First Things First: Prioritizing Requirements[1]

Karl E. Wiegers
Process Impact
www.processimpact.com

Customers are never thrilled to find out they can't get all the features they want in release 1.0 of a new software product (at least, not if they want the features to work). However, if the development team cannot deliver every requirement by the scheduled initial delivery date, the project stakeholders must agree on which subset to implement first. Any project with resource limitations has to establish the relative priorities of the requested features, use cases, or functional requirements. Prioritization helps the project manager resolve conflicts, plan for staged deliveries, and make the necessary trade-off decisions.

Why Prioritize Requirements?

One characteristic of excellent requirements is that they are explicitly prioritized. When customer expectations are high, timelines are short, and resources are limited, you want to make sure the product contains the most essential functions. Establishing each chunk of functionality's relative importance lets you sequence construction to provide the greatest product value at the lowest cost. Customers and developers must collaborate on requirements prioritization. Developers do not always know which requirements are most important to the customers, and customers cannot judge the cost and technical difficulty associated with specific requirements.

A project manager has to balance the project scope against the constraints of schedule, budget, staff resources, and quality goals. One balancing strategy is to drop or defer low priority requirements to a later release when you accept new, higher priority requirements or other project conditions change. If customers don't differentiate their requirements by importance and urgency, the project manager must make these trade-off decisions. Because customers may not agree with the project manager's decisions, they must indicate which requirements are critical and which can wait. Establish priorities early in the project, while you have more options available for achieving a successful project outcome.

It's difficult enough to get one customer to decide which of his or her requirements are most important; gaining agreement among multiple customers with diverse expectations is even more challenging. People naturally have their own interests at heart and they aren't always willing to compromise their needs for someone else's benefit. However, making such priority decisions is one of the customer's responsibilities in the customer-developer partnership.

Customers prioritize requirements initially from the perspective of how valuable each one is to them. Once a developer points out the cost, technical risk, or other trade-offs associated with a specific requirement, though, the customers might decide it isn't as essential as they thought. The developers might also determine that certain lower priority functions should be implemented early on because of their impact on the product architecture. When setting priorities, you need to balance the business benefit that each function provides against its cost and any implications it has for the product's architectural foundation future evolution.

---

[1] This paper was originally published in *Software Development*, September 1999. It is reprinted (with modifications) with permission from *Software Development* magazine.

Games People Play with Priorities

The knee-jerk response to a request for customers to set priorities is, "I need all of these features. Just make it happen somehow." This doesn't help. It can be difficult to persuade customers to set priorities if they know that you may never be implement low priorityrequirements. A developer once told me that priorities are unnecessary, because if we wrote something in the software requirements specification (SRS), we intend to build it. However, that doesn't consider the issue of *when* you should implement each function. Developers may not wish to bother with priorities, because that conflicts with the "we can do it all" attitude they want to convey to their customers and managers.

The reality is that some features are more essential than others. This becomes apparent during the all-too-common "rapid descoping phase" late in the project, when lower priority features are jettisoned to ensure the product is shipped on schedule. Setting priorities early in the project helps you make those trade-off decisions along the way, rather than in emergency mode at the end. Getting a feature half-developed before you determine that it's low priority is wasteful and frustrating.

If left to their own devices, customers will set perhaps 85% of the requirements at high priority, 10% at medium, and 5% at low priority. This doesn't give the project manager much to work with. Steve McConnell identified requirements scrubbing—eliminating those that are not essential and simplifying any that are unnecessarily complicated—as a best practice for rapid software development (see *Rapid Development*, Microsoft Press. 1996).

On one project I know of, the management steering team became impatient when the analyst insisted on prioritizing the requirements. The managers pointed out that they could often do without one feature, but another feature may need to be beefed up to make up for the omitted requirements. They reasoned that if they deferred too many requirements, the resulting system wouldn't achieve the revenue that the business plan promised. When you evaluate priorities, look at the connections and interrelationships among different requirements and their alignment with the business requirements.

Prioritization Scales

A common approach to prioritization is to group requirements into three priority categories. Table 1 shows two typical three-level scales. All such scales are subjective and imprecise, so everyone involved must agree on the meaning of each level in the scale they use. Priority is a key attribute of each requirement that should be included in the SRS or requirements database. Establish a convention for your SRS so the reader knows whether the priority assigned to a higher-level requirement is inherited by all of its subordinate or derived requirements, or whether every individual requirement should have its own priority attribute.

Another issue is the granularity at which you prioritize requirements. Even a medium-sized project can have hundreds or thousands of detailed functional requirements, too many to classify analytically and consistently. You need to choose an appropriate level of abstraction for the prioritization. This could be at the use case level, the feature level, or the detailed functional requirement level, whichever makes logical sense for your situation.

Table 1.
Two requirements prioritization scales.

Names	***Meanings***
High	a mission critical requirement; required for next release
Medium	supports necessary system operations; required eventually but could wait until a later release if necessary
Low	a functional or quality enhancement; would be nice to have someday if resources permit
Essential	the product is not acceptable unless these requirements are satisfied
Conditional	would enhance the product, but the product is not unacceptable if absent
Optional	functions that may or may not be worthwhile

Keep the prioritization as simple as possible to help you make the necessary development choices. You may decide to do an initial prioritization at the feature level and then prioritize the functional requirements within a specific high-priority feature separately. This will help you distinguish the core functionality that must be present for that feature to work at all from refinements you could add in a later release. Include even the low-priority requirements in the SRS. Their priority may change over time and knowing about them now will help you plan ahead for future enhancements.

Prioritization Based on Value, Cost, and Risk

On a small project, the stakeholders can probably agree on requirement priorities informally. Larger or more contentious projects need a more structured approach, which removes some of the emotion, politics, and guesswork from prioritization.

Industry analysts have proposed several techniques that involve estimating the relative value and relative cost of each requirement, such that the highest priority requirements provide the largest fraction of the total product value at the smallest fraction of the total cost. In essence, you're trying to identify those requirements that will maximize the product value within the existing cost constraints. Subjectively estimating the cost and value by pair-wise comparisons of all the requirements is impractical for anything more than a couple dozen requirements.

Another alternative, Quality Function Deployment (QFD), provides a robust and comprehensive method for relating customer value to the proposed product features. A third approach, based on Total Quality Management (TQM), rates each requirement against several specific, weighted project success criteria and computes a score to rank the priority of the requirements. However, in my experience few software organizations are willing to undertake the rigor of QFD or TQM.

In this article I describe a semi-quantitative analytical approach to requirements prioritization. As an example, we'll consider a product called the Chemical Tracking System. The Chemical Tracking System will let research scientists request containers of chemicals to be supplied by the company's chemical stockroom or by commercial chemical vendors. The system will store the location of every chemical container within the company, the quantity of material remaining in it, and the complete history of each container's locations and usage. The Chemical

Tracking System must also comply with federal and state government regulations that require quarterly chemical usage, storage, and disposal reports.

Table 2 illustrates a simple spreadsheet model to help you estimate the relative priorities for a set of product features such as those from the Chemical Tracking System. You can download this Excel spreadsheet from www.processimpact.com/goodies.shtml. This approach distributes a set of estimated priorities across a continuum, rather than grouping them into just a few priority levels. You can apply this model just as well to prioritize use cases or individual functional requirements. This prioritization scheme borrows from the QFD concept of customer value depending on both the customer benefit provided if a specific product feature is present and the penalty paid if that feature is absent. A feature's attractiveness is directly proportional to the value it provides and inversely proportional to its cost and the technical risk associated with implementing it. All other things being equal, those features that have the highest risk-adjusted value/cost ratio should have the highest priority. Of course, these are not the only factors that affect prioritization, so don't use this scheme as your only method for setting priorities.

Apply this prioritization scheme only to negotiable features, those that are not in the top priority category. For example, you wouldn't include items in this priority analysis that implement the core business functions of the product, that you view as key product differentiators, or that are required for compliance with government regulations. You're going to include those features no matter what. Once you've identified those features that absolutely must be included for the product to be shippable, scale the relative priorities of the remaining features using the model.

The typical participants in the prioritization process include:

- The project manager, who leads the process, arbitrates conflicts, and adjusts inputs from the other participants if necessary
- Key customer representatives, who supply the benefit and penalty ratings
- Development representatives, such as team technical leads, who supply the cost and risk ratings.

Follow eight steps to use this prioritization model.

Step 1. List all of the requirements, features, or use cases that you wish to prioritize in a spreadsheet; we'll use features in this example. All of the items must be at the same level of abstraction. For example, don't mix individual requirements with product features. If certain features are logically linked (that is, you would only implement feature B if feature A were included as well), include only the driving feature in the analysis. This model will work with up to several dozen features before it becomes unwieldy. If you have more items than that, abstract related features together to create a more manageable initial list. You can do a second round of analysis at a finer granularity of requirements detail if you need to.

Step 2. Estimate the relative benefit that each feature provides to the customer or the business on a scale from 1 to 9, with 1 indicating very little benefit and 9 being the maximum possible benefit. These benefits indicate alignment with the product's business requirements. Your customer representatives are the best people to judge these benefits.

Step 3. Estimate the relative penalty the customer or business would suffer if the feature is not included. Again, use a scale from 1 to 9, where 1 means essentially no penalty and 9 indicates a very serious downside. For example, failing to comply with a government regulation could incur a high penalty even if the customer benefit is low, as would omitting a feature that any reasonable

customer would expect, whether or not they explicitly requested it. Requirements that have both a low benefit and a low penalty add cost but little value; they may be instances of gold plating.

Step 4. The Total Value column is the sum of the relative benefit and penalty. By default, benefit and penalty are weighted equally. As a refinement, you can change the weights for these two factors. In Table 2, all benefit ratings are weighted twice as heavily as the penalty ratings. The spreadsheet totals the feature values and calculates the percentage of the total product value provided by these features that is attributable to each feature.

Step 5. Estimate the relative cost of implementing each feature, again on a scale ranging from a low of 1 to a high of 9. The spreadsheet will calculate the percentage of total cost for each feature. Developers estimate the cost ratings based on factors such as the requirement complexity, the extent of user interface work required, the potential ability to reuse existing designs or code, and the levels of testing and documentation needed.

Step 6. Developers estimate the relative degree of technical or other risk associated with each feature on a scale from 1 to 9. An estimate of 1 means you can program it in your sleep, while 9 indicates serious concerns about feasibility, the availability of staff with the needed expertise, or the use of unproven or unfamiliar tools and technologies. The spreadsheet will calculate the percentage of the total risk that comes from each feature.

By default, cost and risk are weighted equally, and each carries the same weight as the benefit and penalty terms. You can adjust the cost and risk weightings in the spreadsheet. In Table 2, risk has one-half the weight of the cost factor, which has the same weight as the penalty term. If you don't want to consider risk in the model, set the risk weighting value to zero.

Step 7. Once you enter the estimates into the spreadsheet, it calculates a priority number for each feature. The formula for the Priority column is: priority = value %/ (cost % * cost weight + risk % * risk weight).

Step 8. Sort the list of features in descending order by calculated priority. The features at the top of the list have the most favorable balance of value, cost, and risk, and thus should have higher implementation priority. The key customer and developer representatives should review the completed spreadsheet to agree on the ratings and the resulting sequence.

This semi-quantitative method is not mathematically rigorous, and it is limited by your ability to estimate the benefit, penalty, cost, and risk for each item. Therefore, use the calculated priority sequence only as a guideline. It will take you awhile to calibrate your rating scales for a set of requirements, so iterate through the list after rating all the requirements and make any necessary adjustments. Calibrate this model for your own use with a set of completed requirements or features from a previous product. Adjust the weighting factors until the calculate priority sequence agrees with your after-the-fact evaluation of how important the requirements in your test set really were.

This model can also help you make trade-off decisions when you're evaluating proposed new requirements. Estimate their priority using the model to tell you how they match up against existing requirements, so you can choose an appropriate implementation sequence. Any actions you can take to move requirements prioritization from the political arena into an objective and analytical one will improve the project's ability to deliver the most important functionality in the most appropriate sequence.

Table 2. Sample prioritization matrix for a Chemical Tracking System.

Relative Weights:	2	1			1		0.5		
Feature	***Relative Benefit***	***Relative Penalty***	***Total Value***	***Value %***	***Relative Cost***	***Cost %***	***Relative Risk***	***Risk %***	***Priority***
1. Query status of a vendor order	5	3	13	8.4	2	4.8	1	3.0	1.345
2. Generate a Chemical Stockroom inventory report	9	7	25	16.2	5	11.9	3	9.1	0.987
3. See history of a specific chemical container	5	5	15	9.7	3	7.1	2	6.1	0.957
4. Print a chemical safety datasheet	2	1	5	3.2	1	2.4	1	3.0	0.833
5. Maintain a list of hazardous chemicals	4	9	17	11.0	4	9.5	4	12.1	0.708
6. Modify a pending chemical request	4	3	11	7.1	3	7.1	2	6.1	0.702
7. Generate an individual laboratory inventory report	6	2	14	9.1	4	9.5	3	9.1	0.646
8. Search vendor catalogs for a specific chemical	9	8	26	16.9	7	16.7	8	24.2	0.586
9. Check training database for hazardous chemical training record	3	4	10	6.5	4	9.5	2	6.1	0.517
10. Import chemical structures from structure drawing tools	7	4	18	11.7	9	21.4	7	21.2	0.365
Totals	**54**	**46**	**154**	**100**	**42**	**100**	**33**	**100**	**--**