Estimating Software Requirements

Seminar on Software Cost Estimation
WS 02/03

Presented by
Yong Xia
xia@ifi.unizh.ch

Requirements Engineering Research Group
Department of Computer Science
University of Zurich, Switzerland

Prof. M. Glinz
Arun Mukhija

Date: January, 14th 2003

Content

1. Background

Software requirements are the starting point for every new project, and are a key contributor to enhancement projects, as well. Software requirements are also very ambiguous, often filled with bad assumptions and severe errors, and are unusually difficult to pin down in a clear and comprehensive way.

From a software cost estimating standpoint, the most tricky part of estimating requirements is the fact that requirements are usually unstable and grow steadily during the software development cycle in the coding and even the testing phases.

The observed rate at which requirements change after their initial definition runs between 1 percent and more than 3 percent per month during the subsequent analysis, design, and coding phases. Equally as troublesome, software requirements are the source of about 20 percent of all software bugs or defects, and are the source of more than 30 percent of really severe and difficult defects.

A lot of researches talk about how to make software requirements accurate and complete. And recently many methods, languages, and processes are given to cope with this problem.

Although many works on requirements contain thoughtful analyses of requirements methods and offer interesting suggestions for improving software requirements gathering and analysis, the following two topics are seldom mentioned and studied.

1. Quantification of requirements sizes, schedules, effort, and costs
2. Quantification of requirements errors and defect-removal efficiency

The following part of paper gives an initial, maybe still incomplete, solutions on the above two problems. The results show the quantitative results of requirements sizes, schedules, and efforts for an average project developed by an average team, as shown in Section 2. Productivity rates for different types of the software systems and the attributes affecting the requirements estimates will be also presented in this section. Section 3 gives the quantitative results of requirements errors. Section 4 shows the results of SPR' method on evaluating combinations of requirements factors.

Note that this paper is based on the work of [Jones98]. Only ***very few*** ideas come from the author himself.

First let us review some concepts and methods, which are intensively used in this paper.

1.1 Function Points

The function point metric has proven to be a useful tool for gathering requirements, and also for exploring the impact and costs of creeping requirements. Recall that the function point metric is a synthetic metric derived from the following five attributes of software systems:

1. Inputs
2. Outputs
3. Inquiries
4. Logical files
5. Interfaces

1.2 Defect-prevention technologies

We will introduce some technologies, which prevent the requirements errors and defects, in this section. These technologies can also reduce the rate at which requirements change, or at least make changes less disruptive.

1.2.1 Joint application design

Joint application design (JAD) is a method for developing software requirements under which user representatives and development representatives work together with a facilitator to produce a joint requirements specification that both sides agree to.

Compared to the older style of adversarial requirements development, JAD can reduce creeping requirements by almost half. The JAD approach is an excellent choice for large software contracts that are intended to automate information systems.

In order to work well, JAD sessions require active participation by client representatives, as well as by the development organization. This means that JAD technology may not be appropriate for some kinds of projects. For example, for projects such as Microsoft's Windows 95, where there are many millions of users, it is not possible to have a small subset of users act for the entire universe.

The JAD method works best for custom software, where there is a finite number of clients and the software is being built to satisfy their explicit requirements. It does not work well for software with hundreds or thousands of users, each of whom may have slightly different needs.

1.2.2 Quality function deployment (QFD)

Quality Function Deployment (QFD) is a technique originally for exploring the quality needs of engineered products. Later, it is also applied in software system. Today, QFD is expanding globally, and many of high-technology clients who build hybrid products, such as switching systems and embedded software, have found QFD to be a valuable method for exploring and controlling software quality issues during requirements.

Procedurally, QFD operates in a fashion similar to JAD in that user representatives and design team representatives work together with a facilitator in focused group meetings. However, the QFD sessions center on the quality needs of the application rather than on general requirements.

The QFD method has developed some special graphical design methods for linking quality criteria to product requirements. One of these methods shows product feature sets linked to quality criteria. Visually, this method resembles the peaked roof of a house, so QFD drawings are sometimes termed *the house of quality*.

1.2.3 Prototype

Since many changes don't start to occur until clients or users begin to see the screens and outputs of the application, it is obvious that building early prototypes can move some of these changes to the front of the development cycle instead of leaving them at the end.

Prototypes are often effective in reducing creeping requirements, and they can be combined with other approaches, such as JAD. Prototypes by themselves can reduce creeping requirements by somewhere between 10 and about 25 percent.

There are three common forms of software prototypes:

- Disposable prototypes
- Evolutionary prototypes
- Time-box prototypes

Of these three, the disposable and time-box methods have the most favorable results. The problem with evolutionary prototypes that grow to become full projects is that during the prototyping stage, too many short cuts and too much carelessness is usually present. This means that evolutionary prototypes seldom grow to become stable, well-structured applications that are easy to maintain.

1.2.4 Use cases

The technique termed *use case* originated as a method for dealing with the requirements of object-oriented applications, but has subsequently expanded and is moving toward becoming a formal approach for dealing with software requirements.

The use case technique deals with the patterns of usage that typical clients are likely to have and, hence, concentrates on clusters of related requirements for specific usage sequences. The advantage of the use case approach is that it keeps the requirements process at a practical level and minimizes the tendency to add “blue sky” features that are not likely to have many users.

1.2.5 Change-control boards

Change-control boards are not exactly a technology, but rather a group of managers, client representatives, and technical personnel who meet and decide which changes should be accepted or rejected.

Change-control boards are often encountered in the military software systems domain. Such boards are most often encountered for large systems in excess of 10,000 function points in size. Change-control boards are at least twice as common among military software procedures as they are among civilian software producers. Within the civilian domain, change-control boards are more common for systems, commercial, and outsourced software projects.

In general, change-control boards occur within large organizations and are utilized primarily for major systems. The members of a change-control board usually represent multiple stakeholders and include client representatives, project representatives, and sometimes quality-assurance representatives. For hybrid projects that include hardware, microcode, and software components, the change-control board for software is linked to similar change-control boards for the hardware portions.

The change-control board concept has been very successful whenever it has been deployed, and tends to have long-range value across multiple releases of evolving systems. Change-control boards are now a standard best practices for the construction of large and complex applications, such as telephone switching systems, operating systems, defense systems, and the like.

1.3 Types of the software

Different types of software have different statistical results on requirements sizes, requirements costs, requirements errors, defect-removal efficiency, etc. Therefore, it is reasonable that we first make a short introduction on the different types of software in the view of estimating software requirement, and how requirements of these types of software are gathered, analyzed, and produced.

1.3.1 End-User Software

End-User Software is the application, which is being developed for the personal use of the developer.

Except for a few notes on possible alternatives, the requirements for end-user software exist primarily in the mind of the developer. Changes in end-user requirements usually have no serous implications.

1.3.2 Management Information System (MIS)

MIS projects usually derive software requirements directly from users or the users' authorized representatives.

For MIS projects the most effective methods for gathering requirements include JAD, prototypes, and requirements reviews.

The older method of gathering MIS requirements consisted of drafting a basic set of requirements more or less unilaterally by the client organization, and them presenting them to the software development organization. This method leads to a high rate of requirements creep, and also to adversarial feelings between the clients and the developers.

The requirements approach with the RAD methodology leads to a form of evolutionary prototyping without much in the way of written requirements. While the RAD approach is acceptable for small or simple applications, the results are usually satisfactory neither for large applications above 1000 function points, nor for critical applications with stringent security, safety, performance, or reliability criteria.

MIS requirements can begin by exploring either the functions that the software is intended to perform or the data that is intended to be utilized. On the whole, beginning the requirements by exploring data and defining the outputs appears to give the best results.

There are several commercial software systems, such as SAP R/3, Oracle, IBM, or Computer Associates products, which provide packages for setting up MIS. However, it is still desirable to match package capabilities against fundamental needs and requirements of MIS.

1.3.3 Outsourced Projects

Outsourced projects in the MIS domain are similar in style and content to normal MIS projects with two important exceptions:

1. Outsource vendors often apply a cost per function point rate to the initial requirements in order to give the clients a good idea of the costs of the project. Some modern

outsource contracts also include a sliding cost scale, so that the costs of implementing creeping requirements will be higher than the costs of the initial set of requirements.
2. Outsource vendor that serve many clients within the same industry often have substantial volumes of reusable materials and even entire packages available that might be utilized with minor or major customization. For certain industries, such as banking, insurance, telecommunications, and health care, almost every company uses software with the same generic feature sets, so reusable requirements are possible.

1.3.4 System Software

Here systems software is defined as software that controls a physical device, such as a computer, switching system, or aircraft controls.

Because of the close and intricate relationship between the hardware and software, many requirements changes in the systems software domain are due to changes in the associated hardware.

Requirements gathering in the systems software domain seldom comes directly from the users themselves. Instead the software requirements usually come in from hardware engineers and/or the marketing organization that is in direct contact with the users, although for custom software applications users may be direct participants in requirements sessions.

Requirements, and also specification methods, in the system software domain are closely linked to hardware requirements, and the approaches for the software and hardware domains overlap. Special representation methods, such as Petri nets or state-transition diagrams, are sometimes used in the context of systems software requirements, and even hardware representation methods, such as the Verilog design language, may be applied to software requirements.

Because quality is a key criterion for systems software, approaches that can deal with quality issues during the requirements phase are common practices for systems software.

The close linkage between hardware and software requirements makes quality function deployment (QFD), which is similar to JAD in structure but has the emphasis on the quality and reliability of the application, effective in the systems software domain.

The usual starting point for the analysis of systems software requirements is determining the functions and features that are needed by the system.

1.3.5 Commercial Software

Gathering requirements for commercial software has some unique aspects. For some kinds of commercial software products there may be hundreds, thousands, or even millions of possible users. There may also be many competitors whose software has features that might also have to be imitated.

These two factors imply that commercial software requirements seldom come directly from one or two actual clients. Instead, commercial software requirements may arrive from any or all of the following channels:
1. From the minds of creative development personnel
2. From customer surveys

3. From marketing and sales personnel
4. From sophisticated customer support personnel
5. From user associations, focus groups, and online product forums
6. From analyzing the feature sets of competitive packages and imitating the more useful competitive features

Because the six channels are more or less independent, the requirements for commercial software packages tend to be highly volatile.

1.3.6 Military Software

Military software requirements are usually the most precise and exacting of any class of software. The military form of requirements tends toward large, even cumbersome, requirements specifications that are about three times larger than civilian norms. Although these military requirements documents are large and sometimes ambiguous, the specificity and completeness of military software requirements makes it easier to derive function point totals than for any other kind of software application.

On the whole, military software requirements have somewhat more positive attributes than negative for major systems that affect national defense or weapons. For smaller and less serious projects, the military requirements methods are something of an overkill.

1.4 Positive and Negative Requirements Adjustment Factors

For estimating software requirements, schedules, effort, costs, and quality, both positive and negative factors must be considered.

1.4.1 Positive requirements factors

Among the positive factors that can benefit software requirements production by perhaps 10 percent for assignment scopes, production rates, and defect potentials may be found in the following:

- High client experience levels
- High staff experience levels
- Joint application design (JAD)
- Prototyping
- Quality function deployment (QFD)
- Use cases
- Requirements inspections
- Reusable requirements (patterns or frameworks)
- Requirements derived from similar projects
- Requirements derived from competitive projects
- Effective requirements representation methods

1.4.2 Negative requirements factors

Among the negative factors that can slow down or degrade the software requirements production by perhaps 5 percent, or that can raise defect potentials, may be found in the following:

- Inexperienced clients
- Inexperienced development team
- Novel applications with many new features

- Requirements creep of more than 3 percent per month
- Ineffective or casual requirements-gathering process
- Failure to prototype any part of the application
- Failure to review or inspect the requirements
- No reusable requirements

2. Requirements Estimates

In this section, we show how to derive the requirements sizes, schedules, effort, and costs from our statistical tables.

2.1 Nominal Default Value

From a software cost estimating viewpoint, the nominal or default values for producing software requirements specification are shown in Table 1.

Table 1. Nominal Default Values for Requirements Estimates

Activity sequence	Initial activity of software projects
Performed by	Client representatives and development representatives
Predecessor activities	None
Overlap	None
Concurrent activities	Prototyping
Successor activities	Analysis, specification, and design
Initial requirements size	0.25 U.S. text pages per function point
Graphics volumes	0.01 illustrations per function point
Reuse	10% from prior or similar projects
Assignment scope	500 function points per technical staff member
Production rate	175 function points per staff month
Production rate	0.75 work hours per function point
Schedule months	Function points raised to the 0.15 power
Rate of creep or change	2.0% per month
Defect potential	1.0 requirements defects per function point
Defect removal	75% via requirements inspections
Delivered defects	0.25 requirements defects per function point
High-severity defects	30% of delivered requirements defects
Bad fix probability	10% of requirements fixes may yield new errors

Let us see how to use this table by an example:

For a 1500-function point systems software project, the nominal or average requirements would be about 375 pages in size and would be produced by a team of three technical personnel (working with about the same number of client personnel). The effort would amount to about 9 staff months. The schedule would be about 3 calendar months.

More ominously, there would be about 1500 potential defects in the requirements themselves. About 30 percent of the requirements errors or bugs would be very serious, which would amount to about 450 high-severity requirements errors—more than any other source of serious error.

Since many companies are careless about attempting to remove errors or defects in software requirements, defect-removal efficiency levels against requirements are lower than for other sources of error and average only about 75 percent.

A nominal defect-removal efficiency of only 75 percent means that when the project is deployed there will still be about 375 requirements defects and about 112 high-severity defects still latent. Indeed, latent requirements defects comprise the most troublesome form of after-deployment defects in software systems, because they are highly resistant to defect-removal methods.

This is an estimation of the average companies. Among the more sophisticated software companies, in which the significance of requirements and requirements errors is recognized and the defect-prevention technologies are applied in the project, the value in that table will be different. For example, the combination of JAD sessions plus prototyping can reduce the rate of creeping requirements from 2 percent per month down to perhaps 0.5 percent per month. The details will be further discussed in the next sections.

2.2 Requirements Productivity Rates

Although the default or nominal values for requirements estimates provide a useful starting place, the range of variance around each of the default values can be more than 3 to 1. Other factors, such as whether the requirements are being created for a military or a civilian project, also exert an impact.

Table 2 illustrates some of the ranges in requirements productivity rates associated with various sizes and kinds of software projects.

Table 2. Ranges in Requirements Productivity Rates by Class of Software
(Each staff works about 128 hours a month)

Software class	Requirements productivity, FP/staff month	Requirements productivity, staff hours/FP
End user	1000	0.128
Commercial	200	0.640
Small MIS	175	0.750
Large MIS	75	1.710
Outsource	90	1.422
Systems	75	1.710
Military	35	3.657

2.3 Other attributes

Table 1 just shows the nominal default values for requirements estimates with some initial default assumptions. Software requirements also have a number of attributes associated with them which have impact on the estimation, including but not limited to the following. These attributes greatly influence the requirements estimates.

- **Performed by**. Clients, marketing staff, sales staff, engineering staff, systems analysts, programmers, quality-assurance staff, and software project mangers are the normal participants in requirements.

- **Formal methodologies.** Numerous requirements methods exists, including rapid application development (RAD), OO design, quality function deployment (QFD), joint application design (JAD), finite-state machines, etc. Since these methods affect both the error density and the productivity of requirements, it is useful to record which method is utilized, if any.
- **Requirements tools.** Software requirements automation plays an important role on software requirements estimation. For example, the Bachman Analyst Workbench and the Texas Instruments Information Engineering Facility (IEF) provide automatic derivation of function point metrics from software requirements.
- **Defect prevention methods.** Defect prevention methods, such as Prototype, QFD, JAD, etc. are effective in preventing requirements defects.

Different attributes result in different requirements estimates.

3. Requirements errors

Software requirements errors comprise about 20 percent of the total errors found in software applications, but comprise more than 30 percent of the intractable, difficult errors.

The following table shows the current U.S. averages for software defect origins, which are expressed in terms of defects per function point.

Table 3. Requirements Defects and Other Categories

Defect origins	Total defects per FP	High-severity defects per FP
Requirements	1.00	0.30
Design	1.25	0.50
Code	1.75	0.25
Documentation	0.60	0.10
Bad fixes	0.40	0.15
Total	5.00	1.30

Requirements changes cause similarly bad consequences as requirements errors. In a lot of research works, they are even considered as a type of requirements errors.

In the context of exploring creeping requirements, the initial use of function point metrics is to size the application at the point where the requirements are first considered to be firm. At the end of the development cycle, the final function point total for the application will also be counted.

For example, suppose the initial function point count for a project is 100 function points, and at delivery the count has grown to 125. This provides a direct measurement of the volume of creep in the requirements.

From analysis of the evolution of requirements during the development cycle of software applications, it is possible to show the approximate rates of monthly change. The changes in Table 4 are shown from the point at which the requirements are initially defined through the design and development phases of the software projects.

In table 4 the changes are expressed as a percentage change to the function point total of the original requirements specification. Table 4 is derived from the use of function point metrics, and the data is based on differences in function point totals between: (1) the initial estimated function point total at the completion of software requirements, and (2) the final measured function point total at the deployment of the software to customers.

Table 4. Monthly Growth Rate of Software Creeping Requirements

Software type	Monthly rate of requirements change, %
Contract or outsource software	1.0
Information systems software	1.5
System software	2.0
Military software	2.0
Commercial software	3.5

Note that the real margin of error might be higher (this way of statistics ignores of the requirements modifications–replacement of original requirements by new requirement, in which function points are unchanged), but even so it is still useful to be able to measure the rate of change at all.

It is interesting that although the rate of change for contract software is actually less for many other kinds of applications, the changes are much more likely to lead to disputes or litigation.

Since the requirements for more than 90 percent of all software projects change during development, creeping user requirements is numerically the most common problem of the software industry. A number of technologies are being developed that can either reduce the rate at which requirements change or, at least, make the changes less disruptive (Some research is also carried out in our research group RERG).

4. Evaluating Combinations of Requirements Factors

After Jones and his colleagues at Software Productivity Research (SPR) studied the combinations of requirements adjustment factors, SPR developed a useful method for showing how a number of separate topics interact.

SPR's method is to show the 16 permutations that result from changing 4 different factors that affect software requirements:

1. The use of, or failure to use prototypes
2. The use of, or failure to use joint application design (JAD)
3. The use of, or failure to use formal requirements inspections
4. The presence or absence of experienced staff familiar with the application type

Table 5. Sixteen Permutations of Software Requirements Technologies
(Data expressed in defects per function point; best-case options appear in boldface type)

	Defect potential per FP	Defect-removal Efficiency, %	Residual defects per FP	Rate of creep, % monthly
No prototypes No use of JAD No inspections Inexperienced staff	2.00	60	0.80	4.0
No prototypes No use of JAD No inspections **Experienced staff**	2.00	65	0.70	3.5
Prototypes used No use of JAD No inspections Inexperienced staff	1.50	70	0.45	1.5
No prototypes No use of JAD **Inspections used** Inexperienced staff	2.00	80	0.40	3.0
No prototypes **JAD used** No inspections Inexperienced staff	1.50	75	0.38	1.0
Prototypes used No use of JAD No inspections **Experienced staff**	1.50	77	0.35	0.9
No prototypes No use of JAD **Inspections used** **Experienced staff**	2.00	84	0.32	1.0
No prototypes **JAD used** No inspections **Experienced staff**	1.50	80	0.30	0.9
Prototypes used **JAD used** No inspections Inexperienced staff	1.00	77	0.23	0.6
Prototypes used No use of JAD **Inspections used** Inexperienced staff	1.50	86	0.21	0.6
No prototypes **JAD used** **Inspections used** Inexperienced staff	1.50	86	0.21	0.5

	Defect potential per FP	Defect-removal Efficiency, %	Residual defects per FP	Rate of creep, % monthly
No prototypes **JAD used** **Inspections used** **Experienced staff**	1.35	88	0.16	0.5
Prototypes used **JAD used** No inspections **Experienced staff**	1.00	87	0.13	0.3
Prototypes used No use of JAD **Inspections used** **Experienced staff**	1.50	94	0.09	0.3
Prototypes used **JAD used** **Inspections used** Inexperienced staff	1.00	94	0.06	0.2
Prototypes used **JAD used** **Inspections used** **Experienced staff**	0.70	97	0.02	0.1

In this table, SPR assumes fairly complex applications of at least 1,000 function points or 125,000 C statements in size. For smaller projects, requirements defects and rates of change would be less, of course. For really large systems in excess of 10, 000 function points or 1,125,000 C statements, requirements errors would be larger and removal efficiency would be lower.

The table shows polar extreme conditions; that is, each factor is illustrated in binary form and can switch between best-case and worst-case extemes.

Note that the function point (FP) values used in the table assume the IFPUG Version 4 counting rules.

As can be inferred from the 16 permutations, software requirements outcomes cover a very broad range of possibilities. The combination of effective requirements-gathering technologies coupled with effective defect-removal technologies and a capable team lead to a very different outcome from casual requirements methods utilized by inexperienced staff.

The best-in-class technologies for dealing with requirements are highly proactive, and include the following components:

- Formal requirements gathering, such as JAD
- Augmentation of written requirements with prototypes
- Use of requirements-automation tools
- Use of reusable requirements from similar or competitive projects

5. Conclusion

If the initial requirements for a software project are done well, the project has a fair chance to succeed regardless of size. If the requirements are done poorly and are filled with errors and uncontrolled changes, the project has a distressingly large chance of being canceled or running out of control.

6. References

[Jones98] Jones, C. (1998), *Estimating Software Costs*, Chapter 17, New York: McGraw-Hill

[Jones02] Jones, C. (2002) : "Software Cost Estimation in 2002", Utah: Software Technology Support Center, Hill Air Force Base, http://www.stsc.hill.af.mil/crosstalk/2002/06/jones.pdf