Capstone Project Proposals[*]

Suggestions for deeper explorations

Sudarshan S. Chawathe

This brief guide describes a few key components of good project proposals. It is not a recipe; rather, it is meant to trigger deeper exploration of some of the important questions a project proposal should answer. It is designed to help with the task of developing a nascent idea for a project into a well rounded project proposal. While it is possible to organize the project proposal based on these questions, it is likely that an alternate organization is more convenient for most work.

It is useful to think of the project proposal as providing answers to the following about the project:

1. What is the project all about? What problem does it solve? What are the goals? What is the focus and the overall strategy?
   These questions are what many would consider the main description of the project.

   For example, we may propose a project to design, build, and evaluate a general-purpose sorting library. The simple problem description in this case is quite standard. However, we must indicate why there is a need for another sorting library and how we hope it will be better than what already exists. For instance, we may decide to focus on an optimized implementation for embedded systems and similar environments. In such environments we must cope with limitations such as low CPU speed, strong memory constraints, and the need to release resources on demand, for extended periods. Alternatively, we may decide to focus on sorting data resident on a complex storage architecture, composed of local disks, network-attached storage (local) and data resident on remote servers.

2. Why is it interesting? Why is it fun?
   While a project that strongly appeals to one person may seem humdrum to another, it is nevertheless possible to explain the aspects of a project that make it interesting to the proposer.

   Continuing our sorting example, we may indicate some features of the work that make it interesting: Sorting has widespread applications, from sorting data in user interfaces to low-level operations in the internals of database systems. There is a rich body of work on sorting, which provides a nice foundation and is interesting to learn; at the same time, there are many unanswered questions, both theoretical and practical. If we choose the embedded-systems option, the possibility of using our work as part of an application that runs on cell phones and other similar devices may also be interesting. In these environments, many of the performance assumptions we may be accustomed to making may no longer hold, making the task of adapting existing sorting methods more challenging and interesting. Further, the relative simplicity of the problem makes it amenable to a satisfying, thorough analysis that may not be practicable for more complex problems and systems.

3. How will we know when the work is completed? What are the main tasks and dependencies?
   This question, and its ramifications, is perhaps the most important one, and one that is often overlooked. It is tempting to answer this question by "when it all works" but defining exactly what must work, and how well, is very important. It is useful to think of the proposed work as composed of a core task along with a few additional ones. The core task should be one that we are very confident of finishing well before the deadlines. The additional ones are those we hope to complete, assuming we do not face too many unexpected problems. It is very important to define all these tasks very carefully, keeping implementation dependencies and other constraints in mind. It is also useful to then subdivide each of these tasks, especially the core task, into smaller units that better allow us to budget our time and effort, and monitor our progress.

   In our sorting example, we may define a core task based on sorting data in an embedded Java environment (say, J2ME with profiles MIDP 2.0 and CLDC 1.1), with at least 512 KiB of heap space. For this definition to be meaningful, we

[*]A Guide for Computer Science Majors at the University of Maine. Original version Spring 2008; revised 2010-01-15 (thanks to comments from Joe Bane). chaw@cs.umaine.edu.

must add many more details. (Certainly it is not much of a challenge simply to implement a sorting algorithm that works in this environment.) For instance, we may require the implementation to compare favorably with a naive porting of standard implementations to this environment. Further, this core task (and others, typically) may include experimental evaluation and other analysis task.

4. How will the work be evaluated?
   A project may be evaluated along several dimensions, and the following list is not exhaustive. Not all of these may be applicable to a given project, and they may be of varying importance, depending on the goals of the project.

   (a) Performance. CPU time, memory footprint and bandwidth, disk footprint and bandwidth, network usage, energy requirements, and so on.

   In our sorting example, it is natural to study the CPU, memory, and disk use. If the focus is on network-based storage, the network bandwidth and latency are also important factors.

   (b) Generality. How large is the class of applications and environments in which the results are applicable? It is often possible to achieve high performance at the cost of lower generality.

   For instance, the use of a radix-based sorting may provide improved performance in our running example, but the resulting sorting program may then be inapplicable to data that must be sorted using an arbitrary, user specified, comparison function.

   (c) Simplicity. How easy is it to explain and implement the method? How easy is it to use? Does it require extensive tuning of parameters? How simple and intuitive are the instructions in the user-guide document?

   A sorting implementation that requires the user to specify various parameters, such as cut-over thresholds from quicksort to insertion sort, is less desirable than one that requires only the data and the comparison function.

   (d) Extensibility. How easy is it for us, and others, to extend the method, and implementation?

   For example, a sorting implementation that provides the necessary guidance and hooks to ease implementation on different hardware is preferable to one whose build-in assumptions and design make this task difficult.

   We can certainly think of other dimensions by which to evaluate the work. The important point is to determine and document these at a very early stage.

5. How does it relate to prior work? An insufficient investigation of related work typically leads to disastrous results. A substantial portion of the time and energy spent on a project must be budgeted for investigation of related work. Such work falls into many different categories, such as the following:

   (a) Competition: methods and implementations with goals very similar to ours.

   In the sorting example, this category includes other implementations of sorting on environments similar to the one we study.

   (b) Applications: techniques, implementations, and end-user applications that are likely to use the results of our work.

   Our sorting implementation may be useful, for instance, to database system implementations or user interfaces.

   (c) Components: the tools and methods used by our work.

   For example, our network-data sorting implementation may use a prior implementation of main-memory sorting for small datasets as a module.

   (d) Ideas from other areas: non-obvious connections often exist between diverse areas.

   For example, we may find use for some graph-coloring methods in the design of parallel sorting algorithms.

While it may be very difficult to answer the above questions, and address the issues they raise, to our complete satisfaction early in the Capstone project experience, it is important to have at least reasonable answers to them, along with indications on how these answers are likely to be refined down the road. It is important that they be discussed with all parties involved in the project, especially the project adviser.