Advice for MSc ASE Projects

Mel Ó Cinnéide

This is an informal note on what an MSc project in Advanced Software Engineering entails. In particular it covers aspects of research that ASE students may not be familiar with.

Essentials

The bare essentials of the ASE project are as follows. It is a 30-credit module (Comp40090) with no formal lectures. It is undertaken under the direct supervision of a member of academic staff in UCD and is intended to be equivalent to 3 months full-time work. Part-time students should submit their project in the late summer of their second year, but may be permitted to submit in December of that year for a further fee. The project involves the writing of a report or mini-thesis, usually in the range of 40-60 pages. Usually, though not always, the project also involves the development of some software.

1. Types of MSc ASE Project

There are different types of MSc ASE project. The list below is not intended to be complete, and your project may be a combination of types.

1.1 The Treatise

A treatise is a detailed, in-depth essay on a particular topic. Only a few projects have taken this format and it is not one we recommend. This is an engineering masters so we usually expect the project to involve building something or doing something. However, it is a possible approach if there is a topic you feel passionate about.

This type of project is like an extended literature review. No matter what type of project you do, it will *contain* a literature survey. See later for more detail.

1.2 The Empirical Project

An empirical project tests some idea you have (a hypothesis) against the real world of software engineering.

For example, you might be interested in exploring the hypothesis that refactoring improves code quality. You project might then involve taking a number of open source repositories, measuring the level of refactoring that took place and determining if this correlates with the quality of the software developed. You may well write software as part of this project, to measure the level of refactoring and measure software quality, but the purpose of the software is to help test if the hypothesis is true or not.

This is a very good structure for an ASE project. Much of the consensus best practice in the Software Engineering field has never been properly tested, and since you work in the field there may well be some idea you would like to explore.

1.3 The 'Big Idea' Project

Here you have some big idea (or small idea) that you would like to try out. Perhaps you have worked on databases for years and you have an idea for an improved sharding algorithm. This type of idea is a great basis for an ASE project, though you have to perform a careful literature survey to make sure your idea is new, and you need to have a plan early on for how you will evaluate your solution. Evaluation can be very challenging in this type of project. There is more advice on evaluation in Section 3.2.

1.4 The Technology Project

One feature of Software Engineering is how quickly the technology changes, and many projects involve exploring some aspect of new technology. For example, AI tooling has been having an impact on many aspects of software engineering, and a project investigating this would be interesting.

For example, one ASE project involved developing similar iPhone and Android apps for a particular domain and comparing the challenges faced in each area. Another studied the use of different programming languages in AWS Serverless Environments.

In this type of the project a significant amount of software is written and a working artefact is usually produced. The report is a reflective piece that evaluates what was produced and the lessons that can be learned from the experience. This is a popular model for an ASE project.

1.5 The Development Project

This is a development project that involves writing a piece of software that does X. On its own, this is just a piece of work to be done and is *not* suitable as a masters project. However, if there is a development project you would like to do, you can often make it suitable as an ASE project by adding an angle to it. For example, say you would like to write a refactoring tool for Java. There are many refactoring tools for Java and how to build them is well known. You could 'promote' this to a masters project by adding a little spice to the project, e.g.,

- Focus on little-known refactorings not provided in the mainstream refactoring tools
- Focus on refactorings for a particular domain, e.g. Android development
- Focus on refactorings most suitable for newbie Java programmers
- Use the project as a mechanism to explore an unusual software development process or as a way of evaluating a range of technologies that you might use in developing a refactoring tool.

2. Examples of ASE Projects

Sometimes projects are proposed by UCD staff. More usually you have an idea you are interested in, and contact an academic who works in that area. This idea could be based for example on something you saw in coursework in UCD, or some problem you encountered at work, or something that struck you while reading an online article. For a PhD, this idea has to be fundamentally novel; for your project it simply has to be novel in the sense that it has some twist or angle that differentiates it from what has been done before.

Here are some sample ideas from previous ASE projects:

1. *The dark side of patterns* [Treatise]: an analysis of patterns as they are used in industry, highlighting industrial issues with patterns that aren't apparent from pattern literature.

2. *Cloud Architecture Maturity Model* [Treatise]: Proposes that a Cloud Architecture Maturity Model is needed to augment the Cloud Maturity Model with specific details on architectural and technical techniques that are encountered at each level of the model.

3. *React Native vs Android - an energy consumption comparison* [Empirical]: Develop the same app for both platforms and measure their energy consumption profiles.

4. *Code smell detector for newbie programmers* [Empirical/Development]: code smell detectors are commonplace. This project explored if certain smells are more common among new programmers and focussed on detecting those.

5. *Refactoring Java contracts* ['Big Idea'/Development]: again, refactoring tools proliferate; this one addressed a new area for refactoring: contracts expressed in JML.

6. *Monorepo versus Polyrepo: An Empirical Investigation* [Empirical]: Are there any observable differences between monorepos and polyrepos, e.g. in terms of bug rate or code duplication?

7. *Recipe Nutrition Optimisation using Off-the-Shelf Ingredients* [Development]: Development of a novel approach to creating a nutritionally-optimal meal.

8. *Algorithm development for computation of structural hole connections model*: Attempts to efficiently compute the structural hole connections measure (a concept in social networks) without using adjacency matrices in the computation.

9. *Multi-document Text Summarisation of Politically Biased News Articles* [Empirical]: A study of how automatic text summarisation can be used to reduce the influence of media bias on readers.

3. What makes an ASE project different?

You are all university graduates with plenty of experience, so in what way is an ASE project different from the sort of work you do already? This section explains the main aspects of an ASE project that may be new for you.

3.1 Literature Survey

When you are carrying out a task at work, you do not first spend time reading papers from the ACM *Transactions on Software Engineering*[1]. However that is pretty much what you do on a research project. Your aims in doing this are:

- To master your field. If your thesis is in the refactoring area, you should become a master of the research and tools in the refactoring field.

[1] "TSE" is the top journal in Software Engineering. Try to read one relevant paper from it if possible.

- To demonstrate the novelty of what you are doing. The onus is on you to show that your project hasn't been done before, and the literature survey is how you do it. (This does not apply so much in a Technology or Development project.)

What will you be reading? Here are some categories in order of importance:

1. *Top journals*: TSE was mentioned above. There will be other journals in your area as well. This is the best of SE research and it can be hard work to read. It is not essential to research at this level for your project, but if there is a relevant paper there, do read it.

2. *Other journals*: There are plenty of other lesser journals. For example, *Software Practice and Experience* publishes papers that are more relevant to practitioners.

3. *Conferences*: Conferences vary from the leading *International Conference on Software Engineering* (ICSE), to more local conferences. Conference papers care usually more recent than journal papers (the review process for a journal paper may take years) and are usually easier to read.

4. *Workshops*: workshop papers are usually works in progress. Recent ones are most relevant. Workshop papers that are a few years old may be superseded by a more recent conference or journal paper.

5. *Online magazines, web pages, blogs, white papers*: These are relevant but are not peer-reviewed, and so carry little scientific weight. Read them and cite them, but do not let them be the bulk of your literature review. As a practitioner, you probably read a lot of this type of material, and it will be relevant to your ASE project as well, but make sure to read the peer-reviewed scientific literature as well.

In your literature review, you will cite the papers you have read. See *Report Writing Guidelines for BSc Students* on the ASE pages for advice on how to structure your review and how to cite correctly.

A few hints in this area:

- A recent *survey paper* is a great start to doing a literature review. It does a lot of the work for you.

- Cite at the highest level you can. Do not cite a workshop paper if the authors published a better conference paper (or, better still, a journal paper) on the same topic at a later stage.

- The literature review is ongoing during the project. There is no need to try to do it all upfront. However, if your project is based on an idea you claim to be novel, you do need to read around the field carefully to make sure that it hasn't been done before.

- MSc ASE projects vary a lot in emphasis. If yours is heavily focussed on development or technology, the literature survey may be correspondingly lighter and more focussed on less-academic articles.

- One of first things your graders will look at is your literature review. It will be a great start if they see a good number of journal papers, a number of conference papers, a few recent workshop papers and not too many web sites.

3.2 Evaluation

In software practice you evaluate software by testing its functionality, performance etc., and by testing what customers think of it in the marketplace. For an ASE project, you need to look at what the project is about in order to decide what type of evaluation is necessary.

In Empirical or 'Big Idea' project, your idea is expressed as a thesis, i.e., a falsifiable statement. For example, in Section 1.2 above, the thesis could be stated simply as "refactoring improves code quality." The goal of the project then is to test this to determine if it is true or false. You may think it is obvious, but in fact while refactoring is much talked about, there is little hard empirical evidence that it improves code quality. One way to test this is to take a number of open source projects that have undergone significant refactoring. By applying standard software quality metric suites to the code before and after refactoring you can see what (if any) improvement is achieved. This would require careful statistical analysis of the results to see if the null hypothesis ("refactoring has no impact on program quality") could be rejected. The point here is that even though you may develop software to detect refactorings in open source repositories as part of the project, the key part of the evaluation is to do with the thesis you are trying to test.

This thing to remember is this. Even if you are building a large software system as part of your project, you are not just building it to *do* something, but to *test* some idea. You can take all kinds of shortcuts, once the idea gets properly tested. Usability, performance, maintainability, flexibility etc. do not in general matter all that much when building a research prototype (but clarify this with your supervisor). Focus on building a piece of software that tests your idea.

Another perfectly suitable type of project is where you take something that hasn't been done before and do it, i.e., proof by construction. Technology and Development projects suit this type of evaluation and are a popular type of ASE project. Most of you are professional programmers and hence very competent at proof by construction.

If you are doing an empirical project, there may be little or no software to develop, it is all about constructing the experiment correctly. Your supervisor will guide you in this; Claes Wohlin et al. have written an excellent book on the topic [1].

3.3 Presenting your Results and Conclusions

This all depends on your project, so it is not possible to provide specific details. If you have quantitative data, you should probably use the appropriate statistical tests in your analysis. Your conclusions should be what is warranted by the data -- it is detrimental to your work to try to oversell your result.

Overselling is a very common stylistic error that ASE students make. Much of the literature in Software Engineering is actually Advocacy. We get used to reading claims like "The XYZ framework provides the smoothest user experience" etc. Making such a claim in a scientific paper without proper evidence to support it would not be acceptable, and adopting this type of pitch in your ASE project is to be avoided as well.

3.4 General Advice

These are general tips and advice from my experience of dealing with ASE projects.

- Masters theses are typically undertaken by fresh graduates. As you have industrial experience, you are in a much better position starting your project.
- Realise that scholarly research is probably different what you have done before, so be open to learning new ideas.
- Learn to write well. It is a vital part of research. There are books on this topic, but the simplest thing is to copy the style of the scholarly papers you read. Do not use blog-style writing or corporate-speak in your report.
- When writing, avoid referring to yourself ("I tested this…" etc.). It is not just a textual matter, it may show that your emphasis is incorrectly on your role in the process rather than on what was achieved[2]. Avoid overuse of the passive voice as well; prefer instead to use "we" even though you alone did the work.
- The temporal order of your project should have little or no impact on the order in which you write about things. Organise your report logically. Also, it may have taken you a long time to learn some new technology, but it may not be very relevant to the report.
- Develop the habit of being more critical of what you read. Appealing to Authority is not good science. As you read a paper, ask yourself "is this true? where is the evidence?" Much of what is written about Software Engineering is really advocacy, so learn to see through this.
- The most important parts of your report are: Abstract, Introduction, Conclusion and References. These are what get read first and make that vital initial impression.

4. Conclusion

This note describes what an ASE project entails and provides some guidelines and advice on how to approach your ASE project. Good luck with it!

References

[1] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimental Software Engineering -- An Introduction*, Kluwer Academic Publishers, ISBN 0-7923-8682-5, 2000.