A PRICE Systems Thought Leadership Article

By Arlene F. Minkiewicz, Chief Scientist, PRICE Systems

Estimating Software from Requirements

Abstract

With software, the process of going from requirements to architecture is lengthy and often quite involved. Once you have completed the process, you may have invested considerable time and resources into the project. For this reason, it is often not practical to wait until an architectural view is available to start estimating the cost of your software development project. Clearly an estimate based only on an understanding of what the software is going to do, without a clear picture of exactly how the software will accomplish it, is not going to meet all of your estimation needs. However, if properly done, it should offer enough information to make it possible to early identify those software projects that have the best chance of delivering what is needed when it is needed. It should also offer a sound basis for many of the decisions that need to be made by senior management early in the software lifecycle.

This paper provides a strategy for adapting what we know about estimating from an architectural view to accomplish reasonable estimates earlier in the process. It explores the information that is available early in the process and presents techniques for utilising this information in the context of existing estimating tools and techniques.

Introduction

You have decided to build a new home. You call the builder and ask how long it will take. The builder, if he's any good, will ask a few questions before answering this question. He may ask where you want to build this house, how many people will be living in the house, how many cars you will want to park in the garage, whether you want a family room and a basement, and what kind of a budget you have. The builder is gathering your requirements from this conversation in order to provide you with an estimate of the amount of time and money it will take to build your house. While the builder is asking you these questions, in his mind he is building an architectural view of your house – picturing a rancher with 3 bedrooms, a kitchen, living room, dining room and a two car garage. From this architectural view, the builder is than able, based on his experience building similar houses of the desired quality in the location you have indicated, to give you an estimate of how long it will take to build this house. Starting with your requirements for a house, the builder has constructed an architectural view of the house and based his estimate on that (See Figure 1).

The process of building software is, in many ways, like the process of building a house. The software development team talks with all the interested parties and gathers the product requirements. They then take these requirements, group them logically and develop a software requirement's document. Once this document is complete (or as complete as a software requirement's document ever gets in the early stages of a project), decisions can be made about implementation and an architectural view of the software is constructed. From this architectural view, the software project manager can make a realistic estimate of what the cost and schedule for the development project will be.

The difference between building a house and building software lies in the substantial differences in complexity and variety between software systems and homes. While homes come in many different types and sizes, there is a finite boundary on the typical differences and an experienced builder is able to make that jump from requirements to architecture fairly easily. In the case of software this is

not true. As long as hardware technology continues to improve, the possibilities for software are almost infinite. This means that this transition from requirements to architecture is rarely smooth and can not (except in the simplest of cases) be done in the project manager's head or even on a piece of scratch paper.

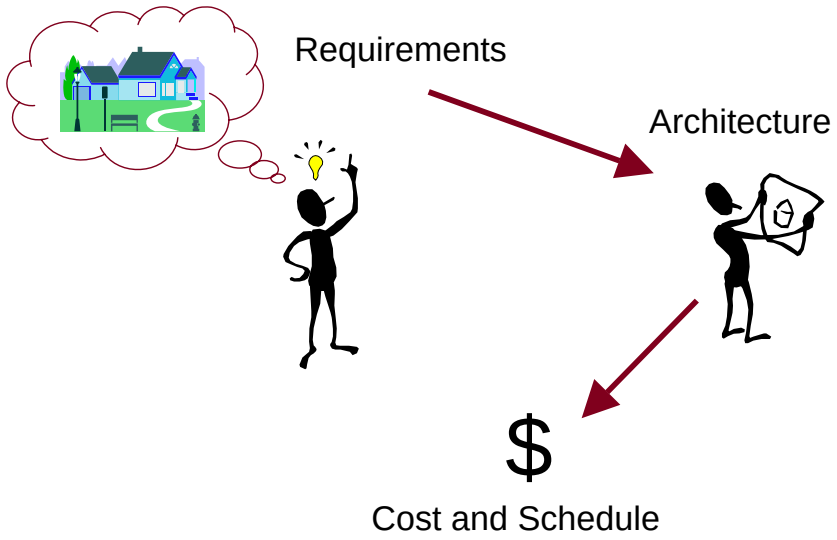

Figure 1 – Requirements to Cost

With software, the process of going from requirements to architecture is lengthy and often quite involved. Once you have completed it you may have invested considerable time and resources into the project. For this reason, it is often not practical to wait until an architectural view is available to start estimating the cost of your software development project. Obviously, an estimate based only on an understanding of what the software is going to do, without a clear picture of exactly how the software will accomplish it, is not going to meet all of your estimation needs. If properly done, it should, however, offer enough information to make it possible to identify early, those software projects that have the best chance of delivering what is needed when it is needed. It should offer a sound basis for many of the decisions that need to be made by senior management early in the software lifecycle. This paper provides a strategy for adapting what we know about estimating from an architectural view to accomplish reasonable estimates earlier in the process. It explores the information that is available early in the process and presents techniques for utilising this information in the context of existing estimating tools and techniques.

Software Project Estimation

There are many reasons why a cost estimate is performed on a software project. Estimates are required to assist in decision making throughout the product development life cycle. They are required to aid project managers in proper planning, resource allocation, and tracking for the duration of a project or concurrent projects. They equip project managers with the wherewithal to negotiate with their management when there are disconnects between requirements and time to market demands that make success questionable. They offer ammunition for managing requirements creep. Estimates make return on investment analyses possible. They provide estimates to complete for on-going projects.

Some of these reasons for estimating demand more accurate results then others. An estimate to help decision-makers decide between several product development options requires less detail than an estimate performed to support the scheduling of an expensive product launch. The wise project manager uses an arsenal of estimating tools and techniques to navigate the many decisions required in the planning and execution of a software project, tailoring the degree of detail in the estimate (and thus the cost of performing the estimate) to the criticality of the need.

There are many ways in which project managers estimate the cost, effort and schedule associated with the software projects they are managing. Many use commercially available estimation tools. These tools will query the user for information about the software project and then determine cost and schedule estimates based on these inputs. Some tools have codified cost estimating relationships in their calculation engines while others use internally maintained databases of past projects to estimate by analogy. Some use a combination of both these techniques. Some project managers use 'home-grown' estimation tools – generally spreadsheets they have developed from past project experiences. Some project manager's estimate from the bottom up – based on a fairly detailed functional decomposition of the solution space. Some project managers guess based on personal historical perspective. Some just guess and hope for the best. All of these techniques, except possibly the last one, require that the project manager understand the problem and its proposed solution to some level of detail that includes implementation decisions and fairly detailed information about solution architecture. This is often more information than is needed or available early in the decision making process

The Requirements View vs. Architectural View

Long before we understand what components we will need to build a software system, we understand what the software system is expected to do. Software products begin with a concept – someone says "Wouldn't it be great if I had a software program that maintained my checkbook". After consensus is gained that this would be great – the next step is to gather product requirements. Product requirements describe all the behaviors (and some non-behavioral things as well) that the software must exhibit in order to fulfill the promise of the concept. Gathering product requirements requires talking to all those who have interest in the final product or who possess useful domain knowledge, including potential users, sales people, marketing personnel, IT personnel, etc. Once these requirements are gathered, they are then disassembled into lower level software requirements (some requirements may also be allocated to hardware but for simplicity we'll stick with systems that require software only). The software requirements are then grouped logically and used as the basis for the solution architecture. Although this process sounds pretty simple, if you are dealing with a complex software system (and most worth talking about are) there's a lot of work getting from the wish list to architecture.

At this point an example would be useful. For simplicity, the example project is small enough that the leap from requirements to architecture could actually be accomplished in one's head. The software item being described is a subset of the administrative functionality required for a bank computer system. Although data storage and retrieval is required, it is assumed that these are being accomplished through database implementations that exist already. The behavioral requirements for this system follow:

- Requirement 1: The system shall allow updates to all customer and account information through the Customer Tracking Screen
- Requirement 2: The system shall update the other 3 display screens with the data entered through the Customer Tracking Screen.
- Requirement 3: The system shall provide reports detailing history of account transactions for individual customers
- Requirement 4: The system shall provide customer listing reports with account numbers and other customer information
- Requirement 5: The system shall be protected through user id and password combinations

These Product Requirements are then further decomposed into the software requirements necessary to satisfy them. These software requirements would then be grouped together, combining all of the similar functionality necessary for each requirement so that the necessary components to implement

these functions could be identified. In this case it was determined that the following architectural components will suffice to implement the requirements:

- Component 1: User Interface module – handles View and Change Data screens and Security.
- Component 2: Interface to account information database
- Component 3: Interface to customer database
- Component 4: Report module

It's important to note the ways that these two views of the software solution are alike and how they differ. The requirements tell us the things that the software needs to do to solve the problem but the architectural view gives us the first glimpse into how the software actually will implement that solution. It is even more important to note that it is very unlikely to find a one-to-one relationship between requirements and architectural elements – in fact it is much more likely to find that the functionality required to implement a single requirement comes from multiple architectural components.

Estimating from an Architectural View

The components listed above show the software architecture in the form required for many commercial software cost-estimating tools. In this, view certain parameters must be specified for each component in order for the tool to perform an estimate. In order to understand what information available at requirements time might be useful, we begin by determining what information from the architectural view has proven useful. The critical inputs required to perform an estimate for each component include the following parameters:

- ***Size of component*** - One of the most challenging aspects of estimating software is assigning a 'size' to the output of the software development process before that process has really started. People count (estimate) SLOC (Source Lines of Code), Function Points, Object Points, Number of Objects, etc. The earlier in the process this determination is made, the more important it is to rely on prior experience with similar products.

- ***Character of the component*** – Most estimating tools and techniques require some quantification of the complexity of the functionality implemented by the component. The reason this type of input is important is that the effort to design, code and test a component varies significantly based on whether it is expected to perform simple or complex processes.

- ***Project start date*** – There are two reasons why this date is important. It makes scheduling possible by providing a starting point. It also provides a basis for making a determination of the maturity (or immaturity) of the technology employed in this development process.

- ***Integration difficulty*** - For the purposes of the estimate, the software system is broken into components. This allows for more precise descriptions of the components. We all know that in order to complete the system all of these components need to work together. Depending on how many interfaces there are between the components and how much experience the integration team has with the software – this integration could be very simple or quite complex. This input provides information about where on the difficulty spectrum the component is expected to fall.

- ***Organisational productivity*** – This input provides quantification of the software development organisation's ability to develop software – determined through calibration or industry standards.

- ***Non-behavioural system requirements*** - In addition to understanding the functions that the software is expected to perform, there is also system level information that applies to the entire project. These requirements can impact the estimate significantly. These requirements include performance, reliability, maintainability, portability and security requirements.

Estimating from the Requirements View

We already know how to estimate from an architectural view and we know what our requirements are. Wouldn't it be great if we could map what we know about requirements into an architectural view and then use a familiar cost-estimation technique to aid decision making before all the architectural decisions have been made? In order to do this we need to determine what attributes must be defined for each requirement and how these attributes map to inputs required from the architectural model. We then need to figure out how to adjust integration costs so they properly reflect the costs of integrating across components rather than requirements even though we are using requirements to describe software functionality. The information that must be determined for each requirement includes:

- ***Size of requirements*** **-** In the same way that it is difficult to determine the 'size' of a component that has not yet been built; it is difficult to determine the size of a requirement that has not yet been implemented. The use of Function Points seems to offer a workable solution to this problem since requirements indicate the functionality expected of the software. It is important when sizing requirements to be aware of how far into the requirements process you are and to understand what that means within your organisation. It is important to understand the expected requirements growth from the point at which the estimate is being made in order to assess accurately the risk associated with the estimate performed.

- ***Duplication of functionality across multiple requirements*** **-** Just as there is a cost associated with each component in our system, there is a cost associated with each product requirement. The difference lies in the fact that with components there is fairly little overlap between components. Costs are not shared between components because each component is intended to implement one part of the desired functionality. With requirements this is distinctly untrue. Let's say that implementing Requirement 3 from above will take 200 person hours and that implementing Requirement 4 will take 220 person hours. However, it is clear that there is a great deal of overlap in the functionality required to implement these two requirements so if both requirements are being implemented for the same system, it will not take 420 hours to build both. It is much more likely that much of the functionality can be implemented once but used in both places. Any system intended to estimate from requirements needs a quantitative measure of how much functionality each requirement shares with other requirements. The cost needs to be shared by all the sharing requirements.

- ***Spread of functional complexity across requirements*** **–** Because multiple requirements can share functionality from multiple components, there is likely to be overlap in the characterisation of function complexity for requirements. The cost impact of this characterisation needs to be spread properly across the requirements.

- ***Integration difficulty*** **–** Requirements are not integrated together, the components that implement their functionality are. The cost of integration of architectural components can be a significant part of the cost of developing an entire system – particularly if the system is complex. It is important that we quantify the difficulty of integrating requirement functionality and then weight that difficulty so that it maps sensibly to the requirements in our breakdown structure.

This does not address all the parameters that need to be specified for a successful estimate, only those that require some sort of mapping in order to use the architectural structure of the cost model to describe a breakdown by requirements. There are many other input parameters to cost models that remain constant between the architectural and requirements view. Parameters like Operational Specification, Project Start Date, Organizational Productivity, Memory and Timing Constraints should be determined for the system and applied uniformly to each requirement.

Mapping from a Requirements View to Architectural View

In order to demonstrate how the mapping from a requirement view to an architectural view might be accomplished; let's revisit the simple example given earlier. The system requirements are as follows:

- Requirement 1: The system shall allow updates to all customer and account information through the Customer Tracking Screen
- Requirement 2: The system shall update the other 3 display screens with the data entered through the Customer Tracking Screen.
- Requirement 3: The system shall provide reports detailing history of account transactions for individual customers
- Requirement 4: The system shall provide customer listing reports with account numbers and other customer information
- Requirement 5: The system shall be protected through user id and password combinations

The first issue that needs to be addressed is the size of each of these requirements and how much of that size needs to be applied to each requirement element in a breakdown structure. The following table (Table 1) shows a method for assigning a size to each requirement that allows for elimination of the duplication of functionality amongst requirements.

The first step in this process is to determine the number of function points that each requirement will implement and which requirements shared some functionality with other requirements. The process that was applied to determine percent implemented for a specific requirement simply split the shared function points evenly across the requirements that share them. For instance, both report requirements used two of the same queries resulting in each requirement being allocated the cost responsibility for 50% of these particular function points.

Using this process, it was simple to determine the proper function point input for each requirement. Keep in mind that assumptions like this are in keeping with the desire for a higher level (and quicker, less costly and less accurate) estimate than would normally result from a more detailed analysis. Another point to consider is that this analysis may sometimes be undesirable if the point of the estimating exercise is to determine which requirements are to be included in a software product. In this case you would want to obtain a full understanding of the cost of each requirement in order to make return on investment decisions.

Table 1 – Function points for requirements

Requirement	Function Points	Requirement that share functionality	% Implemented for this requirement	FP Input for this requirement
1. Updates to Customer and Account Information through Customer Tracking Screen	49	2	61	30
2. Update 3 Customer Display screen	103	1	82	84
3. Reports of account history	52	4	77	40
4.Customer listing reports with account and customer information	40	3	70	28
5. Protection through user id and pw	6		100	6

Once the size has been determined for each requirement, the next step is to determine the functional complexity of the requirements. There are two ways this could be approached. One is to determine the functional complexity of the entire system by understanding the mix of various functions within the system and then using this value as an input parameter to each requirement. The other way is to spend the time to determine a value for each requirement and then rely on the size adjustment to

spread shared functionality correctly. Either of these approaches will also work for determining how to spread reuse of design and code across requirements with shared functionality.

Although the goal is to perform estimates that are relatively independent of implementation details, it is important to note at this point that some decision needs to be made prior to the estimate regarding the programming language, or at least the class of programming language, that is to be used to accomplish the software development. In order for an organisation to make effective use of historical information and account for productivity boosts offered by the latest technology, the estimating tool needs to have some measure of the sophistication of the development environment being employed.

The final hurdle to generating a requirements based estimate is the determination of a measure of the difficulty of integration. It is important to remember that the requirements are not being integrated but rather the components that implement the requirements need to be integrated. In order to estimate this cost adequately, we are required to have some level of knowledge of the likely architecture – although not to the level of detail normally required. If there is a general understanding of the number of architectural components at the time the count is being accomplished, then you can use this as a basis of the component count. If not, use the total system size, which you can determine by adding the adjusted function point counts from all the requirements. From this size the number of components can be estimated. This is an area where organisational history should be applied. The initial integration difficulty should then be based on the number of components. In the absence of organisational history, the following guidelines (Table 2) should be used to quantify integration difficulty on a scale of 0 to 1 where 0 indicates no integration is necessary and 1 indicates a very complex integration.

Table 2 – Quantifying Integration Difficulty

Number of components	INTEGE value
1	0
3	0.25
5-15	0.5
>15	0.7

Once an initial integration difficulty has been determined it should be adjusted up or down if there is additional knowledge of particular obstacles or easements. Items that would make the integration difficulty increase include very complex functionality, very tightly coupled modules, integration expected to take place across multiple locations or in different time zones. Factors that would decrease integration difficulty include very simple interfaces, consistent resources through development and integration phases, and very loosely coupled modules. Once adjustments have been applied, the integration difficulty factor should be applied equally to all the requirement elements.

Conclusions

The estimation of software costs is a complex and complicated process. In order to accomplish an accurate estimate it is necessary to have a fairly detailed understanding of how the software is expected to satisfy its requirements. This level of detail is not reached without a substantial investment into the system. At a certain point in a software project's lifecycle this investment is fully warranted. However, it is often not practical, nor desirable, to invest this much in a software system before an organization has actually committed to its' implementation, but there is still a need to estimate the cost and/or schedule for that system.

This is the quandary software project managers often face. How do we determine the feasibility of a particular software project without breaking the bank on a project that we may ultimately determine is an unacceptable choice based on the feasibility study? We need to find new ways to use our

estimating processes to support multiple types of decisions. One way we can do this is to determine a sensible mapping that allows us to adapt traditional estimating techniques for use earlier in the decision making process. While we obviously must accept the trade-off between accuracy vs. speed and cost of estimate, thoughtful analysis and incorporation of organisational historical data can result in estimates accurate enough to support critical decision making processes early enough in the development cycle to be useful to the decision makers.

About the Author

Ms. Minkiewicz leads the Cost Research Department as Chief Scientist at PRICE Systems. In this role, she is responsible for the research and analysis necessary to keep the suite of PRICE Estimating products responsive to current cost trends. She works with industry leaders to collect and maintain cost research data and offers analyses of this data to the cost estimating community through the PRICE products.

Arlene's most recent accomplishments include the development of a catalog of cost estimating relationships for hardware and systems projects that will be delivered to the cost estimating community as part of the TruePlanning suite.

Arlene frequently publishes articles on estimation and measurement in publications such as Software Development Magazine and Crosstalk. She speaks frequently on these topics at conference such as STC, ISPA, SCEA, IEEE Aerospace Conference, SEPG, and many others. Her 'The Real Costs of COTS-Based Software Systems' paper was recognized in 2004 by ISPA and SCEA as Best Paper in the Software Track. Her paper "A Case Study and Assessment of a COTS Upgrade for a Satellite Ground System", co-authored with Marilee Wheaton of the Aerospace Corporation, received Best Paper in Software Track in 2006 by SCEA and her paper "The Evolution of Hardware Estimating" received Best Paper in the Hardware and EVM Track at the 2007 ISPA/SCEA joint conference.

Arlene can be contacted at arlene.minkiewicz@pricesystems.com

More information on software cost estimating and the TruePlanning Suite can be found at: www.pricesystems.com

PRICE Systems World Headquarters
17000 Commerce Parkway, Mt. Laurel, NJ 08054
voice 1.856.608.7200 / fax 1.856.608.7247 / www.pricesystems.com
Please contact us at www.pricesystems.com/contact/contact.asp for more information.

PRICE Systems - Washington, D.C.
1700 N. Moore Street, Suite 1100, Arlington, VA 22209
voice: 1.703.740.0087 / fax 1.703.740.0088

PRICE Systems International
Meridian Office Park, Osborn Way, Hook
Hampshire RG27 9HY England
voice 44.1256.760012 / fax 44.1256.762122