediately be obvious to a native speaker. – Style and tone are appropriate to a dissertation.

Usefulness

Usefulness could be described informally as 'how much of a change has your software made and to how many people's lives?'. Of course, one can trade off quality and quantity: software for a very small group of users with a rare disability, that makes a huge improvement to their lives, would count as useful for a different reason to software that addresses a minor everyday annoyance for a large number of people.

Usefulness is distinct from technical difficulty: sometimes a very simple program, well executed, can have a large impact. Usefulness and maturity (next page) are harder to separate as less mature software is generally also less useful. However, it is both possible for software to be mature and not useful (it runs but doesn't solve the problem it's targeting), and useful but not mature (works only on the student's laptop, but tests with target users show that they find it useful).

Usefulness	
Fail	– It is not clear in the thesis who the software is for, or what problem it solves. – The thesis claims that the software does lots of different things, but it does none of them well.
Pass	– The thesis makes clear who the software is for (target users) and what problem(s) it solves for them. – The problem that the software claims to solve is real and does not seem contrived to justify the project. – The software does actually solve the problem it claims to solve, or at least makes some steps in this direction. A project that is essentially a clone of some existing software but has no 'unique selling point' would normally not get a higher mark than a pass.
Merit	– The software solves a real problem for a real user or group of users. – The thesis provides evidence of a measurable impact. – The software solves a problem for which no previous software solution existed, or is better than previous solutions in some way. → example: providing a free and/or open-source solution to a problem that previously required commercial software would count as 'better'. → example: writing a desktop application for something that previously existed only as a web application, or vice versa, if there is a good reason why this adds value for a particular user group. – Where other comparable software exists, the project is at least of a similar quality to existing solutions.
Distinction	A point in favour of a distinction (for both usefulness and maturity) would be if the software is already being used productively by the target users, especially if it is making a measurable positive impact on them. → example: your software is a booking system for a tennis club, you can show in your thesis that the club is already using your software to manage its bookings and are highly pleased with it. → example: your software is an app, and has a decent number of downloads and positive user reviews (by genuine users of course). Another point in favour of a distinction would be if your project is comparable in quality, or better than, an established free or commercial product.

Maturity

A mature project is one that is ready to be used for real, or ideally is already being used for real. It is a bigger step than one might at first imagine to go from 'works in my development environment' to 'works for real', especially if you are writing an application for the first time. Consider a web application for people to share recipes for example, with a client side of the application produced in JavaScript and a server-side application hosted on a 'traditional' server with a database. To use this application for real, all the following has to be implemented correctly among many other things:

- Hosted on a server somewhere, and has a domain name to get a proper URL.
- Database properly configured (user accounts, backups etc.)
- Proper log-in system, including secure password storage and a way to reset lost passwords.
- If the password reset is by e-mail, then the application needs to be able to send e-mail.
- Common assets such as stylesheets and scripts would typically be served from a third-party content distribution network.
- Proper error pages for 404s etc.
- Logging and ways to notify the developer or administrator if something unusual happens.
- Certificate for HTTPS, and either automated renewal or policy for doing this manually.
- HTTP security headers (CORS) set correctly for the domains in use.

Being mature enough to use for real is definitely not a requirement to pass a MSc project, but is strongly recommended for projects that are aiming for a distinction. Focusing on maturity will also provide you with many skills for a future career in software development, as well as things to talk about in a technical job interview. After all, customers who pay for software generally expect it to be mature enough that they can use it.

Maturity	
Fail	– The software does not work, or only works when operated by the student. – The software is meant to solve a practical problem, but is more of a proof-of-concept than a working solution. → example: the software is aimed at non-technical users, but needs a sequence of commands run on the command line to start it.
Pass	– The software works as demonstrated on the student's laptop, even when used by the markers. – It is at least close to being ready to use in practice. *No higher than pass* if the student has omitted important considerations for deploying the software in practice (see the examples above) and this is relevant for the project.
Merit	– The software is ready, or at least available to use in practice. → example: for a web-based application, it is hosted on a server and has an URL. → example: for a desktop application, binaries and/or source code with build instructions are available. – It looks and feels like a real program / website / etc.
Distinction	A point in favour of a distinction for maturity as well as usefulness would be if the software is already being used productively by your target users.

Technical difficulty

Technical difficulty can be traded off against maturity/usefulness as long as you are clear in your thesis what your aims where and what results you are claiming (and you have evidence for your claims).

If the technical difficulty of what you want to achieve is by far the most important part of your project, then it may be worth approaching it as a research project rather than a development project, which would involve taking a different approach in your thesis.

Technical difficulty	
Fail	– Trivial work. – The project essentially follows one or more online tutorials without building anything individual on top of them. – Code copied from online or other sources (whether with proper attribution or not) but little or no individual contribution of the student at all. – The is primarily a wrapper around an existing library that already has all the required functionality and there is little to no value added by the student's work.
Pass	– Project is clearly non-trivial, but still of limited scope and/or a straightforward assembling of existing components (although, such a project could still get a higher mark on the grounds of usefulness). – The student needed significant help and guidance from their supervisor to complete the project. *No higher than pass* if the project difficulty is comparable to that of the coursework set in the taught units on the conversion MSc (unless it is very strong on usefulness).
Merit	– The student has solved a problem above the average difficulty level one would expect of a conversion MSc student. – The student has worked independently, both in learning required new skills during the project and in identifying which new skills they need to learn.
Distinction	One criterion in favour of a distinction would be that the student has solved a problem whose technical difficulty is clearly beyond the level we expect for a student on the conversion MSc.

Code quality

Code quality could be described as things you see from looking at the code, not from running the code. Obviously, code that does not run is poor quality, and code with fewer bugs is also higher quality. All other things being equal, between two applications that both look and work identically to the end user, if one has most of the code in a few giant functions with no apparent structure and the other has the code nicely split up into functions and classes with good documentation, the latter has a higher code quality.

An informal test of code quality is "how (un)comfortable would I be if I had to take over this code and maintain it or develop it further?"

Code quality	
Fail	– No source code submitted, submitted code does not build/run due to syntax errors, or code not submitted in a usable format (e.g. printed in a PDF). – Most functionality is in a several hundred lines or more main function. – Serious security vulnerabilities that have been covered in the taught units, e.g. SQL injection.
Pass	– Code is structured in functions, classes or some other sensible way. – Comments where appropriate, but not for trivial things. – Build process, dependencies etc. clearly documented. – Except where restricted to particular platforms by design (and stated as such in the thesis), code builds on all 3 major desktop OSes (windows, mac, linux). – Some form of testing or validation. – Ideally, version control has been used. *No higher than pass* if there are any serious security vulnerabilities or other clear anti-patterns in the code.
Merit	– Automated build process using appropriate tools for the language and platform. – Thorough automated testing (absolute minimum: unit testing), ideally as part of the build process. – Good code structure and documentation. – Comments or documentation emphasise higher-level features such as design decisions or patterns. – No significant security vulnerabilities or anti-patterns in the code. – Facilities for handling user input/output work for languages other than English (e.g. accents handled properly). – If the software is a web application, accessibility has been correctly addressed.
Distinction	One criterion in favour of a distinction would be that the aspects described under 'merit' have been addressed with excellence and are clearly presented as such in the thesis. For example, a clear presentation of how cross-cutting concerns such as security, internationalisation, accessibility etc. have been addressed throughout the project, with specific examples as evidence, might be an example of excellence.

User experience

User experience, and a usability evaluation, is not a necessary part of all software projects. A project would only fail for usability reasons if usability was clearly an important part of the project aims (e.g. designing software for users with dementia), or user experience is claimed to be an important part of the project in the thesis but the project does not deliver on what it promises.

There are established research procedures for usability testing – it is not enough to show your program to a friend and ask them if they liked it. Some of these are taught in the Research Skills unit, and if you claim usability as one of your achievements you are expected to do a proper evaluation to provide evidence for this.

Usability, user interface etc. – design, implementation and evaluation	
Fail	– Usability is claimed as a significant feature of the software, but evaluation is absent, superficial or has clear mistakes. → Example: citing Nielsen's usability principles and claiming this makes the product usable, without any link between the product and the principles. → Example: claiming a quantitatively significant A/B difference with no statistical evaluation at all, or an evaluation that is statistically insignificant. – The software is clearly not appropriate for the intended target users. → Example: accessibility issues (e.g. colour alone used to distinguish important information) where there is a good reason to insist otherwise.
Pass	– A real target user/group is determined in the thesis and there is evidence of engagement with this group. → Example: if the software is meant for elderly, colour-blind etc. users, then it has also been tested with such users. – Where general principles are mentioned, they are linked to specific features of the project. – At least a qualitative evaluation of the project usability at the end. – Evaluation meets the minimum standards as taught in Research Skills.
Merit	– Clear design phase that identifies specific features or constraints for this project based on the target user and goes into some depth, possibly citing research. – Evidence that points identified in the design phase have influenced the implementation and the writing on usability in the thesis is not just retroactive. – Testing of multiple designs or iterative design process – evidence that the project has reacted to results of user tests (which implies user tests are not just carried out at the very end). – Good evaluation with appropriate statistical techniques and valid conclusion/interpretation.
Distinction	One criterion in favour of a distinction would be that the quality of the usability design, implementation or evaluation could be turned into a publication at an appropriate venue (e.g. a HcI conference).