Software Engineering Large Practical 2014/2015

Doctor Allan Clark
School of Informatics

Issued on: Wednesday 17th September, 2014

Introduction

The main requirement is to deliver a competition web site. However what the competition is for is largely left up to your own devising. The *Suggestions* sub-section of section 1 provides some plausible examples.

The practical is divided into two parts with their own associated deadlines. In part one, described in more detail in section 1, you will be expected to deliver a proposal for the web site. In part two, described in section 2, you are expected to deliver an implementation of your own proposal and an accompanying report.

The deadlines are:

- Part One: **Thursday 16th October, 2014 at 16:00**
- Part Two: **Thursday 18th December, 2014 at 16:00**

Officially part 1 carries no weight and 100% of your mark is awarded for your implementation and report submitted for part 2. However some of the mark for part 2 is associated with how well you have followed your proposal and/or adapted it where necessary. Hence without submitting anything for part 1, a small part of the mark for part 2 will be unavailable to you.

— ◇ —

After submission for the first part you will receive some feedback on your proposal. However, this feedback is to be directed by yourself. You are expected to *ask* for details of anything about which you are unsure. At a minimum however, your feedback will contain an assessment of the amount of work proposed. Whether or not there is enough complexity to the task and whether or not it seems feasible to be completed within the time.

— ◇ —

An important aspect of the practical is that you are exposed to more realistic elements of software development than is typically the case for the smaller coursework exercises you may have done to date. In particular you are faced with a set of requirements which must be satisfied, but intentionally those requirements are not necessarily complete.

The task is additionally intended to be a little larger and therefore requires that your code be more maintainable. There are still several aspects of software development which are not tested. In particular, this is an individual exercise, so there is no code sharing.

Developing a web site is a non-trivial undertaking but has a large scope for creativity. It generally involves the melding of several technologies together. This may well mean that you will have to *learn* some new technology or technologies that you are not currently familiar with.

There is also a greater amount of scope for you to make your own decisions about aspects of the practical. This is intended to prepare you to undertake your honours project next year.

— ◇ —

An important part of the practical will be to demonstrate that you have used appropriate software development tools and principles.

This means that:

- Your code should be maintainable, thus clean, reusable code is preferred to complicated and messy code which is difficult to read.
- You should provide appropriate documentation, in particular comments within the source code where necessary.
- You should make appropriate use of source code control.
- There should be evidence that the software has been tested, including *automated* tests.

Frequently asked questions

- *What programming language must I use?*
  - The choice of programming language is left largely up to the student. However, there are some restrictions. I must be able to evaluate your program with minimal setup, this means that it must compile and/or run, unmodified on a DiCE machine without any additional dependencies. Here is an obvious list of languages which should work on DiCE without any problems: C, C++, Python, Java, Haskell, C#, Objective-C, Ruby. However care should be taken with versions. Any other language may be possible, but you should email me first to check that it is appropriate and/or test it yourself on a DiCE machine. Note that it is possible to install DiCE on your own computer even within a virtual box instance `https://wiki.inf.ed.ac.uk/DICE/VirtualBox`.
- *Can I develop my application on my Windows/MAC/Linux laptop/desktop?*
  - Yes. Just make sure that it compiles and/or runs on DiCE as well
- *I'm more familiar with Mercurial/Subversion/Bazaar, can I use that instead?*
  - No. For this practical you need to use the `git` version control system. Whenever you join a team you will need to use their source code control system whether you are familiar with it or not. Unless you can convince the team to switch, but that is unlikely if you are unfamiliar with the current system.
- *Do I have to deploy my website to the world*
  - No. As long as I can run your webserver on a local machine and access it at `localhost` (usually `http://127.0.0.1:8080/`) with a DiCE browser that is fine.
- *I'm really unsure if my proposal will be challenging enough to attract a high mark*
  - That is what part 1 is for.
- *What if I do not make a submission for part 1*
  - As noted above, part 1 does not officially carry any weight. However a part of your mark for part 2 is awarded for how well you have followed and/or adapted your proposal. However, it is still possible to achieve a very high mark for the course without submitting for part 1.
- *Should my web server be ready for production*
  - We are looking for production *quality* code. However it need not be immediately deployable as long as your report demonstrates that the design is adaptable enough. For example you may use a database which does not allow concurrent updates such as SQLite, provided that your design does not rely on this and it could be easily substituted for a database that does allow concurrent updates.

Getting Started

The first thing to do is to initialise your source code repository. You can name your directory whatever you like, assuming you choose $\boxed{\textit{application}}$, you can start your project with:

```
$ mkdir application
$ cd application
$ git init
```

Part 1

Software Engineering Large Practical 2014/2015

Doctor Allan Clark
School of Informatics

Issued on: Wednesday 17th September, 2014

Deadline: Thursday 16th October, 2014 at 16:00

Introduction

One of the most challenging components of software engineering is estimating how long a given task will take. A related and similar question, is how much work can be done in a given amount of time.

The amount of time that you are expected to devote to this entire practical is 100 hours. We will suppose that this first part will take you 5 hours, and hence you have 95 hours to complete your implementation. Hence for this first part you are challenged with estimating a sensible workload for 95 hours.

This is a realistic situation. In real world software development tasks customers are unhappy if you propose to do too little, but equally are unhappy if you fail to meet your deadlines, whether they are self-imposed or not.

Deadline

The deadline for this practical exercise is

Thursday 16th October, 2014 at 16:00

Description

Your proposal is for a software development task that should take approximately 95 hours to complete. There are some constraints.

- You should develop a web application, this need not be deployed globally and can simply be accessed on the `localhost`.

- Your web application must run on DiCE and must be accessible via a browser available on DiCE.
- There should be some notion of a *user account*.
- Users, or some component of their accounts are ranked. What they are ranked upon is the main part of what you must expand for the first part. It could be a game, or linked to some real-world activity.

The ranking scheme is also left up to you to specify. You should specify what events trigger a re-ranking. If the competition is a web-based game one possibility is to calculate new ranks based soley on matches taking place. That is a completed match triggers the update of only the participants in that particular match. Another possibility is that rankings are re-computed regularly, such as once per day.

You should take care that your ranking scheme scales to a large number of users. For example, a ladder scheme is a great idea but will need to be adapted to be suitable for many users. A ladder scheme has some total ordering of the entities being ranked, an entity can challenge any entity above them. If they defeat that entity they are placed above them in the ladder. Usually there are some rules about rejecting a challenge. Such a ladder scheme works very well within local sports/games clubs with a small number of participants. For large numbers of users you may need some method of ignoring non-participating users or allowing new users to jump to an appropriate place in the ladder. Additionally you may want to motivate moving up the ladder even when very far from the top.

What to submit

Your proposal. This should be in a readable format such as HTML, Markdown, or PDF. The sources for this should be stored in a `git` repository. However, if your proposal format demands compilation (for example LaTeX) then please ensure that it compiles on DiCE and/or ensure that the most recent compiled version is in your submitted directory (note that the `submit` command submits a directory and so will submit files in that directory whether or not they are committed to your repository).

Please ensure that there is a `README` file in your directory. This should give details of where your proposal is. In addition this should contain some indication of *what* you would like feedback on. What aspects would you like me to comment on?

In addition to your proposal you can submit some code that you have already written and ask for some guidance on that. You can submit a high-level design description and ask for some feedback on that.

Be specific, the more specific your questions are, the more useful the feedback you receive is likely to be. If you submit some code and ask “What do you think so far?” I may well comment on your formatting style, your choice of language or the names of your files, because I simply won’t know what you are asking.

How to submit

First, get your proposal on to the School of Informatics DiCE computer system.

If your project is in a folder called *`Application`* then you should submit it for the `selp` part `1` using the command:

```
submit selp 1 Application
```

Suggestions

In this section I make a few possible vague suggestions to inspire your own proposal.

Simultaneous Board Games

Turn based board games work well on the web, for example Chess and Go. Both could be played in real-time or turn-based in which you make a move and then wait until your opponent makes a move without requiring that you are both connnected at the same time or for the duration of the game.

Since you and your opponent are not in the same room, it is possible to modify such board games such that both players make their moves without seeing how their opponent has moved. In such a scenario the moves would be played logically simulataneously. For most games the rules would need to be updated, for example what happens if both players move to the same square?

Another idea is to play multiple games *with the same move.* Hence such a player needs to make a move which is useful (or at least not disastrous) for both games. In this way, one could implement three-way chess, or three-way rock-paper-scissors-lizard-spock.

Real World Trackers

A site for tracking real world plans. Such as an exercise plan, in which users are given points for maintaining their exercise plans. Of course one would likely have to rely on user honesty to actually evaluate whether or not such a plan had been stuck to.

One could imagine this for a more objective criteria such as quitting smoking, or a real-world activity with objective scores such as golf, visiting cities/countries, keeping a diary/blog active.

Frequently Asked Questions and Clarification

- *If I don't complete part 1 before the deadline can I modify that part in the time before the deadline for part 2*
  - Yes. However, you cannot expect to receive feedback about your proposal if it is not submitted as part of part-1. Secondly, you may receive marks in the second part for how well you have stuck to your plan *or* how well you have deviated from your plan for good reasons. Obviously you cannot receive such credit if you have not submitted for the first part.
- *What if I choose not to submit for part 1*
  - Then you will have to assume that you have a sensible plan and that your proposal constitutes neither too much nor too little work. That is a risk you take if you choose not to submit for part 1. In addition, as noted above, some of your mark for part 2 is dependent upon having submitted something for part 1.
- *Can I submit any code that I have written so far?*
  - Yes. That is a great idea. Remember though to be specific about what you want feedback on.
- *I have a really great idea, but I think it is too simplistic. What should I do?*

– Be creative. I think most proposals can be fleshed out in similar ways to the above. Consider multi-way competitions rather than one-on-one. Consider other new rules. Consider voting among groups of users. Consider making the site more social with the ability to form discussion groups.

Part 2

Software Engineering Large Practical 2014/2015

Doctor Allan Clark
School of Informatics

Issued on: Wednesday 17th September, 2014

Deadline: Thursday 18th December, 2014 at 16:00

Requirements

You are expected to deliver a working website, which implements your own proposal. The web site need not be deployed and open to the outside world but can simply be deployed on the local host. Essentially I need to be able to test your solution on my machine.

The expected minimum features are:

- A working web site.
- Implementation of whatever the competition you are ranking is.
- User accounts with associated rankings, using an appropriate ranking scheme.

Minimum feature set here means that for an appropriate proposal a perfect solution with a good report can expect to attract full or near full marks.

You are expected to deliver a prototype of a working website. This means that features can be left unimplemented provided you document a reasonable strategy. For example you may not have full authentication for users, but provided your current form of authentication and representation of users can be modified without major architectural changes, this can be documented in your report.

The extent to which leaving features unimplemented is justified will depend largely upon how ambitious your proposal is. Some indication of this will be included in the feedback to your submission to part 1.

What to Submit

In this section I detail what the directory that you submit should contain. The main three things are the `README` file, the `git` repository and your report.

The `submit` command submits a *directory*, this can also be the root of your *repository* but your repository could also be a sub-directory if you wish. Either way, the directory you submit should include the `README` file at the top level. The `README` file should detail where the repository and report are.

I see no reason for you not to track the `README` and the (source of the) report in the `git` repository, but this is not a requirement.

Repository

The repository should track the source code and any documentation for your website. As stated above I recommend tracking your `README` and report in the source code repository.

If you are producing your report mechanically from some other source, for example you are producing a PDF report from LaTeX sources, then please make sure a compiled report is included in the directory. In this case you would likely not wish to commit the compiled report to your repository.

README

The README should at a minimum provide instructions for locally deploying and evaluating your web site. If you have a good automated test suite instructions on how to run this should be provided. If I cannot deploy your code solely by reading your `README` file your submission will lose marks. Be explicit. Do not describe how to run the application in words, demonstrate it with actual commands. Preferably this should consist of exactly *one* command.

Some navigation of your source code. Give some indication of how the source code is structured and in particular where the tests are. This could be a line that directs the reader to the report if the structure of your code is detailed well there.

In addition, your README should contain information about what you would like feedback on. I aim to provide useful feedback, but the feedback is to be solicited by the person submitting. This requires you to think a little about what you would like to learn.

Report

The report is indended to be a short document of around 2-5 pages. It is your opportunity to highlight things which you have done of which you are especially proud and explain things which are not perfect. I do not expect perfect software, the report is the place to explain what you would do with more time and resources.

In addition it represents an opportunity to describe mistakes you have made, or difficulties you have had. A poor design will lose marks, but it will lose fewer marks if you explain that you now understand why the design is poor, and/or why you made that decision in the first place. Simply recognising a short-coming of your design or approach at least means you are demonstrating critical evaluation of your own work.

Producing an application with a poor design, structure or process is unfortunate, failing to recognise that you have done so, is often disastrous. The report is the place to demonstrate that you have recognised this.

In addition, good design is often subjective. I may disagree with a design choice of yours but if you justify it - *even if I disagree with the justification* - this demonstrates that you considered alternatives.

In a similar manner, good design and principles are rewarded but it is often unclear how much intent or luck was involved. Good design by intent is rewarded more than good design by chance. The report is the place to make sure your intent is recognised.

I would expect a reasonable report to include, but not be limited to, the following:

- A justification for design choices, ranging from the language(s) you used to the architecture of your application
- A description of your design
- A description of your testing strategy
- Any deficiences in the functionality
- Any deficiences in the implementation/design
- Make sure that I know of any non-obvious functionality
  - Try not to have user-functionality that is non-obvious. But you may have, for example, a good backup or security strategy.
- Things you would do differently
- Things you are especially proud of

Your report can be in, Markdown, HTML or PDF. Any other format is likely possible but please ask me first well in advance.

Development Requirements

In this section the requirements for the development practices that you should follow are discussed.

Source Code Control

It is important for large software projects that they are well maintained. Part of this is the use of a good source code control mechanism. For this project we will use the `git` source code control system.

Your final submission will be partially judged by how well you have managed your source code using `git`. Mostly this will be based on how appropriate your commits are. Although it is subjective, the general rule is that there should be a single commit for each “unit of work”. Two good rules of thumb are:

- If you cannot describe the work done in a single sentence without using the word “and” you probably have more than one commit’s worth of work
- Could someone conceivably wish to revert some of the changes in the commit without reverting all of them? In this case you again probably have more than one commit’s worth of work.

Importantly these are just rules of thumb. For example you might break the first by stating a few pieces of fixing formatting and spelling errors. The following commit message may well represent a reasonable commit “Corrected some spelling mistakes in the comments and removed some trailing white space”.

It is also possible to commit too often, although this is pretty rare. Generally this will not be penalised, unless the student is clearly abusing the source code control mechanism simply to get around using it properly, for example automating a commit whenever a file is saved.

Github, google code and others

Many of you will know of source code sharing sites such as github[1], google code[2] and bit bucket[3].

These are a great resource, however please refrain from using them for this project. This is an individual practical hence we do not permit the sharing of code between students. If your code is publicly available one of your classmates may use it thus you may be penalised as having shared code. Some of these sites allow private hosting which you can of course make use of if you wish, but it is not required.

Many students wish to make some of the source code they have developed publicly available. After *all* students have submitted you are welcome to post your source code however publicly you like. Including making use of the above mentioned source code sharing sites. The only requirement is that you do not do this whilst the practical is still running. Do not forget that some students may be allowed an extension to the deadline for quite reasonable reasons. You can assume it is okay to publish your source code when you receive your marks and feedback. If you desparately wish to do so before then, please contact me.

Testing

Your software should be suitably tested. This of course depends upon your proposal. If you would like some guidelines for this please indicate this in your submission to part 1. Generally though I would expect some form of automated tests.

For example your ranking algorithm should be tested. This should be quite simple to separate the ranking functionality from the rest of your code. It should also be easy to come up with some example inputs to the ranking algorithm and know what the expected output should be.

Early submission credit

In software development, timing and delivery of completed applications and projects can be crucial in gaining a commercial or strategic advantage over competitors. Being the first to market means that your product has the opportunity to become known and similar products which come later may struggle to make the same impact, simply because they became available second.

In order to motivate good project management, planning, and efficient software development, the SELP reserves marks above 90% for work which is submitted **early** (specifically, one week before the deadline for Part 2). To achieve a mark above 90%, a practical submission must be excellent in all technical and functional aspects, correctly implement the your proposal, be

[1] `www.github.com`
[2] `code.google.com`
[3] `www.bitbucket.com`

implemented in idiomatic code, and otherwise meet all requirements of the practical, **and** in addition to this it must be submitted early.

Work submitted less than a week before the deadline does not qualify as an early submission, and the mark for this work will be capped at 90%. Thus, the mark may be 90%, but it may not be higher than this.

Regardless of when it is submitted, every submission is assessed in exactly the same way, but submissions which attract a mark of above 90% which were not submitted early have this mark brought down to 90%.

To be clear, submitting before the early submission deadline does not guarantee that your mark will be over 90%, but **not** submitting before the early submission deadline guarantees that your mark will **not** be over 90%.

2.1 Assessment

Part two of the SELP is worth 100% of the marks for the course. Your mark will be expressed as a percentage given as an integer between 0 and 100. Thus it is not possible to score more than 100% and it is not possible to score less than zero.

Assessment is based on the following criteria:

- Use of source code control
- Documentation, including source comments
- Testing
- Maintainable code
- Your report
- Early submission

Assessment criteria

This practical exercise will be assessed in terms of the completeness, correctness and efficiency of your implementation of your proposal. The quality of code using language features well.

- Your application should be appropriately structured

  Good separation of concerns and appropriate use of the correct technologies will gain marks

- Your source code should be clear and easy for an experienced programmer to understand.
  - Writing idiomatic code in your choice of language will gain marks.
- Additionally, all else being equal, an application whose code contains examples of poor programming style (such as unused variables, dead code, blocks of commented-out code etc) should expect to attract fewer marks than an application which does not have these problems.
  - Poor programming style will lose marks.
- The content of your report

  - Critical evaluation (both the good and the bad) of your own work will gain marks

Additionally, if you have managed some form of clever optimisation but have not documented (for example as a code comment) the marker may mistake it for sloppy code style (which, arguably, without any associated documentation, it is).

2.2 How to submit

First, get your code on the School of Informatics DiCE computer system, copying it from your own laptop as necessary. If your project is in a folder called `Application` then you should submit it for the `selp` part 2 using the command:

```
submit selp 2 Application
```

2.3 Deadline

The deadline for this practical exercise is

Thursday 18th December, 2014 at 16:00

2.4 Marking

Your submitted coursework will be marked within four (semester) weeks of submission. (This is the period of time approved by the School of Informatics for large practical classes.) That is, marked work should be returned to you by Thursday of week 4 of the second semester. For 2013/2014 this means Thursday 5th of February, 2015.

We will try to meet this deadline but – for good reasons – students sometimes get deadline extensions meaning that their submitted coursework is not available to mark starting at Thursday 18th December, 2014 at 16:00. This can delay the return of marked coursework.

2.5 Frequently Asked Questions and Clarification

- *Should I put comments in my code?*
  - Yes. All else being equal, a submission with comments will attract more marks than one without comments.
- *Will I be graded on the aesthetics of my application*
  - No. This is not a graphic design course. However, *whether* you consider your application to be pleasing to the eye or not, your application's look should be easily changeable. In particular hard-coded aesthetics may lose marks.
- *How will this practical be marked?*
  - Marking is necessarily somewhat fluid. That is the price to be paid for a flexible practical. In addition this will prepare you somewhat for your honours project next year. That said, submissions for the Software Engineering Large Practical are evaluated against your own specifications in this way:

1. The accompanying documentation is read for instructions on how to use start the web server on the localhost

   – Submissions with insufficient documentation will lose marks here.
2. The user instructions if required are read and an attempt to create an account and take part in the competition is made

   – Submissions where this is not easy or possible will lose marks here
3. Correctness will be evaluated

   – Submissions which incorrectly implement the object of the competition will lose marks here

   – Submissions which incorrectly implement the ranking will lose marks here
4. Other additional features of the submission will be explored

   – Submissions with useful additional features will gain marks here
5. The source code will be inspected for good programming style

   – Submissions with insufficient structure will lose marks here

   – Submissions which do not use language features well will lose marks here
6. The architecture and design of your application will be inspected with particular respect to that which is mentioned in your report

   – Submissions which are inflexible to change or enhancement will lose marks here
7. The source code log of development will be inspected

   – Insufficient use, for example few large commits, will lose marks here
8. Finally the report is read

   – Submitted reports which explain what has gone wrong can reduce the marks lost for any of the failings described above

   – Submitted reports can garner extra marks for highlighting things that have worked well in hindsight, particularly if this is well justified or reasoned above

   – Finally a good report *may* make some broader point using your development as an example. For instance you may wish to advise for/against the use of a particular technology in a particular scenario