

First things first: Setting requirement priorities

After most of the user requirements for the Chemical Tracking System were identified, the project manager, Dave, and the business analyst, Lori, met with two of the product champions. Tim represented the chemist community and Roxanne spoke for the chemical stockroom staff.

Dave said, "Now that we have a general idea of the main capabilities you want, we need to think about allocating some of the user stories you've identified to the first few iterations. It's important that we agree on where to start so you can begin getting some value from the system as quickly as possible. Let's do a first-cut prioritization on these user stories so we know what's most important to you. Then we can learn more about exactly what you expect from each of those initial capabilities."

Tim was puzzled. "Why do you need the requirements prioritized? They're all important, or we wouldn't have given them to you."

Lori, the BA, explained, "We know they're all important, but we need to address the most urgent requirements in the first few iterations. We're asking you to help us distinguish the requirements that must be included initially from those that can wait for later iterations. Can you think of certain functionality that would provide the greatest immediate value to chemists or other user classes?"

"I know that the reports that the Health and Safety Department needs to generate for the government have to be available soon or the company will get in trouble," Roxanne pointed out. "We can use our current inventory system for a few more months if we have to."

Tim added, "I promised the online catalog search function to the chemists as a way for this system to save them time. Can we please start on that right away? It doesn't have to be perfect, but we want to get access to the catalogs as quickly as we can."

Tim and Roxanne realized that, because the project couldn't deliver every desired feature at the same time, it would be better if everyone could agree on the set to implement first. They continued sorting their user stories into a top-priority category for early implementation and others that could wait a while.

Few software projects deliver all the capabilities that all stakeholders want by the targeted initial delivery date. Every project with resource limitations needs to define the relative priorities of the requested product capabilities. Prioritization, also called *requirements triage* (Davis 2005), helps reveal competing goals, resolve conflicts, plan for staged or incremental deliveries, control scope creep,

and make the necessary trade-off decisions. This chapter discusses the importance of prioritizing requirements, describes several prioritization techniques, and presents a spreadsheet tool for prioritization analysis based on value, cost, and risk.

Why prioritize requirements?

When customer expectations are high and timelines are short, you need to make sure the product delivers the most critical or valuable functionality as early as possible. Prioritization is a way to deal with competing demands for limited resources. Establishing the relative priority of each product capability lets you plan construction to provide the highest value at the lowest cost. Because prioritization is relative, you can begin prioritization as soon as you discover your second requirement.

Sometimes customers don't like to prioritize requirements, thinking that they won't ever get the ones that are low priority. Well, if you aren't going to get everything you'd like, as is often the case, you should make sure that you *do* get the capabilities that are most important to achieving your business objectives. Sometimes developers don't like to prioritize requirements because it gives the impression that they can't do it all. The reality is that they can't, at least not all at once. Prioritization helps the project deliver the maximum business value as quickly as possible within the project constraints.

Prioritization is a critical strategy for agile or other projects that develop products through a series of fixed-schedule timeboxes. Project teams can populate their product backlog with user stories, features, business processes, and defect stories (bugs awaiting correction). Customers prioritize the stories in the backlog and select which ones they'd like to have implemented in each development iteration. Developers estimate the effort involved with implementing each story and judge how many of these stories they can fit into each iteration, based on their empirically demonstrated delivery capacity as measured by the team's velocity. As new stories are proposed, customers assess their priorities against the contents of the backlog, thus dynamically adjusting scope for the upcoming iterations. All projects should do this to ensure that the team is always working on those capabilities that will get useful software in the users' hands as soon as possible.

On every project, a project manager must balance the desired project scope against the constraints of schedule, budget, staff, and quality goals (Wiegiers 1996). One way to accomplish this is to drop—or to defer to a later release—low-priority requirements when new, more essential requirements are accepted or when other project conditions change. That is, prioritization is a dynamic and ongoing process. If customers don't distinguish their requirements by importance and urgency, project managers must make these decisions on their own. Not surprisingly, customers might not agree with a project manager's priorities; therefore, customers must indicate which requirements are needed initially and which can wait. Establish priorities early in the project, when you have more flexibility for achieving a successful project outcome, and revisit them periodically.

It's difficult enough to get any one customer to decide which of his requirements are top priority. Achieving consensus among multiple customers with diverse expectations is even harder.

People naturally have their own interests at heart and aren't eager to compromise their needs for someone else's benefit. However, contributing to requirements prioritization is one of the customer's responsibilities in the customer-development partnership, as was discussed in Chapter 2, "Requirements from the customer's perspective." More than simply defining the sequence of requirements implementation, discussing priorities helps to clarify the customers' expectations.

Some prioritization pragmatics

Even a medium-sized project can have dozens of user requirements and hundreds of functional requirements, too many to classify analytically and consistently. To keep it manageable, choose an appropriate level of abstraction for the prioritization—features, use cases, user stories, or functional requirements. Within a use case, some alternative flows could have a higher priority than others. You might decide to do an initial prioritization at the feature level and then to prioritize the functional requirements within certain features separately. This will help you to distinguish the core functionality from refinements that can be deferred or cut entirely. As was described in Chapter 5, "Establishing the business requirements," feature prioritization feeds directly into scope and release planning. Don't lose sight of the low-priority requirements, although there's no point in analyzing them further just yet. Their priority might change later, and knowing about them now will help the developers plan for future enhancements.

Various stakeholders need to participate in prioritization, representing customers, project sponsors, project management, development, and perhaps other perspectives. You really need one ultimate decision maker when stakeholders can't agree. A good starting point is for the prioritization participants to agree upon a set of criteria to use for judging whether one requirement has higher priority than another. The prioritization can include considerations of customer value, business value, business or technical risk, cost, difficulty of implementation, time to market, regulatory or policy compliance, competitive marketplace advantage, and contractual commitments (Gottesdiener 2005). Alan Davis (2005) indicates that successful prioritization requires an understanding of six issues:


- The needs of the customers
- The relative importance of requirements to the customers
- The timing at which capabilities need to be delivered
- Requirements that serve as predecessors for other requirements and other relationships among requirements
- Which requirements must be implemented as a group
- The cost to satisfy each requirement

Customers place a high priority on those functions that provide the greatest business or usability benefit. However, after a developer points out the cost, difficulty, technical risk, or trade-offs associated with a specific requirement, the customers might conclude that it isn't as essential as they first thought. The developer might also decide to implement certain lower-priority functions


early on because of their effect on the system's architecture, laying the foundation to implement future functionality efficiently without major restructuring. Some functionality must have high priority because it is required to meet regulatory demands for the application. As with all aspects of requirements development, the overarching business objectives that led to launching the project in the first place should drive priority decisions.

Certain requirements must be implemented together or in a specific sequence. It makes no sense to implement a redo edit capability in release 1 but not implement the corresponding undo capability until some months later. Similarly, suppose you implement just the normal flow of a particular use case in release 1, deferring the lower-priority alternative flows to some later date. That's fine, but you must also implement the corresponding exception handlers at the same time you implement each success flow. Otherwise, you could end up writing code to, say, accept credit card payments without checking to see if the card is valid, rejecting cards that were reported stolen, or handling other exceptions.

Games people play with priorities



The knee-jerk response to a request for customers to set priorities sometimes is, "I need all these features. Just make it happen." They feel that every requirement should be ranked as high priority, and they might not recognize that prioritization will help to ensure the project's success. Start by explaining that all things cannot be done simultaneously, so you want to make sure you work on the right things first. It can be difficult to persuade customers to discuss priorities if they know that low-priority requirements might never be implemented. One developer told me that it wasn't politically acceptable in his company to say that a requirement had low priority. Therefore, the priority categories they adopted were "high," "super-high," and "incredibly high." Another developer who was filling the BA role claimed that priorities weren't necessary: if he wrote something in the SRS, he intended to build it. That doesn't address the issue of *when* each piece of functionality gets built, though.



I recently visited one company that had great difficulty getting their projects done on time. Although management claimed that there would be multiple releases of applications so lower-priority requirements could wait, in reality each project delivered just a single release. Consequently, the stakeholders all knew that they only had one shot to get all the functionality they needed. Every requirement, therefore, became high priority, overloading the team's capacity to deliver.

In reality, some system capabilities are more essential than others from the perspective of satisfying business objectives. This becomes apparent during the all-too-common "rapid descoping phase" late in the project, when nonessential features are jettisoned to ensure that the critical capabilities ship on schedule. At that point, people are clearly making priority decisions, but in a panicked state. Setting priorities early in the project and reassessing them in response to changing customer preferences, market conditions, and business events lets the team spend time wisely on high-value activities. Implementing most of a feature before you conclude that it isn't necessary is wasteful and frustrating.

If left to their own devices, customers will establish perhaps 85 percent of the requirements as high priority, 10 percent as medium, and 5 percent as low. This doesn't give the project manager much flexibility. If all requirements truly are of top priority, your project has a high risk of not being fully successful. Scrub the requirements to eliminate any that aren't essential and to simplify those that are unnecessarily complex. One study found that nearly two-thirds of the features developed in software systems are rarely or never used (The Standish Group 2009). To encourage customers to acknowledge that some requirements have lower priority, the analyst can ask questions such as the following:

- Is there some other way to satisfy the need that this requirement addresses?
- What would the consequences be of omitting or deferring this requirement?
- What effect would it have on the project's business objectives if this requirement weren't implemented for several months?
- Why might a customer be unhappy if this requirement were deferred to a later release?
- Is having this feature worth delaying release of all of the other features with this same priority?



Important If you go through a prioritization process and all of the requirements come out with about the same priority, you really haven't prioritized them at all.



When you evaluate priorities, look at the connections and interrelationships among requirements and their alignment with the project's business objectives. The management team on one large commercial project displayed impatience over the analyst's insistence on prioritizing the requirements. The managers pointed out that often they can do without a particular feature but that another feature might need to be beefed up to compensate. If they deferred too many requirements, the resulting product wouldn't achieve the projected revenue.

Conflicts arise among stakeholders who are convinced that *their* requirements are the most important. As a general rule, members of the favored user classes should get preference in the case of competing priorities. This is one reason to identify and assess your user classes early in the project.

Some prioritization techniques

On a small project, the stakeholders should be able to agree on requirement priorities informally. Large or contentious projects with many stakeholders demand a more structured approach that removes some of the emotion, politics, and guesswork from the process. Several analytical and mathematical techniques have been proposed to assist with requirements prioritization. These methods involve estimating the relative value and relative cost of each requirement. The highest priority requirements are those that provide the largest fraction of the total product value at the smallest fraction of the total cost (Karlsson and Ryan 1997; Jung 1998). This section discusses several techniques people use for prioritizing requirements. Simpler is better, provided the technique is effective.

Trap Avoid “decibel prioritization,” in which the loudest voice heard gets top priority, and “threat prioritization,” in which stakeholders holding the most political power always get what they demand.

In or out

The simplest of all prioritization methods is to have a group of stakeholders work down a list of requirements and make a binary decision: is it in, or is it out? Keep referring to the project’s business objectives to make this judgment, paring the list down to the bare minimum needed for the first release. Then, when implementation of that release is under way, you can go back to the previously “out” requirements and go through the process again for the next release.



Pop goes the requirement

I once facilitated a workshop that had six stakeholders in the room and four more on the phone. We had 400 requirements to prioritize. We opted to decide simply if each was in or out, then figured we’d deal with the “out” ones for the next release. We blocked off several hours in this room to grind through the list. One executive stakeholder had the final prioritization decision when there were conflicts. Shortly into this meeting, he realized that the day was going to be long and monotonous. He decided to have some fun. Every time the team cut a requirement, he made an explosion sound, like blowing up the requirement. It was a fun way to cut scope.

Pairwise comparison and rank ordering

People sometimes try to assign a unique priority sequence number to each requirement. Rank ordering a list of requirements involves making pairwise comparisons between all of them so you can judge which member of each pair has higher priority. Figure 14-1 in Chapter 14, “Beyond functionality,” illustrated the use of a spreadsheet to perform just such a pairwise comparison of quality attributes; the same strategy could be applied to a set of features, user stories, or any other set of requirements of the same type. Performing such comparisons becomes unwieldy for more than a couple of dozen requirements. It could work at the granularity level of features, but not for all the functional requirements for a system as a whole.

In reality, rank ordering all of the requirements by priority is overkill. You won’t be implementing all of these in individual releases; instead, you’ll group them together in batches by release or development timebox. Grouping requirements into features, or into small sets of requirements that have similar priority or that otherwise must be implemented together, is sufficient.

Three-level scale

A common prioritization approach groups requirements into three categories. No matter how you label them, if you're using three categories they boil down to high, medium, and low priority. Such prioritization scales are subjective and imprecise. To make the scale useful, the stakeholders must agree on what each level means in the scale they use.

One way to assess priority is to consider the two dimensions of *importance* and *urgency* (Covey 2004). Every requirement can be considered as being either important to achieving business objectives or not so important, and as being either urgent or not so urgent. This is a relative assessment among a set of requirements, not an absolute binary distinction. As Figure 16-1 shows, these alternatives yield four possible combinations, which you can use to define a priority scale:

- *High-priority* requirements are both important (customers need the capability) and urgent (customers need it in the next release). Alternatively, contractual or compliance obligations might dictate that a specific requirement must be included, or there might be compelling business reasons to implement it promptly. If you can wait to implement a requirement in a later release without adverse consequences, then it is not high priority per this definition.
- *Medium-priority* requirements are important (customers need the capability) but not urgent (they can wait for a later release).
- *Low-priority* requirements are neither important (customers can live without the capability if necessary) nor urgent (customers can wait, perhaps forever).
- Requirements in the fourth quadrant appear to be urgent to some stakeholder, perhaps for political reasons, but they really aren't important to achieving the business objectives. Don't waste your time working on these, because they don't add sufficient value to the product. If they aren't important, either set them to low priority or scrub them entirely.

	<i>Important</i>	<i>Not So Important</i>
<i>Urgent</i>	High Priority	Don't Do These!
<i>Not So Urgent</i>	Medium Priority	Low Priority

FIGURE 16-1 Requirements prioritization based on importance and urgency.

Include the priority of each requirement as an attribute of the requirement in the user requirements documents, the SRS, or the requirements database. Establish a convention so that the reader knows whether the priority assigned to a high-level requirement is inherited by all its subordinate requirements or whether every individual functional requirement is to have its own priority attribute.

Sometimes, particularly on a large project, you might want to perform prioritization iteratively. Have the team rate requirements as high, medium, or low priority. If the number of high-priority requirements is excessive and you're not convinced that they all really *must* be delivered in the next release, perform a second-level partitioning of the high-priority ones into three groups. You could call them high, higher, and highest if you like, so people don't lose sight of the fact that they were originally designated as being important. The requirements rated "highest" become your new group of top-priority requirements. Group the "high" and "higher" requirements in with your original medium-priority group (Figure 16-2). Taking a hard line on the criterion of "must be in the next release or that release is not shippable" helps keep the team focused on the truly high-priority capabilities.

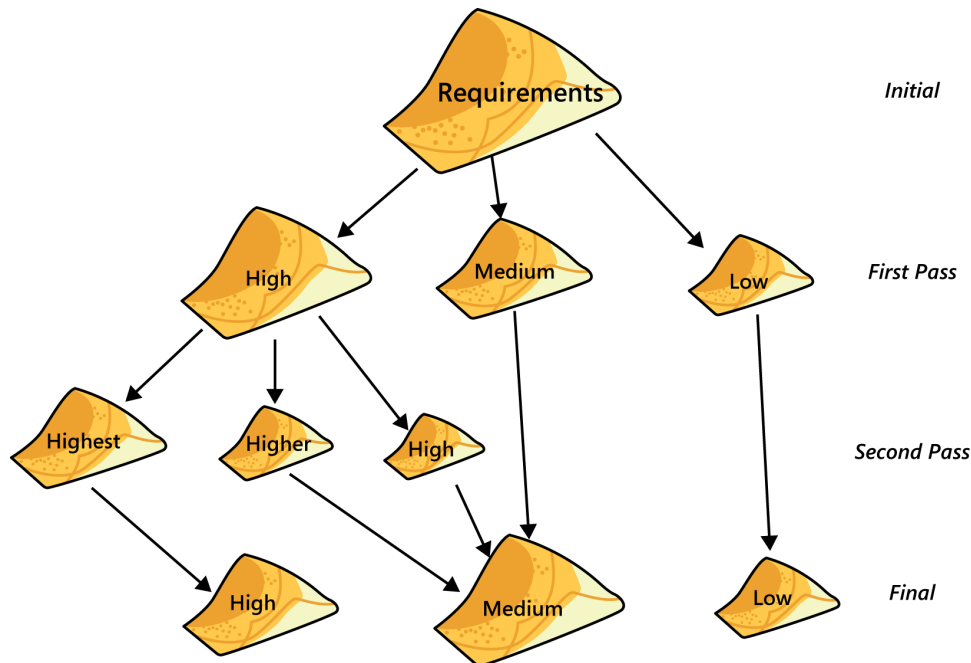


FIGURE 16-2 Multipass prioritization keeps the focus on a manageable set of top-priority requirements.

When performing a prioritization analysis with the three-level scale, you need be aware of requirement dependencies. You'll run into problems if a high-priority requirement is dependent on another that is ranked lower in priority and hence planned for implementation later on.

MoSCoW

The four capitalized letters in the MoSCoW prioritization scheme stand for four possible priority classifications for the requirements in a set (IIBA 2009):

- **Must:** The requirement must be satisfied for the solution to be considered a success.
- **Should:** The requirement is important and should be included in the solution if possible, but it's not mandatory to success.

- **Could:** It's a desirable capability, but one that could be deferred or eliminated. Implement it only if time and resources permit.
- **Won't:** This indicates a requirement that will not be implemented at this time but could be included in a future release.

The MoSCoW scheme changes the three-level scale of high, medium, and low into a four-level scale. It doesn't offer any rationale for making the decision about how to rate the priority of a given requirement compared to others. MoSCoW is ambiguous as to timing, particularly when it comes to the "Won't" rating. "Won't" could mean either "not in the next release" or "not ever." Such distinctions must be made clear so that all stakeholders share a common understanding of the implications of a particular priority rating. The three-level scale described previously, which relies on analysis of the two dimensions of importance and urgency, and focuses specifically on the forthcoming release or development timebox, is a crisper way to think about priorities. We don't recommend MoSCoW.



MoSCoW in practice

One consultant described how a client company actually practiced the MoSCoW method on its projects. "All the action centers around getting an 'M' for almost every feature or requirement that is captured," he said. "If something is not an 'M' it will almost certainly not get built. Although the original intent may have been to prioritize, users have long since figured out to never submit something that does not have an 'M' associated with it. Do they understand the nuanced differences between S, C, and W? I have no idea. But they have figured out the implications of these rankings. They treat them all the same and understand their meaning to be 'not happening any time soon'."

\$100

Prioritization is about thoughtfully allocating limited resources to achieve the maximum benefit from the investment an organization makes in a project. One way to make prioritization more tangible is to cast it in terms of an actual resource: money. In this case, it's just play money, but money nonetheless.

Give the prioritization team 100 imaginary dollars to work with. Team members allocate these dollars to "buy" items that they would like to have implemented from the complete set of candidate requirements. They weight the higher-priority requirements more heavily by allocating more dollars to them. If one requirement is three times as important to a stakeholder as another requirement, she would assign perhaps nine dollars to the first requirement and three dollars to the second. But 100 dollars is all the prioritizers get—when they are out of money, nothing else can be implemented, at least not in the release they are currently focusing on. One approach is to have different participants in the prioritization process perform their own dollar allocations, then add up the total number of dollars assigned to each requirement to see which ones collectively come out as having the highest priority.

The hundred-dollar approach is not a bad way to get a group of people to think in terms of allocating resources based on priority. However, Davis (2005) points out several ways that participants can “game” the process to skew the results. For instance, if you really, REALLY want a particular requirement, you might give it all 100 of your dollars to try to float it to the top of the list. In reality, you’d never accept a system that possessed just that single requirement, though. Nor does this scheme take into account any concern about the relative amount of effort needed to implement each of those requirements. If you could get three requirements each valued at \$10 for the same effort as one valued at \$15, you’re likely better off with the three. The scheme is based solely on the perceived value of certain requirements to a particular set of stakeholders, a limitation of many prioritization techniques.

Another prioritization technique is based on real money, not play money. In Joy Beatty and Anthony Chen’s (2012) *objective chain* technique, you assign an estimated dollar value that represents how much each proposed feature contributes to achieving the project’s business objectives. You can then compare the relative value of features to one another and select which ones to implement first.

Prioritization based on value, cost, and risk

When the stakeholders can’t agree on requirement priorities through the other relatively informal techniques, it might be useful to apply a more analytical method. A definitive, rigorous way to relate customer value to proposed product features is with a technique called Quality Function Deployment, or QFD (Cohen 1995). Few software organizations seem to be willing to undertake the rigor of QFD, although a structured prioritization method adapted from QFD has proven to be helpful.

Table 16-1 illustrates a spreadsheet model to help estimate the relative priorities for a set of requirements. This technique was ranked in the top tier of effectiveness in a comparative evaluation of 17 requirements prioritization methods (Kukreja et al. 2012). The Microsoft Excel spreadsheet is available in the companion content for this book. The example in Table 16-1 lists several features from (what else?) the Chemical Tracking System. This scheme borrows from the QFD concept of basing customer value on both the *benefit* provided to the customer if a specific product feature is present and the *penalty* paid if that feature is absent (Pardee 1996). 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 with the highest risk-adjusted value/cost ratio should have the highest priority. This approach distributes a set of estimated priorities across a continuum, rather than grouping them into just a few discrete levels.

TABLE 16-1 Sample prioritization matrix for the 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.	Print a material safety data sheet.	2	4	8	5.2	1	2.7	1	3.0	1.22
2.	Query status of a vendor order.	5	3	13	8.4	2	5.4	1	3.0	1.21
3.	Generate a chemical stockroom inventory report.	9	7	25	16.1	5	13.5	3	9.1	0.89
4.	See history of a specific chemical container.	5	5	15	9.7	3	8.1	2	6.1	0.87
5.	Search vendor catalogs for a specific chemical.	9	8	26	16.8	3	8.1	8	24.2	0.83
6.	Maintain a list of hazardous chemicals.	3	9	15	9.7	3	8.1	4	12.1	0.68
7.	Change a pending chemical request.	4	3	11	7.1	3	8.1	2	6.1	0.64
8.	Generate a laboratory inventory report.	6	2	14	9.0	4	10.8	3	9.1	0.59
9.	Check training database for hazardous chemical training record.	3	4	10	6.5	4	10.8	2	6.1	0.47
10.	Import chemical structures from structure drawing tools.	7	4	18	11.6	9	24.3	7	21.2	0.33
Totals		53	49	155	100.0	37	100.0	33	100.0	

Apply this prioritization scheme to discretionary requirements, those that aren't obviously top priority. For instance, you wouldn't include in this analysis items that implement the product's core business functions, key product differentiators, or items required for regulatory compliance. After you've identified those features that absolutely must be included for the product to be releasable, use the model in Table 16-1 to scale the relative priorities of the remaining capabilities. Typical participants in the prioritization process include:

- The project manager or business analyst, who leads the process, arbitrates conflicts, and adjusts prioritization data received from the other participants if necessary.
- Customer representatives, such as product champions, a product manager, or a product owner, who supply the benefit and penalty ratings.
- Development representatives, who provide the cost and risk ratings.

Follow these steps to use this prioritization model (it's more complicated to explain than to use):

1. List in the spreadsheet all the features, use cases, use case flows, user stories, or functional requirements that you want to prioritize against each other. We've used features in the example. All the items must be at the same level of abstraction—don't mix functional requirements with features, use cases, or user stories. Certain features might be logically linked (you'd implement feature B only if feature A were included) or have dependencies (feature A must be implemented before feature B). For those, include only the driving feature in the analysis. This model will work with up to several dozen items before it becomes unwieldy. If you have more than that, group related items together to create a manageable list. You can apply the method hierarchically. After you perform an initial prioritization on, for example, features, you can apply it again within a feature to prioritize its individual subfeatures or functional requirements.
2. Have the customer representatives estimate the relative benefit each feature would provide to the customer or to the business on a scale of 1 to 9. A rating of 1 indicates that no one would find it useful; 9 means that it would be extremely valuable. These benefit ratings indicate alignment of the features with the product's business objectives.
3. Estimate the relative penalty that the customer or the business would suffer if each feature were *not* included. Again, use a scale of 1 to 9. A rating of 1 means that no one will be upset if it's absent; 9 indicates a serious downside. Requirements with both a low benefit and a low penalty add cost but little value. Sometimes a feature could have a fairly low value, if not many customers will use it, but a high penalty if your competitor's product boasts that feature and the customers expect it to be there—even if they don't personally plan to use it! Marketing people sometimes call these "checkbox features": you need to say you have it, even if few people really care. When assigning penalty ratings, consider what might happen if you do not include the capability:
 - Would your product suffer in comparison with other products that do have that capability?
 - Would there be any legal or contractual consequences?
 - Would you be violating some government or industry standard?
 - Would users be unable to perform some necessary or expected functions?
 - Would it be a lot harder to add that capability later as an enhancement?
 - Would problems arise because marketing promised a feature to some customers?
4. The spreadsheet calculates the total value for each feature as the sum of its benefit and penalty scores (weighted as described later in the chapter). The spreadsheet sums the values for all the features and calculates the percentage of the total value that comes from each of the features (the Value % column). Note that this is not the percentage of total value for the entire product, just for the set of features you're prioritizing against each other here.

5. Have developers estimate the relative cost of implementing each feature, again on a scale of 1 (quick and easy) to 9 (time-consuming and expensive). The spreadsheet will calculate the percentage of the total cost that each feature contributes. Developers estimate the cost ratings based on the feature's complexity, the extent of user interface work required, the potential ability to reuse existing code, the amount of testing needed, and so forth. Agile teams could base these cost ratings on the number of story points they've assigned to each user story. (See Chapter 19, "Beyond requirements development," for more about estimation on agile projects.)
6. Similarly, have developers rate the relative technical (not business) risk associated with each feature on a scale of 1 to 9. Technical risk is the probability of *not* getting the feature right on the first try. A rating of 1 means you can program it in your sleep. A 9 indicates serious concerns about feasibility, the lack of necessary expertise on the team, the use of unfamiliar tools and technologies, or concern about the amount of complexity hidden within the requirement. The spreadsheet will calculate the percentage of the total risk that comes from each feature.
7. After you've entered all the estimates into the spreadsheet, it will calculate a priority value for each feature by using the following formula:

$$priority = \frac{value \%}{cost \% + risk \%}$$

8. Finally, sort the list of features in descending order by calculated priority, the rightmost column. The features at the top of the list have the most favorable balance of value, cost, and risk and thus—all other factors being equal—should have highest priority. Discussions that focus on those features at the top of the list will let you refine that preliminary ranking into a priority sequence that stakeholders can agree on, even if not everyone gets exactly what they want.

By default, the benefit, penalty, cost, and risk terms are weighted equally. You can change the relative weights for the four factors in the top row of the spreadsheet, to reflect the thought process by which your team makes priority decisions. In Table 16-1, all benefit ratings are weighted twice as heavily as the corresponding penalty ratings, penalty and cost are weighted the same, and risk has half the weight of the cost and penalty terms. To drop a term out of the model, set its weight to zero.

When using this spreadsheet model with prioritization participants, you might want to hide certain columns that appear in Table 16-1: Total value, Value %, Cost %, and Risk %. These show intermediate results from the calculations that could just be a distraction. Hiding them will let the customers focus on the four rating categories and the calculated priority values.



Or, we could arm wrestle

One company that introduced a requirements prioritization procedure based on this spreadsheet found that it helped a project team to break through an impasse. Several stakeholders had different opinions about which features were most important on a large project; the team was deadlocked. The spreadsheet analysis made the priority assessment more objective and less emotionally charged, enabling the team to agree on some conclusions and move ahead.

Consultant Johanna Rothman (2000) reported that, “I have suggested this spreadsheet to my clients as a tool for decision-making. Although the ones who tried it have never completely filled out the spreadsheet, they found the discussion it stimulated extremely helpful in deciding the relative priorities of the different requirements.” That is, you can use the framework of benefit, penalty, cost, and risk to guide discussions about priorities. This is more valuable than working completely through the spreadsheet analysis and relying exclusively on the calculated priority sequence. Because requirements and their priorities can change with time, use the spreadsheet tool throughout the project to help manage the backlog of work remaining to be done.

This priority model’s usefulness is limited by the team’s ability to estimate the benefit, penalty, cost, and risk for each item. Therefore, use the calculated priorities only as a guideline. Stakeholders should review the completed spreadsheet to agree on the ratings and the resulting sorted priority sequence. If you aren’t sure whether you can trust the results, consider calibrating this model for your own use with a set of implemented requirements from a previous project. Adjust the weighting factors until the calculated priority sequence correlates well with your after-the-fact evaluation of how important the requirements in your calibration set really were. This will give you some confidence in using the tool as a predictive model of how you make priority decisions on your projects.

Trap Don’t over-interpret small differences in calculated priority numbers. This semi-quantitative method is not mathematically rigorous. Group together sets of requirements that have approximately the same calculated priority numbers.

Different stakeholders often have conflicting ideas about the relative benefit of a specific requirement or the penalty of omitting it. The prioritization spreadsheet includes a variant that accommodates input from several user classes or other stakeholder groups. In the Multiple Stakeholders worksheet tab in the downloadable spreadsheet, duplicate the Relative Benefit and Relative Penalty columns so that you have a set for each stakeholder who’s contributing to the analysis. Then assign a weighting factor to each stakeholder, giving higher weights to favored user classes than to groups who have less influence on the project’s decisions. Have each stakeholder representative provide his own benefit and penalty ratings for each feature. The spreadsheet will incorporate the stakeholder weights when it calculates the final value scores.

This model can also help you to make trade-off decisions when you're evaluating proposed requirements additions. Add the new requirements to the prioritization spreadsheet and see how their priorities align with those of the existing requirements baseline so you can choose an appropriate implementation sequence.

You don't always need to use a method this elaborate. Keep your prioritization process as simple as possible, but no simpler. Strive to move prioritization away from the political and emotional arena into a forum in which stakeholders can make honest assessments. This will give you a better chance of building products that deliver the maximum business value with the minimum cost.



Next steps

- Reevaluate the requirements in your backlog for an upcoming release, using the definitions in Figure 16-1 to distinguish requirements that truly must be included in that release from those that could wait if necessary. Does this make you change any of your priorities?
- Apply the spreadsheet model illustrated in Table 16-1 to prioritize 10 or 15 features, use cases, or user stories from a recent project. How well do the calculated priorities compare with the priorities you had determined by some different method? How well do they compare with your subjective sense of the proper priorities?
- If the model's priorities don't match what you think is right, analyze which part of the model isn't giving sensible results. Try using different weighting factors for benefit, penalty, cost, and risk. Adjust the model until it provides results consistent with what you expect. Otherwise, you can't trust its predictive capability.
- After you've calibrated the prioritization model, apply it to a new project. Incorporate the calculated priorities into the decision-making process. See whether this yields results that the stakeholders find more satisfying than those from their previous prioritization approach.
- Try one new prioritization technique today that you have not used before. For example, if you use MoSCoW already, try using the three-level method to see how it compares.