JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No.8, September-October 2004

Prioritizing Requirements

Donald Firesmith, Software Engineering Institute, U.S.A.

Abstract

In this column, I address the often difficult task of prioritizing requirements so that the highest priority requirements can be implemented first as part of the scheduling of an incremental, iterative, and time-boxed development cycle. After defining the meaning of the term "priority", the purpose and benefits of requirements prioritization are listed. This is followed by a brief discussion of the challenges and risks that a requirements team must face when prioritizing requirements. Then, various techniques for prioritizing requirements are identified, and finally a set of recommendations (including a recommended prioritization process) are made.

1 INTRODUCTION

On projects producing large complex software-intensive systems, it is not unusual to have hundreds and even thousands of individual requirements. And it is also not unusual for the customer organization acquiring such systems to have valid reasons to want each and every one of its requirements implemented. Yet, such projects cannot avoid the following fundamental facts of life:

- **Differences in importance**. Not all requirements are equally important, and the many different stakeholders in the system typically will not agree as to which requirements are most important.
- **Limited project resources**. All projects have limited resources in terms of budget, staff, and schedule. It is usually impossible to implement all of the requirements, at least not during the system's current release. Thus, non-trivial systems are typically implemented using an incremental development cycle in which the requirements are incrementally developed and implemented.
- **Long schedule**. Such large incrementally-developed systems require many months or often multiple years to develop, during which the requirements are subject to significant iteration as the business environment changes, business needs change, and new requirements are identified.

Cite this column as follows: Donald Firesmith: "Prioritizing Requirements", in *Journal of Object Technology*, vol. 3, no. 8, September-October 2004, pp. 35-47.
http://www.jot.fm/issues/issue_2004_09/column4

- **Small RE budget**. Requirements engineering rarely receives more than 2-4% of the project budget, although several studies show that projects are more successful when 3-4 times as much of the budget and schedule is invested in getting the requirements right. Thus although requirements activities and tasks are typically time-boxed, the boxes allocated to requirements engineering are typically much too small and work must be properly prioritized.

These facts of life make the prioritization of requirements a critically important part of requirements analysis that every requirements engineer must perform. Unfortunately, there is little agreement within industry as to how, when, and why requirements should be prioritized. In fact, some books provide no guidance beyond merely stating that prioritizing requirements is important [Schneider 1998]. Hopefully, this column will help clarify requirements prioritization and make it easier for the reader to prioritize requirements and effectively use their priorities.

2 POSSIBLE MEANINGS OF PRIORITY

A fundamental problem with prioritizing requirements is that the phrase "prioritizing requirements" can have very different meanings to different stakeholders. So let's start by looking in the dictionary. According to the Merriam-Webster Online Dictionary, the term "priority" means:

1. The state of being prior (i.e., given precedence in terms of date or time)
2. Given or meriting attention before competing alternatives
3. Given preference

Although all three of the preceding definitions are closely related, definitions 1 and 2 are very different from definition 3 in terms of their impact on how and why requirements are prioritized and on how the resulting prioritizations are used. The first two definitions deal with scheduling, whereas the third definition deals with relative importance. And it is not unreasonable to schedule based on more than importance (as is implied by rate monotonic scheduling and the prioritization dimensions listed in section 5).

Some authors (e.g., [Sommerville 1997]) recommend another approach. For them, prioritizing requirements means categorizing raw *potential* requirements from the standpoint of importance into:

1. **Essential requirements** that must be included in the system (i.e., the actual requirements)
2. **Useful capabilities** that would reduce system effectiveness if left out
3. **Desirable capabilities** that make the system more desirable to certain stakeholders

Although this categorization implies that some requirements are merely useful capabilities or desirable "nice-to-haves", it also contradicts the very nature of requirements as being mandatory or required. Clearly, only the first category of potential

requirements above contains real requirements. The rest, although useful information, are not requirements and are thus outside the scope of prioritizing *requirements*.

Based on the preceding, prioritizing real requirements could mean:

1. **Prioritization by implementation order**. Prioritizing requirements is the requirements task of determining the implementation order of the requirements in an incremental and iterative development cycle.
2. **Prioritization by importance**. Prioritizing requirements is determining the order of importance to some stakeholder or class of stakeholders of the requirements along one or more dimensions (e.g., personal preference, business value, cost of implementation, and risk).

The remainder of this column is largely an argument that requirements prioritization is determining the implementation order of requirements and that prioritization by importance is merely one means to that end.

3 PURPOSE AND BENEFITS

The *purpose* of requirements priority can thus be to:

- Determine the relative necessity of the requirements. Whereas all requirements are mandatory, some are more critical than others. For example, failure to implement certain requirements may have grave business ramifications that would make the system a failure, while others although contractually binding would have far less serious business consequences if they were not implemented or not implemented correctly.
- Help programs through negotiation and consensus building to eliminate unnecessary potential "requirements" (i.e., goals, desires, and "nice-to-haves" that do not merit the mandatory nature of true requirements).
- Schedule the implementation of requirements (i.e., help determine what capabilities are implemented in what increment).

Properly prioritizing requirements provides the following significant *benefits* to the project:

- **Modify schedule**. When using an iterative incremental development cycle, it enables the project manager and customer to modify the project schedule to deal with the project realities of limited resources and fixed deadlines.
- **Improved customer satisfaction**. It improves customer satisfaction by increasing the likelihood that the customer's most important requirements are implemented and delivered first.
- **Lower risk of cancellation**. The project is less likely to be cancelled during development because valuable progress is being demonstrated with each increment. And even if the project must be cancelled before the delivery of the

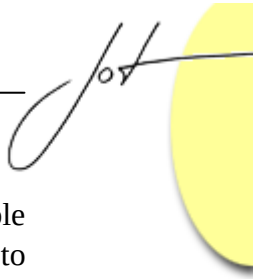
final increment, it is not a total loss because some important functionality has been implemented and delivered.

- **Address all requirements**. Prioritizing requirements is a good way to force stakeholders to address all requirements and not just their own.
- **Estimate benefits**. Priorities provide management and engineering with a rough estimate of the benefit of the different requirements, which is useful when performing cost/benefit analyses of the requirements to determine where best to expend limited project resources in preparation for requirements negotiation.
- **Prioritize investments**. Requirements priorities can help determine how to prioritize the investment of limited project resources. For example, the project can allocate most of its limited resources for quality assurance and system testing according to the highest priority requirements.

4 CHALLENGES AND RISKS

There are numerous challenges and associated risks that must be addressed when prioritizing requirements including:

- **Mandatory nature of requirements**. By definition, all requirements are required (i.e., mandatory). This leads some stakeholders in the requirements to believe that all requirements should have the same highest priority. However, although all real requirements are mandatory at a given instant in time, they are not all equal in terms of current importance or value to the customer or other stakeholders. Even when they admit that in theory different requirements can have different priorities, they may still strongly push for having 85-90% of the requirements be classified as high priority, thereby eliminating the benefits of prioritization [Wiegers 1999]. Priorities should have a reasonable distribution [Wiegers 2000], although enforcing a strict distribution can also lead to problems (e.g., as when grading on the curve in a class of legitimate A students).
- **Large number of requirements**. A very large number of requirements need to be prioritized. It is not unusual to have hundreds of requirements, and very large systems and systems of systems often have thousands of individual requirements. It is difficult to consistently prioritize such a large number of requirements. This is why priorities are often grouped into a manageably small number of categories. This is also why techniques for determining the priorities of all requirements such as pair-wise comparisons or Quality Function Deployment (QFD) typically do not scale unless requirements are previously grouped in some manner (e.g., by major system function or by limiting the requirements to those that are needed for the very next release). Another approach is to only apply semi-quantitative prioritization schemes to requirements of intermediate priority [Wiegers 1999].



- **Limited resources**. Because of cost and schedule limitations, it is rarely possible to implement all requirements in any given increment. It is also difficult to determine how much of the project's limited resources are worth expending to implement the different requirements.
- **Quality requirements**. The architecture and costs, both development and maintenance, of most systems is largely driven by such quality requirements as availability, interoperability, performance, portability, reliability, safety, security, and usability. Unfortunately, these types of requirements are often given far too little priority, are not specified at all, or are specified in a vague untestable manner. If the quality requirements are not specified or not specified properly, they will not be properly prioritized.
- **Goals vs. requirements**. System and software requirements are typically multiple levels below the business goals and needs that drive them. Thus, it is often difficult to directly relate requirement priority to business goal importance.
- **Changing priorities**. The priorities of requirements will typically change over time because:
  - The business environment and needs change.
  - The stakeholders in the requirements may change.
  - The requirements stakeholders change their minds as to which requirements are most important to them, especially once they understand the cost and schedule implications [Wiegers 1999].
  - Individual requirements change.
  - The system may be incrementally developed so that some of the requirements are implemented before others. Thus, the important priorities become the priorities of those remaining requirements that have yet to be implemented.
  - As new requirements are added, the relative priorities of existing requirements may need to change accordingly [Fellows 1998].
- **Incompatible priorities**. Different types of stakeholders tend to prioritize requirements differently (e.g., they tend to prioritize use cases higher when they are the actor that benefits from the execution of the use case). Even different stakeholders within the same stakeholder type prioritize them differently because of their different individual needs, experiences, and levels of training.
- **Stakeholder and developer collaboration**. Only the stakeholders can properly prioritize the requirements, while only the developers can properly estimate the cost and schedule consequences of the stakeholders' priorities [Wiegers 1999]. This requires the stakeholders and developers to collaborate during requirements prioritization, and this may be difficult due to contractual and organizational factors.
- **Incompatible requirements**. Some requirements types may be incompatible (e.g., performance vs. maintainability and reliability, usability vs. security) in the

sense the increasing compliance with one requirement makes it more difficult to achieve the other requirement.

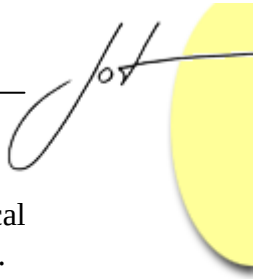
- **Lack of trust**. Customers desiring the immediate implementation of all requirements might mistakenly assume that the real reason to prioritize the requirements is the developers' devious desire to eliminate some of the more difficult or risky requirements.
- **Non-requirements**. The initial set of raw unanalyzed requirements prior to prioritization and negotiation often includes desired capabilities or items from somebody's wish list that are not really requirements (i.e., not mandatory) and thus do not deserve a priority.
- **Subjective prioritization**. Most prioritization approaches are subjective, biased, and influenced by project politics. They also ignore the reasons why stakeholders set their priorities the way they have. However, more objective approaches (such as the number of other requirements that depend on a requirement, the number of objects and stakeholders that interact with a requirement, the estimated cost of implementing the requirement, numerical weightings, etc.) are often expensive, impractical, and do not scale well when applied to large numbers of requirements.
- **Consequences of poor prioritization**. Incorrectly prioritizing and scheduling requirements for implementation can lead to serious financial consequences [Davis 2003] as well as significant stakeholder dissatisfaction.

5 TECHNIQUES

Prioritization Dimensions

Requirements can be prioritized along many different, related and even opposing dimensions. And these dimensions can be valued differently by different stakeholders. For example, requirements can be prioritized by:

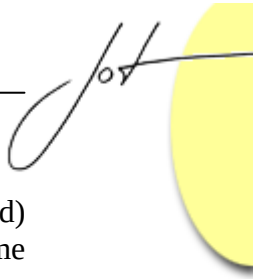
- **Personal preference.** Different stakeholders (e.g., customers, users, marketing, operators, maintainers, and architects) will prefer certain requirements over others. This is especially true when practical reasons such as schedule and budget mean that all of the requirements cannot be implemented and released during the current build of an incremental development cycle.
- **Business value.** When implemented, different requirements will have different values to the business. Some requirements will be critical, whereas others will be less important though still mandatory. Some potential requirements are not requirements at all but merely desirable though not necessary features or characteristics, and others will be merely characteristics that would be nice to



have or items on someone's wish list. Also, some requirements have a tactical usefulness, whereas others have a more long-term strategic value to the business.

- **Harm avoidance.** The opposite of prioritizing requirements in terms of their business value when implemented is to prioritize requirements in terms of the harm that can or will occur if the requirement is not implemented. This would especially be true of safety and security requirements, which are specifically specified to avoid accidental and malicious harm to valuable assets due to hazards and threats of attack respectively. Also, stakeholders who are risk adverse would tend to prioritize all requirements in terms of the damage or danger to be avoided if the requirement is implemented
- **Risk.** Related to harm avoidance is risk management. It may well make sense to prioritize requirements by the risks associated with their implementation. For example, one can attempt to implement those requirements having the highest risk first so as to deal with the resulting problems during development. On the other hand, it may make sense to implement the lowest risk requirements first in order to maximize the amount of the system implemented by ensuring that limited resources are not wasted on trying to implement high risk aspects of the system that may be impossible to successfully implement. Postponing the implementation of high risk requirements can also maximize the time available to research the risks and determine appropriate risk mitigation approaches.
- **Cost.** The implementations of different requirements have different development or life-cycle costs. Given limited budgets, cost can be an important and even overriding factor when prioritizing requirements. Thus, the highest priority requirements may be those that the project can afford to implement first.
- **Difficulty.** Related to prioritizing requirements by risk and cost is prioritizing requirements by their estimated difficulty to implement. As with risk, one can implement either the difficult or the easy requirements first, based on whether one considers it more important to deal with difficult requirements first or to hold off on the most difficult so that a larger number of easy requirements can be implemented first.
- **Time to market.** Some requirements take more effort and thus more calendar time to implement given limited development resources. In certain application domains in which competitors are marketing competing systems, time to market can be an important factor when prioritizing requirements.
- **Requirements stability.** Some requirements are relatively stable whereas others are very much subject to change during development. To minimize unnecessary rework, it may well make sense to implement stable requirements first and hold off on the implementation of the most volatile requirements until late in the development cycle.

- **Dependencies among requirements.** Certain requirements depend on other requirements [Davis 2003]. For example, requirements at a lower tier in the overall system structure "implement" requirements on a higher tier. Thus, software requirements implement subsystem requirements which implement system requirements. Dependency relationships between use cases and usage scenarios imply dependencies between their priorities. Similarly, the interaction and postcondition requirements of a use case implement the overall requirement of the use case. Derived requirements are engineered to support more fundamental requirements, which depend on the implementation of the derived requirements. Similarly, certain secondary requirements support core requirements that are essential to the success of the system and should be prioritized in terms of how they support these key requirements. In all of these cases, the implementation of certain requirements depends on the implementation of other requirements and this implies dependencies on their priorities. Thus, if requirement A depends on requirement B, then the priorities of these requirements should be consistent and requirement B should have a priority that is at least as high as A (e.g., B should be implemented either before A or during the same build as A).
- **Implementation dependencies.** When developing large systems, certain components of the system depend on other components (often foundational and infrastructure components) of the system. Requirements pertaining to these foundational components often need to be implemented before other requirements. Similarly, certain capabilities (e.g., safety and security) need to be architected and built into the system rather than added on later during development. Therefore, architects (who are important stakeholders of requirements) often prioritize requirements in terms of the optimum implementation order of the requirements.
- **Different kinds of requirements.** Different kinds of requirements (e.g., functional, data, interface, quality, and constraints) may need different approaches to prioritization. Non-functional requirements may be prioritized directly whereas functional requirements may be prioritized indirectly via their use cases and scenarios.
- **Legal mandate.** Requirements may be given higher priority if mandated by law, by regulation, or by governmental, international, national, or industry standard [Wiegers 2000]
- **Frequency of use.** Functional requirements may be given higher priority based on the expected frequency or volume of usage [Wiegers 2000].
- **Reuse.** If a requirement is highly reusable within a product line, then it might be wise to give it a higher priority so that no system within the product line has to wait for its implementation.



As the preceding list illustrates, there are many factors that can (and probably should) influence the priority of a requirement. One of the common mistakes that some approaches make is to consider and use only a single dimension when prioritizing requirements. For example, eXtreme Programming tends to only consider business value as defined by the customer [Beck 2001]. Although more difficult, it makes far more sense to consider all relevant factors when prioritizing requirements, even though some dimensions will prove far more important than others, at least for certain requirements.

Prioritization Approach

Once the actual requirements have been identified, prioritization can then be used to categorize them for the sake of scheduling into:

- Requirements that have already been implemented
- Requirements that are being implemented during the current build, increment, or release
- Requirements that are to be implemented during the next build, increment, or release
- Requirements that are to be implemented during some future build

To avoid requirements churn and meet schedule, the set of requirements being implemented as part of the current increment must be well known and frozen reasonably early during the build process and prior to release. But the farther into future increments one goes, the more fluid the assignment of priorities to requirements and requirements to increments becomes. In fact, some mandatory requirements may never be implemented because they never bubble up to the top of the stack before eventually being dropped because they are no longer needed.

Prioritization Techniques

Various techniques can be used to determine, negotiate, and develop a consensus regarding the priorities of the requirements:

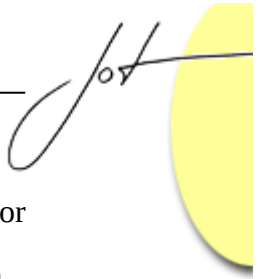
- Business Case Analysis / Return On Investment (ROI) estimation
- Pair-wise comparisons
- Prioritization working groups
- Scale of 1-to-10 rankings
- Voting schemes (e.g., give each stakeholder a specific number of votes to distribute amongst the requirements or classes of requirements being prioritized)
- Weightings (e.g., weight the votes of different stakeholders)
- Value-Based Software Engineering [Boehm 2003]
- WIN-WIN [Boehm 2001]
- Quality Function Deployment (QFD)

The best approaches to use also depend on many factors such as the number of requirements to be prioritized and the formality of the requirements engineering process.

6 REQUIREMENTS PRIORITIZATION PROCESS

Given the need to properly prioritize requirements, I recommend the following basic subprocess (related to the process in [Fellows 1998]) to incorporate requirements prioritization into the requirements engineering process:

1. **Convince stakeholders.** During requirements planning, the requirements team needs to convince the stakeholders in the requirements engineering process of the importance of prioritizing requirements.
2. **Train stakeholders.** Prior to eliciting requirements, the requirements team should train the requirements stakeholders in the requirements prioritization process.
3. **Categorize raw potential requirements.** During *requirements identification* (a. k. a., elicitation, discovery, invention, gathering), the requirements team should work with the stakeholders to categorize the *raw potential* requirements into actual requirements, useful capabilities and desirable (nice-to-have) capabilities so that the actual requirements can be prioritized.
4. **Prioritize the actual requirements**. During *requirements analysis*, the requirements team should work closely with representative stakeholders to prioritize the *actual* (i.e., mandatory) requirements. This includes negotiation with the stakeholders to develop a consensus and validation of the resultant priorities with them. Requirements prioritization should be done on an iterative and incremental basis, concentrating on those requirements that are most likely to need to be implemented in the current or next release and those requirements that are of intermediate priority and thus most likely to need significant negotiation. To develop proper priorities, representatives from all major stakeholder groups need to be represented. For the requirements team led by a professional requirements engineer, this should include one or more dedicated customer representatives, user representatives, architects, system testers, and subject matter experts. Other stakeholders that also need to be involved in requirements prioritization include operators, maintainers, safety engineers, security analysts, and any others that have a significant stake in the scheduling of requirements implementation.
5. **Publish the priorities**. During *requirements specification*, the requirements team should publish the priorities so that all effected stakeholders know and can use the priorities.
6. **Estimate effort.** Led by the technical leader and architecture team, the development team that must actually implement the requirements creates and records realistic estimates of the effort required to implement each requirement. These estimates should be based on current staffing, with the understanding that stakeholder inputs may require management to increase staffing size (but only if



the technical leader agrees that increasing staff will actually decrease schedule or increase deliverable functionality).

7. **Schedule development.** The requirements team should work with the management team and the development team to allocate requirements to increments and to schedule the incremental implementation of these increments based on priorities of the requirements, the required effort to implement the requirements, and the available resources. This allocation of requirements to releases and scheduling of releases should be updated with each increment to deal with changes in requirements and resources. Allocation and scheduling should also include both development-internal increments as well as releases to the customer organization and deployment to the users.
8. **Maintain priorities.** During *requirements management*, the requirements team should work with the requirements stakeholders to maintain the requirements parameters as they change. This will typically include storing the priority as metadata (i.e., an attribute) in the requirements repository, and then updating the value of the priority as it changes.

7 CONCLUSION

As the preceding information has hopefully demonstrated, prioritizing requirements is both a critical and yet difficult task for the requirements engineering team. Many risks, challenges, and issues must be properly taken into account if a useful set of priorities is to be developed, negotiated, maintained, and used. Still, requirements prioritization is critical because it:

- Forces stakeholders to openly address the relative importance of their requirements,
- Leads to increased communication and consensus among stakeholders,
- Provides a logical basis for requirements negotiation, and most importantly
- Enables management and engineering to rationally schedule of the development and release of large complex software-intensive systems when using an incremental, iterative, time-boxed development cycle.

REFERENCES

[Beck 2001] Kent Beck and Martin Fowler, *Planning Extreme Programming*, Addison-Wesley, 2001, pp. 48-49 and 63-69.

[Boehm 2003] Barry Boehm, "Value-Based Software Engineering," *Software Engineering Notes*, Vol. 28, No. 2, ACM, March 2003.

[Boehm 2001] Barry Boehm, Paul Grünbacher, and Robert O. Briggs "Developing Groupware for Requirements Negotiation: Lessons Learned," *IEEE Software*, Vol. 18, No. 2, IEEE, May/June 2001.

[Davis 2003] Alan M Davis, "The Art of Requirements Triage," *Computer*, Vol. 36, No. 3, March 2003, pp. 42-49.

[Fellows 1998] Larry Fellows and Ivy Hooks, "A Case for Priority Classifying Requirements," *Eighth Annual International Symposium on Systems Engineering*, Seattle, Washington: International Council on Systems Engineering, 1998.

[Lauesen 2002] Soren Lauesen, *Software Requirements: Styles and Techniques*, Addison-Wesley, 2002, pp. 2226-227,304, 378-379.

[Schneider 1998] Geri Schneider and Jason P. Winters, *Applying Use Cases: A Practical Guide*, Addison-Wesley, 1998, pp. 84.

[Sommerville 1997] Ian Sommerville and Pete Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.

[Wiegers 1999] Karl E. Wiegers, "First Thing First: Prioritizing Requirements," *Software Development*, September 1999.

[Wiegers 2000] Karl E. Wiegers, "Karl Wiegers Describes 10 Requirements Traps to Avoid," *Software Testing & Quality Engineering*, January/February 2000.

ACKNOWLEDGEMENTS

Many thanks to Ian Alexander, Keith Collyer, Andrew Gabb, David Gelperin, Rolf Goetz, Simon Hutton, Naveen Kumar, Frank Moisiadis, Scott Overmyer, Abd-El-Kader Sahraoui, Emmie Lou Tucker, and Ricardo Valerdi from the requirements engineering mailing list at re-online@it.uts.edu.au; their observations and recommendations provided many useful insights regarding the proper prioritization of requirements. I would also like to thank my coworkers Peter Capell and Tim Morrow at the SEI who reviewed this column prior to publication for their helpful comments and suggestions.

Disclaimers

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

The views and conclusions contained in this column are solely those of the author and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

About the author

Donald Firesmith is a senior member of the technical staff at the Software Engineering Institute. He has worked exclusively with object technology since 1984 and has written 5 books on the subject. He is currently writing a book on the engineering of safety requirements. Most recently, he has developed an 1100+ page informational website on the OPEN Process Framework at http://www.donald-firesmith.com. He can be reached at dgf@sei.cmu.edu.