NOTES ON PRESENTING THESES

Aaron Sloman

School of Computer Science

The University of Birmingham

Last updated Feb 1992

CONTENTS

{1} Introduction

These notes are intended to guide, but not direct, research students when planning and writing their theses. The notes are primarily concerned with research theses (MPhil and DPhil/PhD). For course-based MA or MSc students the requirements are not so stringent, but the notes may be of some use to them too.

It is assumed that the topic of research is AI, Cognitive Science, or Computer Science (including HCI), though many of the comments are more generally relevant.

I have produced this document because too many research students are being asked by their examiners to re-submit theses after substantial modification. This is wasteful for everyone and we should try to move to a situation where most theses are accepted first time, and where none of the re-writing requested is simply to improve presentation or organisation. I.e. only modifications of substance should be necessary.

There is no fool-proof way to ensure that no re-writing will be required: No matter how good the student and the supervisor, examinations are always chancy and there is always the risk that examiners will require additional substantive work to be done, e.g. to extend the range of a program, fix a flaw in a proof, extend the generality of some notation, include discussion of some relevant literature not considered, etc.

Most of the points below are not concerned with the substance of a thesis but with its presentation. They complement TEACH PSTYLE which makes general points about project descriptions, applicable to a variety of levels e.g. undergraduate, masters, doctoral, etc. (TEACH PROGSTYLE discusses programming style.)

Presentation may seem relatively unimportant, but part of what is being assessed is your ability to communicate (`Doctor` once meant `teacher`). Moreover, a presentation that makes the structure of your work clear can reveal gaps in arguments and other deficiencies, thereby helping to improve the substance. It can also help others appreciate the real strengths of your work, and how it contributes to our knowledge or understanding.

This document does not live up to its own standards as regards presentation! It needs to be re-written with a clearer and more logical structure, with hierarchical section headings, etc. But I have not had time.

Comments and suggestions for improvement are welcome.

{2} Some general comments

{2.1} You are not expected to write a world-shaking thesis, nor a mammoth tome. Avoid any temptation to add bulk simply to make the thesis look more substantial.

{2.2} The object of the thesis is to demonstrate to the examiners that you can do research of a quality that should lead to one or two refereed journal publications. Be prepared at the oral examination to say which bits, if any, you think are publishable in that form. In rare cases you can argue that it is publishable only as a whole, because all the bits are too closely connected for separate publication. (That can be a sign of poor organisation in your thinking.)

{2.3} NB it is not enough simply to develop some useful software or programming technique. Lots of people working for commercial organisations do that. They may make a lot of money, but they don't get PhDs for it. A PhD thesis must advance KNOWLEDGE in some way. So you must include an analysis of the new knowledge embodied in your program or technique. This requires comparing your work with work done by others, as described in later sections in this file. It also requires you to provide an analysis of the strengths and weaknesses of your program or technique, clearly identifying its limitations, explaining why it succeeds where others have failed, etc. Merely demonstrating that you can do something new has been described as the `Look ma, no hands!` approach. It isn't enough.

{3} Good communication is extremely important

{3.1} The thesis should also demonstrate that you can communicate effectively, not only to narrow specialists in your field but also to others who can be expected to find the results interesting. This means that obscure jargon should be avoided (unless it is explained first) and the same goes for notation.

{3.2} It also means using examples all over the place to illustrate the general points you are making. Examples should be clear and pointed, and preferably very short and memorable. Try to avoid `cluttered` examples: a good example should be the simplest one that illustrates the problem you are addressing or the idea you are presenting.

{3.3} If you use a formal notation that is not widely known, then it is a good idea to give English translations of any formula that is at all complex (e.g. has more than a dozen symbols or has a deeply nested structure). Even if the notation is well known, not everyone will be familiar with it, so when in doubt give a translation. (But don't overdo it.)

{3.4} If you introduce a lot of technical terminology that is not generally known to all workers in the field (not just your specialised sub-field, but the discipline as a whole) then you should have a special section summarising the terminology with cross references to places where terms are defined.

{3.5} Try to think of your reader as intelligent but given to wilful misunderstanding. Try, especially, to anticipate the main forms of prejudice that could produce misunderstanding and guard against them with persuasive explanations, examples, etc.

{3.6} Remember: if anything can be misunderstood it probably will be!

(I've incorporated a number of comments by Alan Bundy in this section.)

{4} Motivate the reader

{4.1} An important aspect of communication is always making sure that the reader wants to read on. So in the very first chapter, and at intervals thereafter, you should make sure that you say why what you are doing is interesting or important. It may be because it solves some important theoretical problem. It may be because it solves an important practical problem. It may be because it reveals an underlying unity in a variety of previous theories and techniques. If you can't give good reasons why people should be interested in your work, then you probably shouldn't be doing it.

{4.2} Another aspect of motivation is making sure that the reader knows WHAT you are doing, as well as why. Good illustrative examples are important for this. The section on scenarios, below, expands on this. But it is not enough to give examples: you must present both illustrative examples and a general characterisation of the scope of your work. This includes negative characterisations: i.e. state clearly what the programs cannot do, which facts the theories, cannot explain, etc. This will lead into the section on possible further work. (See below)

{5} Structure the thesis

{5.1} Good communication does not necessarily imply writing in the style of a novel. The work should be structured so that important points (e.g. definitions, examples, theses, proofs, etc.) are easy to find if one looks back at the thesis. So make sure that they are numbered or labelled in some way that facilitates cross reference. (E.g. this document is numbered so that you can easily communicate with the author – or others – by complaining about paragraph (4) for instance.)

It is also advisable to have lots of section headings with a numbering scheme that shows the structure (e.g. chapter 4 section 3 first sub-section will be numbered 4.3.1). All section headings should be listed in the main table of contents giving page numbers.

{5.2} If you have tables or figures, number them according to the chapters they are in. E.g. the first figure in chapter 4 is Fig 4.1, the first table is Table 4.1. Alternatively base the numbering on subsections. This will help with their location in large chapters. E.g. the third figure in section 4.3.2 is Fig 4.3.2.3.

{5.3} It is not customary to include an alphabetic index of subjects in a thesis, but your examiners will probably be grateful if you do. Certainly if you define technical terms (`the frame problem`) or abbreviations (`ATMS`) or acronyms make sure you have a list giving page numbers where they are defined. It is probably also wise to assume that the reader will NOT remember an acronym last used 60 pages earlier. So whenever you use an abbreviation ask yourself whether the reader should be reminded of its definition or given a cross reference to the definition.

{6} Commonly required sections of a thesis

The following section or chapter headings are required in most theses, though you may find it preferable for some of the items to be spread across several chapters. Some of the points made in this section are expanded in later sections.

{6.1} Introduction This should give an overview of the main objectives of the thesis, including an account of why the work is worth doing (see section on motivating the reader), and a summary of what has and has not been achieved. It may also be helpful for the reader to have an indication of how you solved your problem, even if you can't yet give full details. The introduction should include an overview of the whole thesis, helping the reader to understand what is coming later, and providing information on which bits to read if he wants to take short cuts.

{6.2} Review of related work Sometimes this needs a separate chapter sometimes not, e.g. where you have such reviews in a number of different chapters dealing with different topics. However, it is very important in a thesis that you demonstrate that you are familiar with relevant literature, that you can expound it properly and that you know exactly what your own work adds to the work of others. (See sections on surveying related work).

{6.3} More detailed statement of the problem you have worked on The first chapter need not go in to full technical detail. It should be readable by people who are not experts in your field. A later chapter, which may come before or after the literature review, or be combined with it, can go into full technical detail on the nature of the problem.

{6.4} One or more chapters outlining your own solution. There are two main strategies that can be followed.

(a) Give a high level overview of the solution, then a more detailed overview, then a still more detailed description. I.e. this is the method of `progressive deepening`.

(b) Present your solution in stages, e.g. describing different techniques or partial solutions separately, followed by a chapter showing how they are combined.

Even if you adopt (b) it is probably a good idea to have an element of (a) - i.e. start with a high level overview before going into the details.

{6.5} Illustrations/Demonstrations of what the solution achieves This may consist of examples of the execution of the program, solutions to theoretical problems, uses of the new notation you have developed, etc. It is important that you don't merely provide the examples but also give some analysis showing what they are examples of. Be sure that the reader understands the scope of your thesis. This includes being honest about what the work does not achieve. If you claim complete generality then then you are almost certain to fall flat on your face when someone provides a counter-example later on.

{6.6} Discussion of possible further developments No PhD thesis is ever complete. There are always limitations to the concepts, theory, technique, or program. Make sure that you have looked for such limitations and that you have some ideas about how further work may reduce them. This could be part of a concluding chapter.

{6.7} Discussion of how to evaluate the thesis This could either be a separate chapter, or part of the introduction and concluding chapters. Show that you know how someone should assess your work. Explain what would count as success or failure for your project. Evaluation of a theory in cognitive science might include doing experiments on people (even if you have not done them you should say which experiments would be relevant.) Evaluation of theoretical work in computer science might include attempting to apply it to design of new languages or hardware, or to software engineering techniques, or simply to the solution of other theoretical problems. Evaluation of a new program or technique would include comparing what it achieves with what is achieved by previously existing software and techniques. There are several different dimension in which software can be evaluated: generality, usability, portability, maintainability, understandability, efficiency, etc. Make sure you and your readers know which dimensions are relevant to assessing your work.

{6.8} Conclusion This should summarise the main points of the thesis, evaluate what has been achieved (see discussion of evaluation), summarise the way it compares with prior work, mention limitations and failings, and sketch possible future developments or future research suggested by your work.

{6.9} Acknowledgements It is conventional, though not absolutely necessary, to have a section acknowledging the people who have helped, inspired, advised you, the institutions that have supported you etc. E.g. if you have a research council studentship say so and give the studentship number. If you have taken someone else's idea and developed it, this should be stated in the main text, though a brief note can also be included in the acknowledgements section. Most people appreciate acknowledgements, but don't go over the top and include everyone you know, including the bus-driver who gets you to work....

{6.10} Bibliography Every item referred to in the thesis must be included here. There are different styles for bibliographies and citations, as described below.

{6.11} Appendices These may include detailed mathematical proofs, detailed definitions of formalisms, detailed descriptions of programs or techniques used, and examples of the program's execution if you have developed a program. Opinions differ on whether actual code should be an appendices. I strongly recommend that where there are any interesting algorithms or techniques the code should be included. But don't include all the trivial details of your program, including low level routines (e.g. concatenate two lists).

{7} Issues concerning length

{7.1} Length limits in exam regulations are UPPER BOUNDS, not targets to aim at. If the main text of your thesis is over 150 pages or 55000 words, then there's a good chance that it is too long. Trim all waffle and repetitions.

{7.2} If thesis plus appendices is much over 220 pages or 80,000 words then it is probably too long and you risk making busy examiners upset at having to be burdened with it - unless the whole thing is fascinating, from beginning to end.

If you have lots of diagrams or pictures (e.g. for a thesis on vision or image processing), that can justify additional bulk. Similarly if there's lots of empirical data that you have collected, e.g. for input to your program. Even then ask yourself whether it ALL needs to go into the thesis, or only a sample that makes the points adequately.

If you have empirical data that are too bulky to go into the thesis, make sure they are preserved in a form that will allow others to access them, e.g. to check out the claims in your thesis or other publications.

{7.3} Include the INTERESTING parts of the program in an appendix. Include only enough to enable a competent programmer to replicate your program if necessary.

Do NOT include obvious and trivial procedures (e.g. defining a procedure to intersect two lists, or join two lists, or count the elements of a list satisfying a predicate, etc. are all trivial. Don't include them. A procedure to compare two networks and build a description of the difference is not trivial. Include that, unless you can refer to a widely available publication that describes the algorithm very clearly.) If in doubt about what to include ask your supervisor for advice - then use your own judgement - it's YOUR thesis.

Even if you do not include all your program code in the thesis, you should be willing to make it available to others so that they can test your claims, criticise your work, or build on it. (Sometimes software cannot be made available in this way because it is commercially valuable.)

{8} The opening chapter(s)

{8.1} Start with a good, clear, compact, complete overview. By the end of the first few pages of chapter 1, your reader should have a very good idea of your main achievements, including whether you have written a program and if so what sorts of things it can and cannot do (at least at a high level of abstraction).

Alan Bundy has suggested to me that it is useful if students who are starting to write up their theses first compose what he calls a 'thesis message': a sentence (or more) per chapter playing two roles: as a description of the chapter and as a part of an argument when read in sequence. This helps the student locate gaps in the argument, ensures that the chapters are in the right order, ensures that there IS a message rather than a collection of disconnected thoughts, etc.

This message should then be reflected in the title, abstract, introduction, conclusion and thesis as a whole.

{9} Surveying related work

{9.1} A literature survey is a necessary part of any thesis. It can take different forms, e.g. bunched in one place or distributed across several chapters so that literature is discussed in the context where it is relevant (make sure the reader knows which you are doing - e.g. after the first portion of a literature survey state that remaining literature will be surveyed in other chapters where it is relevant).

{9.2} One thing you should avoid is a very superficial survey in which you cover 25 authors in 10 pages giving a potted summary of each that will give little information to readers who do not already know the work.

{9.3} Choose a few of the authors who have made the main contribution to the field and give an in depth discussion that will show the examiners that you are able to expound the work of someone else clearly, accurately, and critically. Then, if necessary, give a list of other work in the field saying that you don't have space to survey it in detail. At least that will show that you are aware of it. Better still if there are different approaches, different views, or different kinds of results, etc. then organise the list into different categories.

If possible choose at least one author whose views are opposed to yours and discuss the issues in detail.

{9.4} Imposing a new structure on previous work in the field is one way of making a contribution to knowledge.

{9.5} Tracking down relevant literature is less fun than working on your program, and therefore too many students don't do the job properly. It is your responsibility to make sure that you have covered all the main relevant work. By the end of the first year or so, any good research student should know more about recent literature in the field than his or her supervisor, who is probably too busy to keep up properly. So don't just depend on your supervisor to tell you what to read. Your survey must include both recent work and the most important earlier work, which you can track down by following up references in other people's bibliographies. If you don't look at the history of your topic you are in danger of re-inventing wheels, including wheels that other people have demonstrated dohn't work. (This happens too often.)

{9.5} The next two sections are also relevant to the literature review.

{10} Two kinds of literature survey: scene-setting and critical

{10.1} It's up to you whether you expound your ideas before or after the literature survey. Sometimes discussion of other work nicely sets the stage for your solution. In other cases your own analysis provides a conceptual framework that makes it easier to expound and classify or criticise the work of others.

{10.2} It may be useful (as Alan Bundy pointed out) to distinguish two kinds of literature survey: scene-setting and comparative evaluation. An early chapter (e.g. chapter 1 or chapter 2) can include a `scene setting` survey to help the reader understand what problems you are addressing and how they relate to the work of others. Later on, either in a separate chapter, or distributed over several chapters, you can include `comparative evaluation` surveys to show in detail how your work extends or improves on others (or how it doesn't!) This is part of the process of convinving examiners that you have done something original and significant.

{11} Criticising the work of others

{11.1} Remember that before refuting X you should present the views of X in as strong and convincing a form as possible: otherwise you risk being accused of refuting a caricature or straw man. (I owe this point to the writings of Karl Popper.) There is no point arguing against a view that only a fool would support. If possible improve on X's own arguments before you try to refute them - i.e. anticipate possible ways of wriggling out of your criticisms. Too often people write criticisms of a particular view without asking `How would I react to this criticism if I were a really intelligent and well informed person on the other side?`.

{12} Scenarios and examples

{12.1} Many AI theses have made good use of sample scenarios (a) to demonstrate what the problem domain is and (*b*) to demonstrate what the program can do. A scenario is an example of a hypothetical or actual piece of behaviour, e.g. solving a problem, making a plan, engaging in a dialogue. Good examples of expositions using scenarios are the theses by Winograd and Sussman.

{12.2} It is important that scenarios serving these two purposes are clearly distinguished, unless the same one serves both. I.e. if you use a scenario to define the problem domain, but your program cannot cope with it, say so, so that your reader is not given false expectations. I.e. If you give examples of scenarios make sure you indicate clearly which can and which cannot be done by your program. Also make clear whether the input and output are as they would be for your program, or whether you have tarted them up for the purpose of communication.

{13} Describing programs: a formal description

{13.1} If you have written a program make sure that in addition to a scenario giving examples of what it can do, you also give a fairly formal account of its capabilities. Mere examples don't, by themselves, make clear what the program cannot do. Also, readers don't want to have to plough through lots of verbiage when a concise formal account will suffice.

{13.2} If there is a way of explaining an algorithm or relationship in a well known formalism, e.g. algebra or predicate calculus do so - don't just witter on in English. However, if you use formulae or equations, re-state them in English if they are at all complex, for instance if there are more than two implicitly or explicitly quantified variables or more than a dozen or so symbols, or if the notation is not widely known.

{13.3} Avoid vagueness, imprecision, etc. If you say that there is a relationship between two observables say WHAT it is. E.g. don't just say that measuring X enables you to infer Y - give the formula or relationship. (But be clear whether this is an empirical or a mathematical result.)

{13.4} If there are difficult new concepts, include the verbal explanations, with illustrative examples, but make sure that there is a formal summary for quick reference later. E.g. someone doing related work wanting to check that she has covered all the cases you've dealt with should be able to go through an explicit check list without having to dig it out of the main expository text.

{13.5} `Formal` does not necessarily mean expressed in a formal language. E.g. it may be enough to lay out the information in a formal way, using tables, charts etc. It must be concise, clear, and well structured.

{14} Describing programs: saying WHAT they do

{14.1} As far as possible separate out your description of WHAT a program does from HOW it does it. A common mistake is to muddle these two up.

The account of WHAT the program does (not HOW it does it) should include at least the following:

```
What kind of domain(s) it is concerned with.
What objects, properties, relationships, events etc. in that domain
    it can deal with.
    (Make sure you give an EXHAUSTIVE specification of the types of
    things, not just a few examples. Although the TYPES should be
    exhaustively specified, you need not include all the INSTANCES
    of those types.
    For the sake of clarity list those things it cannot do that
    readers might be tempted mistakenly to assume it does do.)
What inputs it can have initially.
    If possible, give a formal specification (e.g. a grammar) as
    well as examples and descriptions in English.
Where it gets its inputs from:
    The user at a terminal? Another procedure that invokes it
    with arguments? A global database or set of files on disk?
What interactions can occur (if it is an interactive program)
    If possible give a grammar or other formal specification
    for its possible behaviours.
What (final) outputs it produces.
    If possible give a grammar for the output, and a formal account
    of the relations between input and output. If it is not possible
    to give a formal account that is clearer and more concise than
    the program itself say so and explain why. (How did you know
    what program you were trying to write?)
Where the output goes
    Printed on the terminal? Results returned to a calling
    procedure? Stored in a global database or disk file?
What operations it can do. E.g if it is a simulation program,
    what exactly are the actions it simulates. If it is a problem
    solving program what kinds of solutions can it generate.
```

{14.2} A summary account of what your complete program does should occur near the beginning of the whole thesis. Descriptions of smaller components can be left to chapters outlining the implementation.

{14.3} If you have a Prolog predicate that has lots of different rules to handle different cases, then don't SIMPLY include all the rules - explain the principle on the basis of which you have produced all those cases. (e.g. `There is one case for each type of insect, and 300 types of insects are dealt with`, or `The predicate handles different types of vehicles and there is one case for possible number of wheels, from 2 to 20....` etc.)

{15} Describing programs: saying HOW they work

{15.1} The account of HOW the program works should be presented in a manner that is, as far as possible, independent of the particular programming language used, so that someone could use the description to re-implement the program in another language.

{15.2} Don't mix up levels. Choose a level of description and stick to it. If you refer to a sub-system in that description, then you can describe it somewhere else, again sticking to a suitable level. A very common way to make descriptions unintelligble to the reader is to switch rapidly between a high level overview and gory details.

{15.3} At each level of description you can explain how a program, or part of a program works by presenting:

```
a. A specification of its main components (e.g. main datastructures
   or databases and the main processing components) and

    1. their tasks (what sort of things they do)
    2. the data-flow between them
    3. the control relations (what calls what)
        (including how much is conceptually parallel even if
        implemented sequentially).

b. How the different kinds of objects, etc. etc. are represented
    (i.e. what sorts of data-structures are used - i.e. how many
    different types there are, what their components are, how they
    are related, etc.)

c. What algorithms are employed (except where they are trivial)

d. How the main program is broken up into smaller ones, at the
    next level of detail.

e. What limitations the program has and why.

f. A formal or informal complexity analysis: How its performance
    scales up with problem size - e.g. is it linear, exponential etc

g. The implementation environment: machine, operating system,
    language and compiler/interpreter used. (e.g. not just Lisp,
    but whose Lisp system.)

h. Some indication of times for typical problems on the system
    described in (g).
```

Use block diagrams and flow charts where this will help to make things clear, but be sure the semantics of such diagrams are not left unspecified. Don't use the same kind of diagram for two different purposes (e.g. a state transition diagram and a block diagram showing components). Don't use the same kind of box for two different kinds of components, e.g. a database and a processing module. Don't use the same kinds of arrows or links to represent different relationships, e.g. flow of data and flow of control.

{16} Further information about the program

{16.1} If you have a program state clearly what it achieves: is it something that others could make use of either as a sub-program, or as a template for a family of programs? For what purposes can it be used - does it have practical applications or is it only of theoretical interest? Does it test some hypothesis? Did it help you clarify certain concepts?

{16.2} If there is a program say whether it is in a form in which others can run it, what user documentation there is, etc. Give a brief outline of what one has to do to run it. Don't include user documentation in the thesis (unless that is a major result of the research, e.g. for a project on human-machine interface design).

However, user documentation should exist somewhere, and examiners should be able to request it. They MAY also ask for a demonstration of the program as part of the oral examination, so make sure that the program is working and demonstrable.

{16.3} Explain the relationship between the implementation and your requirements for the program. Is it provable that your program does what you intend it to do, or can you simply claim that you have tested it exhaustively? In the latter case give a description of the range and variety of the tests used. Don't include them all in the thesis if that would make it too bulky - just give an overview.

{16.4}* Try to motivate the design decisions for your program. Don't bother to comment on all the details, but for all the main features, or at the least the features that you think worthy of mention, be very clear as to whether you chose them because

```
a. You are trying to model something that you think works in the
same manner as your program.

b. You chose that design in order to explore its possibilities
    (Suggested by Luc Beaudoin)

c. There were several different options, and no rational way of
choosing between them, so you have made an arbitrary choice.

d. It is just an implementation detail where the choice is of no
theoretical significance anyway.

e. It was chosen for efficiency, or clarity, or maintainability
(i.e. for some good engineering reason.)

f. The choice was largely determined by the available hardware and
software. (E.g. there was a library and you just used it.)

g. You couldn't think of any other way to do it.

h. You can't remember why you chose that method....
```

Don't waste time explaining details that are of no theoretical interest in the main part of the thesis. If they are worth reporting because they are not obvious put them in an appendix.

{17} Criteria for evaluating your thesis

{17.1} Make clear how YOU think your work is to be assessed. What would have counted as an unsuccessful outcome for your research? The mere fact that you have written a program that meets your specification or that you have written down some axioms and definitions is not in itself evidence of success.

{17.2} Why should someone want a program to do that? Why should anyone be interested in your axioms and symbols? What is the new knowledge? Is there a new useful design technique? Have you discovered some new important algorithms or heuristics for dealing with a class of problems? Have you found new ways of representing information usefully? Have you got a new model of some kind of human cognitive ability? Does the model suggest new kinds of empirical research that can be done? Does the model explain previously known facts better than previous theories? What does the program add to the theory? Does the program simply help to check that your theory is consistent and workable? Does the program allow new predictions of human behaviour to be made when given appropriate inputs? Does it merely explain a range of possibilities without predicting which ones will actually be realised (in the sense explained in chapter 2 of A.Sloman The Computer Revolution in Philosophy)?

I.e. Make clear what new knowledge you have discovered, and its nature, eg.

```
previously unrecognised flaws in the work of others
new requirements for a design or simulation
a new formalism or programming language
a new computer representation
a new algorithm or heuristic
improvements on the work of others
new facts about how people work
new concepts / taxonomy /conceptual framework for thinking about
    a particular domain
a new previously unrecognised problem for AI or cognitive science
a new design for human/machine interface
new relationships between old problems
a new generalisation of things previously thought to be
    different or unrelated (e.g. techniques, representations,
    problems, etc.)
new predictions about human behaviour
new negative results - concerning what won't work.
```

{17.3}* If you have collected empirical data explain what their significance is, and why anyone should be interested. Rememer: it is terribly easy to do experiments on people and collect data. Unlike physical or chemical apparatus almost anything you ask people to do will produce some response because they are intelligent and will interpret your instructions. For the same reason there is likely to be individual variation. But merely collecting data, finding averages, drawing graphs, etc. is of little scientific value. There must be some interesting theoretical implication, or the data should illustrate some important concept. If the data refute some previously believed theory then that can be a useful piece of research.

Also when you have collected data beware of wild and woolly speculation as to how they are produced. If you offer an explanation, be sure that it is the sort of explanation that is sufficiently precise and detailed to be the basis of a construction of a working mechanism. E.g. to say of someone `He solved the problem by constructing an image in his mind and examining` is to say something pretty worthless. It is not at all clear what it means to construct an image in one's mind: is it just a metaphorical way of speaking, like `The pressure was building up in him`, or `He was pulled in two directions at once`, or is there something like a 2-D surface within his brain, on which visual processes of analysis and interpretation operate? All too often psychological writings use such psuedo-explanations because they make people feel they understand. The ancients thought they understood what it meant to say that different kinds of matter (Earth, Air, Fire and Water) all sought out their `natural` place. Don't fall into similar traps.

{18} Program style

{18.1} In your program do not use incomprehensible abbreviations for procedure names or variables. E.g. if use `top_left_corner` NOT `tlc` You may be able to remember what `tlc` stands for but you can't expect readers to do so. If you have done this in your code, then expand it into something intelligible for readers in the thesis. (Saying you have done so.)

See also TEACH PROGSTYLE

{18.2} Include comments with the programs, and where appropriate give examples as well as general explanations in your comments. But don't mix up the program with extensive examples of trace printing.

{19} Program output and tracing

{19.1} Give examples of `trace` printing produced by your program running, but make sure the examples are carefully selected to highlight interesting points. Nothing is more offputting than pages and pages of indigestible program output. You can't expect readers to plough through masses of tedious detail. If necessary, add comments by hand to draw attention to interesting points. Also edit out repetitive output, but indicate that you have so.

{19.2} Don't assume that the output of standard tracing (or spying) facilities is adequate for readers just because it is adequate for you, the program developer. In 99% of cases it is better to devise special-purpose printing procedures that will print things out in a form that is both clear and interesting to read.

{19.3} For a trivial example, look at the trace output produced by the following Pop-11 instructions (mark and load them):

```
lib river
start();
trace putin, getin, crossriver, getout, takeout;
putin(chicken);
getin();
crossriver();
getout();
takeout(chicken);
```

{19.4} You may represent things in a compact form e.g. using numbers, or using an ordered list of names, but don't expect readers to remember that the first number in the list represents the number of children and the second name is the name of the spouse. Make sure your trace output includes all relevant information to help the reader, even if it was not necessary for you. But don't assume the reader is STUPID. Think of the reader as a bright second year doctoral student in the same general area as yourself - but not working on the particular sub-topic.

{19.5} Choose appropriate bits of illustrative output to work into the main text. You can include more details in an appendix. But don't put it ALL into appendices: examples of the program behaving should be part of your main exposition if the program is worth reporting on at all.

{20} Proof-reading is very important

{20.1} Before submitting the thesis read through it carefully, asking yourself - how would I feel about having to read this if I were a very busy and over-worked academic who doesn't necessarily know this sub-field in great detail. Look for opportunities to improve clarity, remove repetition, correct errors, etc.

{20.2} Use an automatic spelling checker if possible, to help you find most errors. Many errors e.g. grammatical errors, cannot be checked automatically (yet). Get a friend to read the thesis looking for typographical and other errors.

Double check the spelling of EVERY proper name, e.g. every author's name in your bibliography. Too many people are sloppy about spelling of names. E.g. do not confuse things like Jonson/Johnson/Johnstone Ramsay/Ramsey, Allan/Alan/Allen, etc.

Examiners get VERY annoyed if they are used as proof-readers compiling long lists of minor errors for you to correct.

If you have any figures or tables that are referred to in the text make sure that the figures (e.g. lettering, captions) are consistent with the text. Too often either the text or a figure is changed without the other being changed too. It is infuriating to read in the text about the item labelled X in Figure 4.5 when there is no label X in Figure 4.5.

Sometimes there are reasons for putting figures or tables separately from the text that discusses them (e.g. you want to put several figures on successive pages, and the text discussing them is fairly lengthy). In that case make sure that someone turning to a figure or table who has not yet read the text can either tell what it is about or will know where to look for an explanation (e.g. the caption to the figure or table should mention the section where it is explained.) For examples of how to do this look at the journal: Scientific American.

A good text formatter will enable you to avoid most of these problems, including, for example, labelling of figures.

{21} Avoid embarrassing omissions (proofread carefully)

{21.1} If not everything can be inserted by the text processor you are using be prepared to write things in by hand in black ink. But make sure there are NO omissions. While you are preparing the text keep a file listing all the items to be inserted by hand, then before you make photocopies go through that list carefully, ensuring that you have made all the inserts. Leaving such things out (e.g. figure captions, references, etc.) gives an impression of sloppiness.

{21.2} Similarly if you don't have a detailed reference when you write something and leave a gap in the text MAKE SURE you fill in the gap for the final version. Again, keep all such things in a `to be done` file and check them carefully before submitting.

{21.3} Make sure that every item that you refer to in the text is included in your bibliography. It is very irritating for the reader to find a reference to (Bloggs 1903) then not find the entry in tbe bibliography.

A good text formatter should help you avoid this.

{21.4} Don't put items in the bibliography simply because they are relevant. Include ONLY those things you refer to explicitly somewhere. A bulky bibliography is of no intrinsic merit. The only exception is the case where production of a new bibliography on the material of your thesis is itself part of the work, in which case say so.

{21.5} Use footnotes only to give details of references to literature, or possibly to remind the less well informed reader of some technical point. Do not use them for substantive extensions to the discussion, qualifications, etc. If you have something important to say work it into the main text, even if you label it as a digression from or parenthetical insertion into your main. This is one way in which clear section headings are useful.

{22} Try out the thesis on a friend or colleague

{22.1} Try to find the right balance between excessive terseness and excessive verbosity. Try out your draft thesis on at least one other *D*.Phil student who should be able to say whether it is readable interesting, clear, convincing, etc.. (Similarly you should be prepared to read at least one draft thesis written by another student.)

{22.2} In particular don't assume that everything you have explained will always be remembered by your reader. So if the reader is likely to need a reminder say something like `the floozle strategy (defined in section 3.4.2) can be used....` Don't repeat points just because they are relevant in different contexts. It is unkind to confront examiners with a repetitive and therefore unnecessarily bulky thesis.

{23} Bibliography and references

{23.1} There are different conventions about format for references and bibliography. Look at journals in your field and see what they do. Often there are instructions for authors inside the back or front cover. Choose a set of conventions and stick to them. E.g. some bibliographies put initials after surname, some before. Some people put the publication date immediately after the author's name, some at the end of the entry. It is important to distinguish titles of book chapters or journal articles from titles of books or journals.

```
 - Use ‘single quotes like this’ or ’like this’ for the titles of
   chapters or articles. (Not all printers can handle "‘" nicely,
   so check that out before you use it. Using an asymmetrical pair
   of single quotes can look very silly in the printout.)

 - Titles of books or journals should be italicised or underlined.
```

{23.2} There are different conventions for bibliographical refererences within the main text. The most compact is by a numerical reference (e.g. [45] would refer to bibliography entry number 45.)

Alternatives are name and date (Smith 1975, Jones 1932a) or name and number (Smith [45]). As a reader I prefer name and date since very often that will suffice to tell me what is being referred to whereas if it is just a number then I am forced to turn to the bibliography. Also there is more chance that you will get numbers wrong as you update the bibliography.

A good text formatter can ensure that you follow uniform citation conventions, and will often give you a choice of formats.

{24} Quotations

{24.1} If you include a quotation from another author make sure you give a FULL reference including page number so that it can be found quickly and easily. E.g. the reader may wish to check the context to see whether you have misunderstood.

{24.2} Don't assume that just because you can understand French, Urdu, or Latin, or whatever, your readers can. If you include a quotation in a foreign language you MUST give a translation into English as well.

{25} If English is not your native language get help

{25.1} If you are not a native speaker of English you are STRONGLY advised to pay someone who is a native speaker to work carefully through your thesis improving the spelling, syntax, etc. where necessary. The examiners have to be convinced that the English is at least up to the standard required for publication in an English language journal.

Examiners differ on the importance that they attach to linguistic competence. Some examiners insist that the thesis be written in good English even if you are not a native speaker of English. Others are willing to make allowances for language if the scientific and technical content is good enough. The best advice is to take no chances and assume you are going to get the first kind of examiner.

{25/2} Even if you don't speak English there is no excuse for spelling errors, as there are spelling checkers available on most computers. Make sure that you double-check the spelling of names of other authors.

{26} Encouraging final comment!

{26.1} If you find these comments daunting, remember that many other people, not all of them geniuses, have succeeded in getting PhD.s However, too many of them have been asked by examiners to do substantial chunks of re-writing first.

{27} References (Good scenarios)

G.J. Sussman *A Computer Model of Skill Acquisition.* New York: American Elsevier, 1975.

T.S. Winograd, *Understanding Natural Language.* Edinburgh: Edinburgh Univ. Press, 1972. (Winograd's scenario is re-printed in many books, e.g. M.Boden's Artificial Intelligence and Natural Man.)

{28} Further reading

Phillips & Pugh *How to get a PhD* Open Univeristy Press ISBN 0 335 155367 (paper back)

Bundy, A. du Boulay, J.B.H., Howe, J.A.M. & Plotkin, *G.*, 'How to get a Ph.D. in A.I.', in *Artificial Intelligence: Tools, Techniques and Applications, O*'Shea, *T.* & Eisenstadt, *M.* (eds.), Harper & Row: London, 1984.

The following Poplog TEACH files are also relevant.

```
TEACH PROPOSALS
TEACH PSTYLE
TEACH PROGSTYLE
```

Acknowledgements

Several people at Sussex, and Alan Bundy in Edinburgh, have contributed useful suggestions, which led to revisions of early drafts.

$poplocal/local/teach/theses

The University of Birmingham 1992.