Com S 430
Guidelines and deliverables for tutorials

Overview

Your assignment is to write a document and to give a talk and demo on your topic. For most topics, the written part will generally be in the form of a tutorial. You should expect to write somewhere between 6 and 12 pages, although, your document may be longer depending on the number of screenshots, illustrations and/or code examples you include.

The talk should be about 12 minutes and will generally not be able to cover everything that is in your written document; do your best to convey the main ideas and at least some of the technically interesting details. Both the paper and the talk will be graded on technical content, organization, and presentation (the writing for the former, and the oral presentation for the latter). Approximate weights are as follows:

- 35% Written tutorial technical content, including code examples
- 25% Written tutorial organization of document and quality of writing
- 20% Talk content and organization
- 20% Slides and oral presentation

Deliverables include:
- your written paper in pdf form
- an archive of accompanying code examples if applicable
- and your presentation slides.

Deliverables are due at the time of your presentation, except for the first five presenters.

The target audience is a technically savvy group of your professional peers and possibly your boss. You are going to invest significant effort to learn about a framework, language, or language feature. You are writing a paper to try to synthesize everything you've learned for everyone so that the rest of your team does not have to work so hard. Look back on your own learning curve and struggles, and think carefully about how everything should be presented when someone is seeing it for the first time.

No doubt you have had to read quite a bit of technical documentation, and you know how frustrating it can be. Hopefully you can draw on that experience to think of ways to smooth the learning curve for your readers or listeners as much as possible.
Assume that everyone in your audience is working on concurrent systems in some way, so there is a lot of interest of how things are actually implemented in terms of threads and tasks, and how thread-safety is guaranteed. Anticipate their questions.

Some topics are much more broad than others. It is part of your responsibility to figure out how to narrow the scope in order to deliver something with interesting technical content. Once you get started on your reading and experimentation, you are encouraged to discuss the scope of your paper with the staff if you are not sure what to emphasize.

In almost all cases, you should create some original code examples that you can demo and that the reader can potentially try out, and include them separately. Be extremely selective about including code listings in your document or slides, and try to visually highlight just what’s important for what you are discussing. (You'll also be submitting an archive of your examples and demo code.)

**Written component**

You can use a direct and conversational style, but the document should be relatively formal. (You are primarily talking to your co-workers and other developers, but your boss may be reading it too.)

- Above all: be clear, concise, and organized.
- Avoid unnecessary words, irrelevant concepts, and any other forms of b.s.
- Avoid making claims you can't substantiate in your references.
- Be careful to define terminology and use it consistently.
- Understand who your audience is and make appropriate assumptions, including familiarity with previous presentations.
- Avoid using the passive voice ("Mistakes were made"). Be direct.
- You are not writing a blog, so don’t use “I” or talk about your experiences and opinions (the use of "we" or "you" may be appropriate, within reason).
- Avoid slang, sarcasm, and flippancy. (Humor in small doses may be ok.)
- Do not evangelize. You are not trying to sell anything; this is not a "persuasive essay".

You should expect to have:

- Table of contents (this is trivial to create if you use your word processor’s headings correctly).
- An introduction. This is extremely important. You tell the reader what the document is going to be about. This typically requires giving a brief overview of the tool, language, or library you are covering, but the main purpose is to introduce the *document* itself.
If you are making specific assumptions about the reader's preparation (other than being an experienced developer) it can be helpful to state those here, e.g. "We assume the reader is familiar with the basics of Erlang functions and list operations."

- An overview or motivation section. Give the reader some context for the problem being solved by this language or library. What is it for? Is it language feature for general-purpose programming, an enterprise tool, a technique, or what? What problems does it solve? Would it make sense to briefly describe some other ways to solve the same problem? In some cases it makes sense to combine this with the introduction.
- Linear organization into sections that have accurate, descriptive titles. (A listing of API calls does NOT qualify as "organization").
- When including illustrations, use labeled figures and refer to them. If you use your word processor’s features for cross-references this is easy.
- If a code example is more than 10-12 lines, make it a separate figure and refer to it. This way you can avoid big blank spaces on a page and avoid splitting a code listing across pages.
- References section. Be scrupulous in attributing quotations, pictures, or code examples to sources. All sources should be cited and listed in the References section using the ACM format. If you have sources that you used but did not directly cite in the paper, add a section entitled "Additional Sources" for these. For ACM citation format see http://www.cs.ucy.ac.cy/~chryssis/specs/ACM-refguide.pdf

**Talk**

You will have a strict 15 minute time limit to allow some time for questions, discussion, and editorializing by your instructor. Practice your talk and time yourself, aiming for about 12 minutes. Give your talk to your roommates. Give your talk to your dog. Give your talk to the fermenting potato salad in the fridge. Try to be comfortable with the material to the point that you don’t have to rely on memorization. (However, memorizing your talk is much better than reading it from your slides or notes!)

- You should prepare slides (PowerPoint or other) to help present your main ideas. **Do not use screens full of code** in your IDE as a presentation format.
- Remember that the #1 mistake in preparing slides is to use too many words. You want the audience to be listening to you, not furiously reading your slides. Carefully choose a few words or phrases that will help the listener focus on what you are saying rather than distract from what you are saying.
- Use pictures or illustrations whenever you can.
- Choose code examples judiciously, make the font large enough to read and visually highlight the lines or details that illustrate your point.
- Use animation only if it is helpful somehow to getting your point across; otherwise it’s usually just a distraction.
- Your presentation should normally include a live demo if possible. If what you are presenting is a development tool it is ok to bring up and/or use an IDE as part of your demo. *This is where many presentations crash and burn*, so do this sparingly. Be aware of what the audience can see and follow: make the font large enough to see, make your mouse/cursor operations slow and deliberate, and be sure to pause and visually highlight the line or lines you are trying to emphasize or discuss. If you have 30 lines showing on the screen, *you* may know which line you're looking at, but nobody else does.