COMP700 Resources: Managing your Project
¡ draft v0.3 !

C. Maria Keet
email: keet@ukzn.ac.za, home: http://www.meteck.org
School of Computer Science, University of KwaZulu-Natal

May 8, 2011

Contents

1 Introduction

Doing your 4th-year honours project (thesis) is quite different from the assignments and exams your are familiar with: it spans a longer period and there are no intermediate official checks to test your progress and force you to keep going. You will have to rely on your own motivation and planning of your work to complete it. To be sure, you will have one or two supervisors to guide you with your project, but they are not there to do the work for you. In addition, the amount of work that is expected from you cannot be done the evening before it, like cramming for an exam session, but should be spread out over the whole period allotted to the project. This will require some planning of the activities you will carry out for the thesis so as to stay on track and stick to the commitment you made to yourself. This document provides some

suggestions and guidelines how to plan and manage your activities that will help you bring it to successful completion (without stressing too much).

2 The kind of activities

The minimum 'milestones' for your honours project are the presentation of your initial project proposal and the 2-3 pages you will submit, the honours mini-conference day with a short write-up and a presentation of your work, and the final thesis. For a MSc and PhD thesis, these may include an additional year report and presentation, and/or 'entry exam', and/or publications of your results as requirements to graduate. But what should you do for it, and how to go about doing it? This section helps you identifying the subtasks and the next section provides suggestions how to structure them.

2.1 Types of theses

The kind of activities will vary depending on the nature of your project. In the following, I speak for CS as discipline only.

You should first consult the "research methods" resources about 'what is CS' [2] and 'doing research in CS' [5] (and, optionally, 'paradigms in CS' [3]). In a nutshell, there are two principally different thesis (be they honours, MSc or PhD theses), each with sub-variants:

1. Theory
   (a) Novel theoretical advance (frameworks, theorems, proofs)
   (b) Predominantly literature review (Hons/MSc only)
2. System
   (a) Proof-of-concept implementation
   (b) Application through full design cycle (Hons only, though occasionally seen at MSc-level)

From a *CS research* perspective, the pecking order of importance and coolness (and perceived difficulty) is: 1(a) > 2(a) > 1(b) > 2(b), though in countries like the UK and USA, which have a more technology-oriented culture, they probably would switch the position of 1(a) and 2(a) and let others (Europe, mainly) do the theory development for them. From a perspective of someone who aims to get a good job in the *IT industry*, the pecking order of usefulness is probably 2(a) > 2(b) > 1(a) > 1(b).

They require different competencies and, in part, different activities, but there are also commonalities, which we go through in the next sections.

2.2 Core activities

There are five main phases to any thesis, being:

1. Exploration
2. Problem definition
3. Proposal
4. Body of research/development, including generating a written paper trail
5. Complete the production of the thesis and the defense

I will comment on them briefly in the following subsections.

2.2.1 Exploration

The principal purpose of the exploration phase is to become acquainted with the area you have chosen. This you do by searching for related work and reading up on background material. Your supervisor likely will give you some initial hints where to look, which provides an easy start for looking up other resources: the papers in the 'references' section of that material and, when you go to Google scholar (`http://scholar.google.com`) and search for the paper, it will return that paper and a "cited by" link. When you click on that link, you will see who used the paper for something else (which may well be something you wanted to do but someone else already did). Some journals and publisher already provide you with a "This paper has been cited by" section on the official page where the paper was published, saving you a Google search (those lists are also partial, however). Also, searching with related or slightly broader terms can generate interesting results.

2.2.2 Problem definition

Having read up on your topic, and, more importantly, having *analysed critically* the papers you have read, you may have observed gaps or limitations. Or you had a rough idea of the problem already, but you have not been able to find any resource that proposes a satisfactory solution. These observations help you to come to a precise definition of the problem that has to be solved or the formulation of a hypothesis to be falsified. Understanding the problem(s) is not as easy as it sounds, nor is formulating it properly. For instance, the following are *not* problem statements:

1. Need to optimise the algorithm for the travelling salesman problem.
2. How to support the user in selecting the appropriate part-whole relation.

The first one is not, because it indicates a *desire* to achieve something (optimization), and the second one is not, because it looks at doing something (holding his hand as 'support'?) for no particular reason. What, in theory, could be the real problems underlying those two are (mildly exaggerated and simplified):

1. Real-time applications require calculation of the solution to the travelling salesman problem within one minute, but all the extant algorithms x, y, and z take umpteen minutes to compute the answer, hence there is no technology that meets the performance requirement demanded by the client.
2. Both experienced and novice modellers do not know which part-whole relation is appropriate between classes in their domain ontologies and they often choose one arbitrarily, which causes an incorrect representation of the knowledge with respect to reality it aims to represent and leads to wrong deductions with respect to the subject domain semantics. (This, in turn, leads to erroneous suggestions to users of, e.g., decision support systems in medicine, therewith potentially threatening a patient's life.)

Problems are things that can be solved—maybe not by you, but that is a separate topic—and can be shown to be solved (e.g., a correct proof demonstrating P = NP), and research questions can be answered. Here's a typical example of problematic questions of the form "Can X be solved by Y?", e.g., "Can one use theorem prover ABC to efficiently draw inferences in higher order logics?" or "Can one use algorithm ABC to efficiently compute the solution to the travelling salesman problem?". It fails if the answer is 'no' and it even fails if the answer is 'yes' but the encoding has not been optimised (hence, less efficient), and when is something 'efficient', compared to what? In addition, while focussing only on ABC might be sufficient for a BSc project (depending on the difficulty of ABC, if it has been implemented before etc.), it is too narrow for a MSc/PhD topic.

A sensible research question for your project could be "is ABC faster than DEF in computing the answer to the travelling salesman problem?" (and if so: why; if not: why not?),

which can be easily reformulated into a hypothesis ("ABC is faster than DEF in computing the answer to the travelling salesman problem") or into a problem ("it is unknown if ABC faster than DEF in computing the answer to travelling salesman problem"—although then one has to include in the problem statement that someone cares to know, and why).

The indications provided in this section are not exhaustive, but hopefully gives you at least a basic notion. To practice formulating a problem or research question, you could first have a look at how others did it by, e.g., analysing the papers you read, finding the formulation of the problem/question (it should be written somewhere in the introduction), or else extracting it from the text.

2.2.3 Proposal

With the problem definition in hand, you can start setting out how a solution should look like, and how that is to be done. This is formulated into a thesis proposal. The purpose of a proposal is to clarify the goal of your thesis, it outlines possible steps to carry it out, it gives you a yardstick to measure your achievements, and it is a first 'seed text' for the thesis.

It is an important document, because if your proposal is good, you may well 'fly through' your thesis and obtain a good mark. On the other hand, if the proposal is bad, you may embark a journey on the wrong road. When is a proposal *not* good? There can be many reasons for that, among others:

- You did not search good enough, and the problem has been solved already;
- The problem is too big to tackle in the short time available to complete your thesis;
- The problem is too small and narrow for a thesis;
- There is no clearly measurable outcome planned;
- Nobody cares about solving that problem;
- Interesting and hot topic, but there are many well-funded research groups working on it already (you are most likely going to be scooped and/or they will outperform you).

What makes for a good proposal? This is not simply the opposite of a bad proposal. Here are several parameters:

- The solution/answer should serve some purpose;
- Results will be concrete;
- Permits theoretical and practical approaches;
- Feasible within the allotted time and available resources;
- There is faculty competent in the topic who can guide you (or they know someone who is and can).

You are expected to present your proposal to the faculty before embarking on the core tasks of the project, precisely to avoid the former and aim for a good proposal.

2.2.4 Core research/development

The tasks to carry out, i.e., the item "Body of research/development, including generating a written paper trail" listed above, differ along the type of thesis. For theory, the outcome is always—and only!—in writing, for systems development, it is both in writing and well-documented software.

Moreover, software development has a clear design procedure, which should be reflected in the activities you plan. For proof-of-concept software, the (high-level) tasks would be something along the line of:

- More theory & literature research
- Design of the algorithm
- Program the algorithm
- Test and evaluate the algorithm

On the other hand, tasks for straight-forward application development in, say, the waterfall methodology would be:

- Requirements analysis
- Design conceptual model and logical model
- Program the implementation
- Test it
- Evaluate with users

Theory-oriented theses may seem somewhat more flexible than system-oriented, but are not. The tasks look like:

- More theory & literature research,
- Design framework/method
- Evaluate framework/method
- Compare your solution with related works

or:

- More theory & literature research
- Design language/algorithm/modelling solution
- Prove correctness of encoding/complexity
- Compare your solution with related works

The most important difference, however, is that with theory-oriented theses, *writing is doing research*, i.e., it is an *integral part* of the tasks, whereas this is not the case with system-oriented thesis. Sure, one has to generate a paper trail with the latter type, but when, say, developing the conceptual data model for the application, the model is the real outcome of that task, not its explanatory text.

2.2.5 Complete the production of the thesis and defense

You are expected to provide a written and oral account of your project work. There is separate documentation that describes the structure of reports and writing style (e.g., [1, 4, 6]), so I will not go into those details here.

While a presentation takes only about a week to prepare, writing it clearly and coherently takes a lot longer—and always longer than you think it will take. Here at UKZN, you are expected to write *two* different documents: one shorter 'article-like' document (about 4 pages, double column) that exhibits the core of your work, and one longer document with the details, i.e., the thesis. Upfront, it may seem easier to first write the former and then the latter, but the quality of the former will be better when you first write the latter and 'slim it' to fit into 4 pages, because having the overview and the details allows you to generate a more accurate core.

Needless to say, each of the above-mentioned tasks can be broken down into sub-tasks, down to the point where initially the major task "do an honours thesis" has been broken down into manageable activities that are doable in the day-to-day work, one step at a time toward that end-goal.

3 How to structure your activities

The previous section already indicated some sequence in the activities, but it did not give any practical hints on how to do that for your project work. The first you have to determine are your *milestones* and *deliverables*, then you break up the major activities into smaller tasks, and based on that, you can make a schedule for your activities.

3.1 Milestones and deliverables

Deliverables are the *concrete things* you have to deliver to someone before or at a *specified deadline*. For instance, your written proposal and your thesis are deliverables, and perhaps your supervisor asks you to hand in a first full draft of the thesis by a certain date. They are requirements and constraints that have to be met in order to have a chance to pass the course.

A milestone is a *significant* moment of time with a certain *accomplishment* of a task in the project. For instance, the proof of a theorem or the first version of the implemented algorithm. They help you to be pleased/satisfied/proud of the progress you have made, and they will keep you focussed on the important tasks to do.

You should include both in you schedule.

3.2 Dividing up the tasks

We now have a list of the general, high-level, tasks that have to be carried out, and some basics regarding the deliverables and milestones that have to be included. They do not make for a schedule yet. What is missing is a further break-down into subtasks.

For instance, take "literature research". This entails:

- sourcing the literature
- reading it
- analysing it (summarise, pros and cons of their proposal)
- follow up on their references
- making a bibliography, and
- writing a "State of the art" section in your thesis that synthesises the relevant literature.

Task planning does not have to go into extreme detail. For instance, for "making a bibliography", planning very fine-grained subtasks like 'explore reference management software', 'read the bibtex manual' is a bit over the top (in my opinion); you can note those in your project log (see Section 4.2) that you did them as part of that task, though.

In addition to the subdivision of the larger tasks, you will decide where your actual milestones will be. For instance, you are developing an application for Needy Client Corporation and, depending on the methodology you use, part of the usual software development cycle may be the identified task to "develop a prototype". Subtasks can be identified; e.g.:

- determining what features should function in the first prototype
- the programming itself
- the initial testing
- the presentation of the prototype to the employees of Needy Client Corporation, and
- processing their feedback

A milestone in this list of tasks could be the penultimate one: it is your first presentable thing of your thesis work to the outside world. Observe that here I had not chosen the last event in the list: clients complain anyway and once they see a prototype or mock-up, they tend to generate new requirements and wishes that they want you to take into account, which might sound discouraging initially. Just that they want more, does not imply that what you did is insignificant. Milestones for theory-oriented activities could be, e.g., the proof, the algorithm etc.

3.3 Introducing breaks

Let's be honest to yourself: will you really work on the thesis each day from now until late November? If you plan to do so, and you skip even a single day, you may feel 'guilty' for slacking. If you plan (and do), say, on average 30 minutes more per day for two weeks, you have a well-deserved day off! Or, perhaps, there is a major family event some time in the upcoming months: there is no shame in integrating that into your schedule—in fact, it is

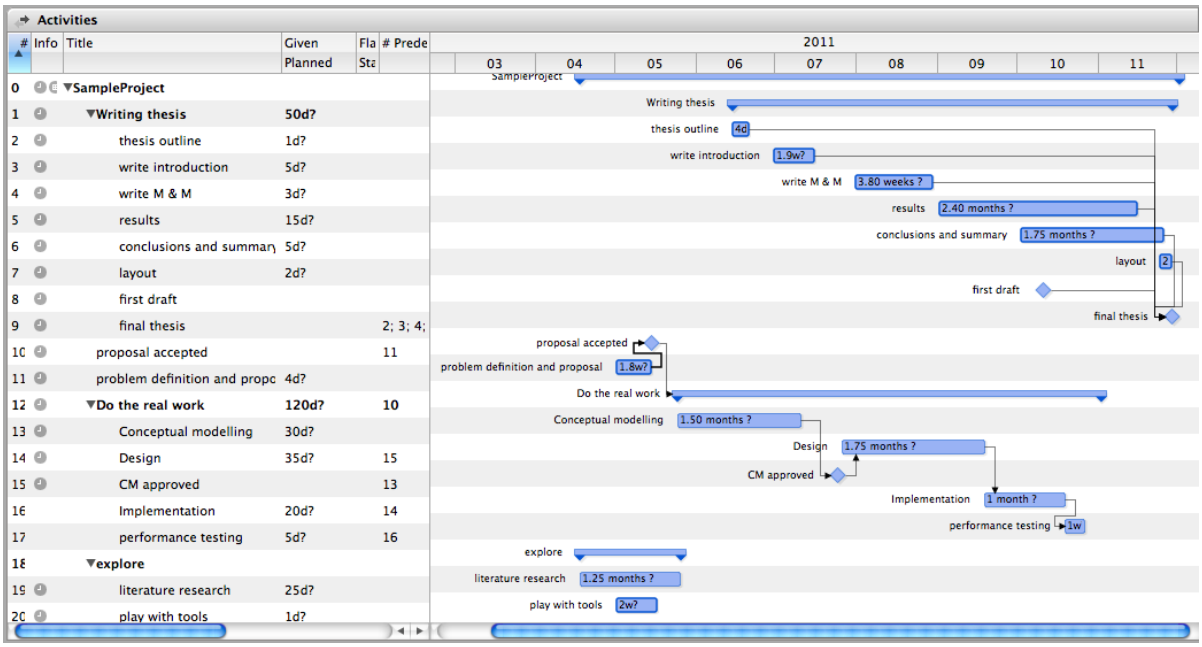

Figure 1: An incomplete Gantt chart (made with Merlin).

better to do so, so that you can enjoy the event, instead of beating yourself up for not working on the thesis and getting behind in the planning. Planning breaks is a bit like fooling yourself, but it works wonders. So, yes, you can plan breaks into you schedule; I would even say that you are encouraged to do it.

Why is that so? What you have to take into account is that the 'problem' with research and development projects is that you always can do more. This is in stark contrast with a normal course module where there is a clearly delimited piece of knowledge and skills you have to acquire, and once you know it, it's enough. For theses, there is no upper bound, really. As a consequence, you may end up working all the time on it. However, occasionally taking a step back, taking a break and coming back with a fresh mind and energy helps you see things in a new light, gets you over a lapse in motivation, or makes 'writer's block' vanish. Or you can use it as a carrot for yourself to get a less pleasant task done more quickly ('if I complete x, I can go to the beach for a whole weekend'), and actually enjoy the time off.

3.4 Making a schedule

We are now ready to start making a schedule, using the deliverables, milestones, main and subtasks. There are strict deadlines for the presentations and for the deliverables you have to hand in, but the rest is up to you to decide (in collaboration with your supervisor). In theory, there are two ways to put actual dates to the activities: calculating forward, and calculating backwards—the end product should be the same schedule anyway.

While it is possible to put your schedule in a MS Word or Excel sheet, this is rather inflexible to manage, because with every little change (say, you were ill for a week), you will have to update all of it manually. There is not only a plethora of software for scheduling, there are also different scheduling systems. PERT (program evaluation and review techniques) and Gantt are the most well-known ones. Figure 1 shows an (incomplete) Gantt chart. The key elements of such a chart are:

- Break-down of activities and sub-activities (on the left)
- Milestones (noted on the left, visualised with a blue diamond on the right)
- The estimated workdays for each (sub-)activity (4th column in the left-hand pane)

- The assignment of start and end dates together with the estimated workdays (which generates the blue rectangles)
- The dependencies between the activities ("Precedes" in the left pane, visualised as lines connecting diamond and/or rectangles in the right-hand pane)

The software signals any inconsistencies. For instance, if you were to have included a thesis submission deadline of Dec 1, 2011 and it being the end of the whole project, but subsequently you add an end date for, say, performance testing, to be Dec, 5, then it will flag that your schedule won't work. Or when you add an activity write conclusions and abstract that you think will take you 5 days (1 day = 8 hours) to complete and set the start date Nov 24, 2011 but your predefined actual days in the week have only 4 hours, it will tell you that you go over the project deadline and are supposed to fiddle with the dates and workdays until it fits with its internal constraints.

Hence, the 'forward' calculation means adding your activities sequentially, adding estimated workdays, estimated begin and end date. It will tell you soon enough if you have time left (perhaps you can add an activity?) or short on time—even before you have really started. The 'backwards' calculation is in that respect more dangerous for your project: you may end up with an initial schedule where either you should have started a few weeks ago and are behind already or it gives the false impression you can slack until August.

If you cannot make a schedule that fits with a realistic amount of estimated workdays for each activity, you have a problem. There can be several reasons for it. Look at it from the bright side: it is better to know upfront that your informal intuitive planning does not work so it can be amended early on instead of getting totally stuck or overworked later.

4 Avoiding and dealing with delays

First, and foremost, to be able to deal with a delay, you need to make yourself aware that a delay exists, or when a delay is, or can be, avoided. Keeping a project log and taking the 'onion approach' are two mechanisms for this, which will be introduced in the next two sections. But despite all things, you still may incur a delay for some reason or other; suggestions how to cope with that are described in Section 4.3.

4.1 The onion approach

A mechanism to keep you focussed is to use the so-called *onion approach*: identify a core of essentials that really must be part of your thesis, a list of important components, and the nice-to-have ones. This helps to ensure you at least will have you core results by the deadline, and if there is time remaining, you can gradually add extra layers to the core result.

For instance, your thesis revolves around the novel implementation of some algorithm in a fancy way. Having it implemented is a task, but so are testing on performance, developing a nice user interface, and it needs a user manual. Obviously, the implementation of the algorithm itself is a core task, and then the second or the third one, and writing a manual. Two fictitious examples are shown in Figure 2. How does your 'onion' look like? Does it match with your (initial) schedule? Did you set the right priorities from the start? Perhaps you have to revise your schedule. For instance, you may have an order of tasks a, b, c, d, e, and f in the initial schedule, because you know upfront you like doing b more than f, but in the logical order of things, f just have to be done at least before c, and possibly also before b.

Reading this information at the very beginning of the project duration enables (or would have enabled) you to think about prioritisation of your activities first, and devise a sensible planning based on that. Monitoring your schedule, then, can be done through keeping a project log.

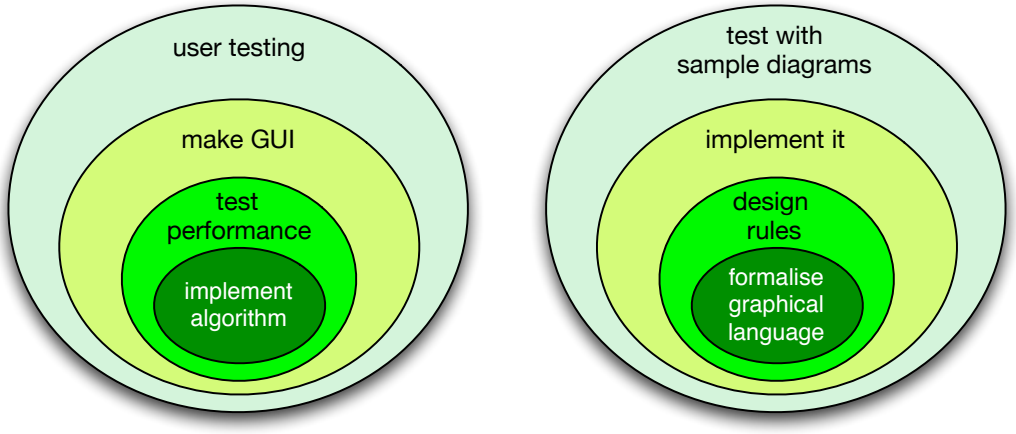

Figure 2: Two fictitious examples of how an 'onion' of tasks may look like for a thesis.

4.2 Keeping a project log

A project log is a document where you summarise the activities and ideas relevant to the project you did that day, and the time spent on it. Although it may seem to bit of a tedious task to do at the beginning, it is useful for several reasons as you go along:

- You can see for yourself whether you really spend a reasonable amount of time on the thesis over the weeks and months.
- You can cross-check it with your planning: are you working on the tasks you set out to do or being side-tracked with other activities?
- You have a single location where you can write down ideas you may have to follow up, which are otherwise easily forgotten about a few months later.
- If you need to 'backtrack' from a dead-end in the development, you know where to backtrack to.
- It provides a trail with which you can demonstrate that it really is your work.
- It helps you evaluating your work (the progress in the development of ideas, software, and thesis writing) and reflect on you work habits: were they effective? (if yes, why, if not: why not?) Would you do it the same way if you had to do it again?

Assuming you have an outline of the activities and made some schedule (see Section 3), then you can format that into a project log by simply adding a 'comments' and/or 'progress' column and a 'actual time spent' column. An example is shown in Figure 3. For instance, it states that in the week of May 10, the statement of requirements is *scheduled* to be completed and, indeed, this was the case ("**11-5:** finished revised SoR"). It also contains text like "**17-6:** See lab notes on testing today. 3.5h", which indicates there is a clear distinction between the kind of things you did (testing) and the technical issues encountered (documented in the lab notes): the project log is for recording the former information. It also contains some tasks and ideas to (possibly) follow up on ("**4-9:** added table 3.4 on design features. Re-read/correct this later" and "**30-8:** or maybe add an appendix on a brief explanation of ORM?", respectively).

4.3 Review and revise

Despite good planning, you may not keep up with your planning. The way to deal with the delay depends on what causes it. There can be many reasons for delays, e.g.:

1. You underestimated the problem you aimed to solve because
   (a) It is harder than you thought and doing task x takes more time,

Week	Tasks	Comments	Progress
May 10	1. Statement of requirements 2. Introduction chapter report	Intro chapter with title, aim and description. SoR and the relation diagram confined to Appendices.	**10-5:** made the required changes to the ER model, entered meeting answers in doc file and partially adjusted the SoR to reflect yesterday's meeting. 4h **11-5:** finished revised SoR, started filling some of the appendices for TMA02, added more detail to the schedule. 1h **15-5:** working on the 'appendix' (diagrammatic representations + screenshots of ER, ORM and LogModel). Still need to write out the logical schema with the table definitions. 1h **21-5:** written the logical model, i.e. the domain and table SQL statements. 6h *week 13 total: 12h*
....			
June 14	1. First draft theory section 2. Start DB implementation	Should have an empty database by then. Start to consolidate info on theory chapter, include discussion on DB modelling	**14-6:** reminded Brian of my question about references and literature. Made first version of the database, called bact2.db. 4h. Notes after the table **15-6:** added the notes from yesterday's work. 1hr. Need to try out the new 'create table' def written below. Organised files properly in subfolders. Read about the Gene Ontology Consortium. 1h. **17-6:** See lab notes on testing today. 3.5h. **19-6:** reading over TMA03 requirements, project guidelines and adding stuff to the report outline (pretty much dome with Q1 of TMA03 – still need to read it again at a later date). 1.5hr. *week 18 total: 10h*
...			
Aug 9	Data entry, brushing up report		**30-8:** re-created the logical model due to aforementioned food changes (bactdb2.IML), still need to print that out. 0.5h reading up on another 'applied bioinformatics' thesis that could be of use to beef up the techie part of the report. 1h. Or maybe add an appendix on a brief explanation of ORM? 7k words is just not enough. Made changes to bact2.IMO (better 'reading' of the fact types – printout again), added/changed a few bits in the report, reading more of the thesis, may be a waste of time re report improvement after all. 1.5h. **1-9:** improved summary, last section discussion and the odd footnote. 1.5h. Added some on usability and evaluation. 2h. **3-9:** reading + adding fig 1.1 about ontologies to the report. 3h (+ a little time I did not record). **4-9:** added table 3.4 on design features. Re-read/correct this later. Added a little to fig 1.1. 2.5h. *week 29 total: 12 hrs*

Figure 3: A section of a project log.

   (b) You don't have sufficient competencies in topic x, or
   (c) Scratching the surface revealed a plethora of underlying problems that also need to be solved.
2. You had a few drafts but the computer crashed and you did not make a backup.
3. You need resource x to continue that was supposed to be available by week 5, but only came available by week 10.
4. Family emergency.
5. ...

Delays caused by 1(a) are easy to detect if you keep a project log: you planned for task x to take a week, and after, say, 4 weeks your still working on it. Among other typical causes: testing software and bug-fixing is a common task that takes more time than one initially anticipates, and designing good and efficient algorithms is not as easy as it sounds.

If you do not have sufficient competencies, then you will have to put in extra time and effort, consulting textbooks etc. If that fails within a reasonable amount of time, try harder. If that fails, you are probably not capable enough in the selected field, but perhaps you are in another topic, so there might be a way to change topic—but this means you will have to put in extra time to make up for the lost time.

Delays due to 1(c) are quite common in science. This most likely will result in redefining the topic of your thesis to something more specific to narrow down the problem into something feasible within the allotted time.

As for cause 2: DO MAKE BACKUPS EACH DAY!

The third cause is a catch-all for various *external factors* that can mess up your planning. Say, the Internet was down due to a cyber-attack, there is a 3-week strike, it takes ages to get your test subjects together, the specific hardware upgrade hasn't arrived, your supervisor decided to go on an extended research visit out of town, etc. The latter three causes can be anticipated and if you know from your initial planning that that may/will occur, then revise your schedule such that those things that may take more time are attended first, and you can work on other activities in the meantime (like reading papers, writing a framework of your thesis). Your project log will help with that as well, to demonstrate you did start early with the possible time-consuming things and did all feasible other activities as a stop-gap, which

makes people somewhat more lenient to accept the 'external factors' excuse and make amends.

5 Closing remarks

Rest me to say that these are only suggestions and guidelines. They are based on related material and my own experiences. There is not a 'one way' to do a thesis, and I have tried different approaches with different theses I completed successfully (in the sciences, engineering, and humanities), which partially depended on the topic of the thesis. Making a schedule of activities (including a prioritisation of activities), sticking to them, and keeping a logbook invariably were useful, and start writing early.

References

[1] M. Ashby, "How to write a paper," January 2000, engineering Department, University of Cambridge. [Online]. Available: http://www-mech.eng.cam.ac.uk/mmd/ashby-paper-V6.pdf

[2] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," *Communications of the ACM*, vol. 32, no. 1, pp. 9–23, 1989.

[3] A. H. Eden, "Three paradigms of computer science," *Minds & Machines*, vol. 17, pp. 135–167, 2007.

[4] G. D. Gopen and J. A. Swan, "The science of scientific writing," *American Scientist Online*, vol. November-December, 1990. [Online]. Available: http://www.docstyles.com/library/ascience.pdf

[5] C. Johnson, "What is research in computing science?" glasgow Interactive Systems Group (GIST), Department of Computer Science, Glasgow University, Glasgow, G12 8QQ. [Online]. Available: http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html

[6] N. Stanton, *Mastering Communication*, 3rd ed. Macmillan, London, 1996.

Additional sources consulted

Nutt, Werner. How to write a PhD Thesis. Heriot-Watt University, Edinburgh, UK. 2004. [presentation slides].

The Open University. The IT and Computing Project—Resources. Faculty of Technology. version 2. 2003 [CD-ROM CDR0579].