**How to Write a Software Project Proposal**

This document describes how to develop a proposed software project in a structured manner. Although primarily intended for an academic course in software engineering, it has wider applicability.

**1.   Introduction**

Writing a solid software proposal requires a *major effort* (weeks or months of work) and you may not be able to produce it on time to start working on your project. You risk falling behind the rest of the class.  
Therefore, only highly motivated and knowledgeable teams should consider proposing their own projects.  
Working on a *modified version* of an existing project is not such a bad idea. After all, [everything’s been done before](http://www.amazon.com/Everythings-Been-Done-Before/dp/B0067FMTRO)—we’re always building on existing ideas and products, regardless of whether they were developed in this course or somewhere else.

Although your own proposal may be novel, you should not expect different treatment from other teams in the class who are working on the projects that were defined for them. Working on your own project is your choice and you should be ready to invest any additional effort required to excel in the class.

**2.   Writing a Software Project Proposal**

Software engineering proposal is a document that a software developer submits to a business customer for acceptance. The proposal describes the problem to be solved and explains the resulting benefits to the customer.

**What is Important in a Software Proposal**

The key for a great proposal is to invent a great idea.  
There is no “official template” for writing software proposals.  
To sum up: **Content is the key. Form just helps to convey it.**

Keep in mind that this is a proposal only, so you do not know that it will be accepted as such. Therefore, you do not want to waste time on formalities by following some formal descriptions/formats/templates.   
The key is to *diagnose the problem* and *propose a treatment*, so to convince the customer to accept your proposal.  
The customer wants to know first that you know **what** you are proposing to do.  
They are not interested in idiosyncrasies of software engineering or programming. They assume you know that.  
Only if you receive the customer’s approval, will come the issue of knowing **how** to do it.

The most important thing about a software engineering proposal is that the **proposal is about the problem domain**, ***not*** about programming. The proposal should clearly describe the motivation for the proposed work and the proposed solution along with its *expected business value*. The proposal should **not** contain any technical jargon. There are three key components of a software engineering proposal:

DIAGNOSE PROBLEM   ==>   PRESCRIBE TREATMENT   ==>   DESCRIBE PLAN OF WORK

**How To Write a Software Engineering Proposal**

To write a software engineering proposal, follow these steps:

1. **Problem diagnosis:** describe the **problem domain**, describe clearly what kind of issues exist in the current practice that you would like to address.  
   Be as specific as you can and provide as many details and examples as possible.  
   Describe the problem domain and the problem that you’re planning to solve. People usually make a mistake of describing at a very high level the problem, too generic, and then make a huge leap and dive deep into the tiny detail of their own solution. You must make effort to bridge this gap incrementally. Start with a brief description of high-level context (few sentences or a paragraph), then describe some specific issues that you’re interested in, then provide more specific details about the sub-issues that your work will tackle.  
   The best approach is to observe personally the current practice, so that you know what you are talking about. Another useful approach is to interview “domain experts,” people who are working in your target domain and who will be your potential customers. Expert opinion carries greater weight/credibility to your statements and analyses than a naive guess.  
   Think of yourself as a journalist, interviewing your potential users and documenting their opinion about current problems they are facing and suggestions on how to address those problems.
2. **Proposed treatment:** describe how you propose to address the diagnosed problems.  
   What specific interventions will you introduce?  
   What kind of metrics will you use to evaluate your success in solving the targeted problems?  
   How will you know that you achieved your objective?  
   Discuss the *business value* of your proposed solution. What will your customer and users gain from your proposed system that they are lacking now?  
   Be as specific as possible in describing the envisioned benefits of your proposed solution. Provide example scenarios of how your proposed system will be used and explain how this solution is better than the current practice.
3. **Plan of work:** make a convincing case that you know how to achieve the proposed goal. Step-by-step, go in details about what needs to be accomplished, how long it will take, and how it relates to other parts (independent vs. builds upon another part). You cannot know all the details yet, because you haven’t even started, but your plan should outline the main steps so that it is clear that you have a plan. Do not just copy a generic plan from a textbook, because that looks lame.  
   Describe your team. What are the strengths and expertise of each team member? Explain why your team size is adequate to tackle the problem, and why the problem size requires your team and not fewer people.  
   Keep in mind that this is only an initial plan so that you can give your customer a preliminary estimate of costs and expected completion date. You will need to adjust both of these estimates as you progress, but hopefully not by much.  
   State how you will know that you succeeded. How will you measure the success of your system in addressing the customer’s problem that you diagnosed?

Your proposal must be written in *lay language*, plain English, and you should avoid any engineering terminology (unless your problem domain is an engineering process, i.e., you will develop software to improve an engineering process).

The proposal should accurately describe the **user experience**, though in lay language rather than using software engineering jargon. The proposal *is* about the user experience of the proposed system, so this must be as accurate as possible. Otherwise, the decision to accept or reject the proposal will be based on inadequate information. On the other hand, the proposal should avoid discussing the implementation details of the system (how it will be done). It is useful, though, to include what is necessary to accomplish the proposed goal, such as access to certain *data* (e.g., financial reports, traffic reports, etc., depending on the problem domain), other *resources* (e.g., sensors, devices, equipment), or *expertise* (e.g., statistician, security expert). It helps to know whether such resources are available and at what cost.

Although a naive user may not be interested in specific algorithms or software libraries that you may be planning to use, it is important that you mention their existence. The reason is again that your customer (or course instructor) needs to be assured that you know what you are up to. Otherwise, the cutomer may be worried that you don’t really know what you may need, and this may be an open-ended project that may take forever and very likely may fail. If you say that you will reuse an existing software or algorithm, this significantly reduces the uncertainty about the proposed work.

Check [**here**](http://www.ece.rutgers.edu/~marsic/thesis-guide.html) for more details about writing engineering documents.  
See additional links of interest at the bottom of that webpage.

**Why Writing a Software Proposal is Difficult**

It is not that writing itself is difficult. What is difficult is learning about your problem domain, how things are done now, how your proposed system will change that. Even if you’re familiar with the domain, it takes time to think new solutions. For example, you may be using a parking garage every day, or even working as a garage operator, but it takes a great deal of time to figure out how to propose a smart parking system.

One may argue that it is to be expected at an initial stage that your proposal will be sketchy on details. Writing a detailed project proposal requires time. I agree. I spent a lot of time preparing these project descriptions.  
That is why you should avoid wasting time on formalities and focus on what matters. Which is, *diagnosing the problem* and *proposing a treatment*. But you cannot diagnose a problem unless you have a deep understanding of your problem domain. You cannot tell your customer how you will improve their work unless you know very well how they are doing their work now!

This is where the problem arises. You may be excited about writing new programs, but you cannot diagnose a problem and propose a treatment unless you know your problem domain. You must know how your customer does their business now and how your proposed system will affect your customer’s work. It is important that you are excited about developing something you feel is valuable and important.

A key quality of a great software engineer is that he or she is willing (scratch that, *excited*) to learn new problem domains.  
A great software engineer will learn medicine if he or she is to develop a medical software system; he or she will learn sociology if he or she is to develop a social networking system (popup quiz: what’s [Dunbar's number](http://en.wikipedia.org/wiki/Dunbar%27s_number) and why [Path](https://path.com/) limits your social network to 150 friends?); the engineer will go observe how restaurant personnel do their work if he or she is to develop a software system to help run a restaurant; etc.

**Software Project Must Include Programming**

When deciding about the project, the most important thing to keep in mind is that the software product you will develop should require lot of programming. This is a *design* rather than a *programming* course. However, in order to learn good design, the process must include actual implementation of the design. Therefore, the project ***must include programming***.

For example, creating a data-bank and ***processing*** this data is a programming task. If your system just allows the user to set or modify values of different database records, that is **not** data processing. Examples of programming (data processing) include, data-bank re-organization, keyword-based search and retrieval, filtering and summarization of the data-bank, selecting a small group of users for notification, etc. However, just saying: *The system will provide search facilities*, may mean that you are using SQL database calls and doing **no programming at all**. And, programming is done in a programming language, such as C, C++, Java, or C#.

**Note: Designing web pages *passive* web pages which contain *only* HTML is *not* programming.** Therefore, your project *can* include web page design, but that is not enough.  
On the other hand, designing active webpages using AJAX, Flash, etc., is programming and is perfectly acceptable for the class project.

Focus more on **automatic** data processing: not what user can manually do, but what system does automatically to save the user’s effort. For example, suppose you are designing a system for online purchasing of airline tickets. You could incorporate the following data processing features:

* When user queries for the available flights, sort the results by price or some other attribute before displaying them;
* Allow pending reservation requests for a price below a certain threshold, or in case another passenger cancels a completely booked flight;
* Design a feature for marketing to the users based on their travel history;
* If the flight gets delayed or cancelled, the system should automatically send notification to the passengers;

Strive to reduce the user’s manual interaction with the system and increase the system “smarts”! Let the system do the work for the user! Nobody is impressed with a system where the user has to invest hard work to get even simple tasks accomplished.*Make every effort to reduce the number of clicks and keystrokes necessary for task completion.*

Keep in mind that the key purpose of this course is that you learn how to do **modular design** of software and how to document the design using **symbolic representations**, i.e., UML diagrams. Thus, you should *avoid flashy user interfaces and proprietary software*. These will definitely not contribute positively to your grade, and may have negative effects. Limit your software to the tools commonly and *freely* available on the Web.

**Choice of the Topic**

The proposed topic can be almost anything; however, it is advisable to come up with something novel and innovative to keep you interested in its realization, and because innovativeness tends to make impression on the reviewers.

Your project may or may not be Web based, but it must include programming.

The process of casting out the right-size project for a given period of performance can be very frustrating. You cannot ever know what you’ll encounter while working on it. I believe that it’s better to start more ambitious and later scale down, if necessary. Start with an initial idea of desirable services and discuss as much details as possible with your partners to figure out whether or not you as a team can develop it in the given time frame. Of course, you’ll have chance to adjust your goals during the semester, as you learn more details about your target product.

**Individual Product Ownership in a Team**

When working in a team, it is important to create a balanced division of labor. This will ensure that all team members are contributing to the success of the project. The best approach for balanced workload is to assign **product ownership**. Your “product” is what you deliver to your customer. User interface, graphic design, database interaction, unit testing, etc. are ***not*** products. Different parts of that product (i.e., *functional features*) should be owned by different team members.

*Product ownership does not necessarily equal labor division*. Your customer wants to know who is the person to go to when she wants to know about the progress or any issues related to a given functional feature. Who actually did the work is your concern and should be reported in the individual contributions breakdown in your future deliverables. That cannot be planned in advance. Ideally, the product owner should build the product he owns, but the actual work may be subcontracted if need be, because of individual expertise available in the team.

Novices often make mistake by subdividing the work along the lines of different computer disciplines or individual expertise. For example, one person says “I’ll do the database,” another person says “I’ll do the user interface,” etc. Do not subdivide the work by sub-systems of the proposed system. It’s too early for such subdivision while writing the proposal. You don’t know what subsystems will be there, what architecture you will adopt, and how much effort will be needed. Instead, subdivide your work by functional features.

Initially we split work to different functions, but later we may drop some use cases if important ones turn out to be very difficult, so we need to reassign responsibilities. That's ok now, when we know what is difficult and why people need to be moved, but we cannot know that initially, so initially we make a guess that every person will be able to develop one function.

**3.   Working from an Existing Proposal**

Note that the purpose of project descriptions that you will find there is ***not*** to specify what you should do in your project.  
The reason why I provided these project descriptions is just to save you some time wondering what to do and give you some preliminary ideas, so that you can start thinking about your own proposed system. So, this is just to start you thinking about the issues. You can add or subtract anything that you feel would be appropriate.  
You should also check what the student groups in the past years did.

If you choose to work from an existing project, your main task for the proposal is to describe *how different* will be your project from the projects that were done by past students in this course.

**Example Proposals**

You can find example past proposals by student groups in the download materials. They are not posted as separate documents, but they are included in the final reports, in the section called “Customer Statement of Requirements (CSR)”.

**Borrowing From Past Projects**

You may borrow ideas and implementation from past projects as much as you like. It is fine that you borrow 100% from a past project, take everything and then build on top of it! In fact, this approach would be preferred instead of seeing you reinvent the wheel.  
However, what is important is that you contribute *new value*! For example, you can implement better some features/functions from an existing project; your implementation may be faster, or easier to use. Or, you may add novel features / extensions to an existing project.

**4.   Proposal Format, Submission, and Feedback**

**Proposal Format**

Each proposal should contain the following elements:

1. Project title, Group number
2. URL of your project’s web-site (optional)
3. Team profile:
   1. Individual qualifications and strengths (such as: programming, design, presentation, documentation, management and organization)   
      Note: It is expected that every team member shall be involved in all project activities; this only indicates individual strengths, not their sole responsibilities
   2. Name of the elected team leader, if any (having a team leader is **optional**)   
      The team leader should act as facilitator, to organize group meetings and generally keep track of project activities
4. Proposed project description:  
   Your project description must provide detailed explanation of your proposed project, as described above in [Section 2](http://www.ece.rutgers.edu/~marsic/Teaching/SE/proposal.html#DESCR).   
   You may also describe typical customers for your proposed system and mention if you already have a customer or someone interested in your proposed project.  
   At the end of this section, summarize in a bulleted list what the user will be able to do with your system (“*functional features*”).
5. Plan of work and product ownership:  
   Your plan of work should list and describe the items that you are planning to accomplish in the short term (next few weeks).  
   Split your team into pairs of students (a team of 6 students will have 3 pairs), and each pair should list what specifically they will contribute to the end product:
   1. *Functionality*: which functional features (from the itemized list at te end of the previous section) each sub-team will contribute.  
      Example functional features are customer registration, data capture and storage, data processing to extract statistical parameters, etc.  
      User interface, graphic design, database interaction, unit testing, etc. are ***not*** functional features.
   2. *Qualitative property*, if any, that you will contribute, such as tune up the system performance to achieve response time under *x* seconds, or develop and evaluate an easy-to-use user interface for this-and-this specific functional feature, or ensure confidentiality of a specific set of data, etc.

Product ownership is critical to demonstrate that each team member will play a clearly defined role in the proposed project.

Proposals that are missing any of the above sections will be returned without review.

**Proposal Submission**

Each team should submit a *single document* for their team project. Only **PDF document** format will be accepted.   
Here is a [PDF writer](http://www.cutepdf.com/Products/CutePDF/writer.asp) you can use from Windows. Macs automatically convert postscript to PDF by clicking on the file icon. There are UNIX utilities that convert postscript to PDF for Linux users; make sure you include all fonts in the created PDF.

**Feedback**

We will provide feedback in case we feel the proposed project is not novel or properly described.  
Proposals that are missing any of the sections listed above under “Proposal Format” will be returned without review.   
The reasons for revision request include:

* Project does not appear to include programming
* Project does not appear to be novel
* Project does not appear to be sufficiently ambitious
* Some team members appear not to have clearly defined and important role

The students are encouraged to be **ambitious** in the first iteration of the project. You will have chance to revise your project objectives at the end of the first iteration, by the time of Demo #1. At that time, you must be **realistic** and you may have decide that you need to scale down the project scope, so to be able to finish it by the end of the semester.

**4.   Additional Links to Software Proposal Writing**

♣   Angsuan Chackraborty: [How to Write a Software Proposal](http://angsuman.taragana.net/archive/how-to-write-a-software-project-proposal/)

♣   Elizabeth Smith (Demand Media): [How to Write a Proposal on Web Applications](http://smallbusiness.chron.com/write-proposal-applications-31984.html)

♣   Sun Associates: [10 Tips for Technology Proposal Writers](http://www.sun-associates.com/resources/10tips.html)

♣   James Abela: [EXAMINATIONS Writing a Proposal](http://www.jamesabela.co.uk/exams/Proposal.pdf)

♣   Georgia Perimeter College: [Proposal Writing Stages and Strategies](http://facstaff.gpc.edu/~ebrown/infobr3.htm)

♣   Sid Hutchinson: [How to Write a Software Proposal](http://www.ehow.com/how_8434600_write-software-proposal.html)   (updated May 17, 2011)

♣   Andy Carr: [How to Write a Software Development Proposal](http://www.ehow.com/how_8402174_write-software-development-proposal.html)   (updated May 11, 2011)