Editor: Capers Jones, Software Productivity Research Inc., 1 New England Executive Park, Burlington, MA 01803-5005; phone (617) 273-0140; fax (617) 273-5176; capers@xanadu.spr.com or 7543.231@compuserve.com

Strategies for managing requirements creep

Capers Jones
Software Productivity Research

One of the most chronic problems in software development is the fact that application requirements are almost never stable and fixed. Frequent changes in requirements are not always caused by capricious clients (although sometimes they are). The root cause of requirements volatility is that many applications are attempting to automate domains that are only partly understood. As software design and development proceeds, the process of automation begins to expose these ill-defined situations. Therefore, although creeping requirements are troublesome, they are often a technical necessity.

Several threads of research and some emerging technologies are aimed at either clarifying requirements earlier in development or minimizing the disruptive effect of changing requirements later.

Several emerging technologies are aimed at either clarifying requirements earlier or minimizing the disruptive effect of changing requirements later.

Why requirements change

Requirements change because software is expanding the ways companies operate. In a sense, creating software requirements is like hiking in a gradually lifting fog. At first only the surroundings within a few feet of the path are visible, but as the fog lifts, more and more of the terrain can be seen.

Not all software applications have unstable requirements, but the majority do. Certain kinds of engineering and scientific software, and software that is embedded in physical devices (such as automotive fuel injection systems), may reach a point of stability early. In fact, once these requirements stabilize, they may stay constant for several years.

However, when software deals with business factors, change is almost inevitable. Suppose your company is building an inventory-management system. While your system is being built, you acquire another company with a different product line. Obviously this will affect inventory management. Other significant changes can be caused by external factors over which the software team and even the clients have no control, such as changes in the tax law or changes derived from business process reengineering studies.

Changes in the requirements for commercial software are sometimes driven by competitive pressures. Look at today's word processors and spreadsheets. Most exhibit "functional overkill" because every time one vendor develops a new feature all the others quickly imitate it. The result is that commercial word processors and spreadsheets have ballooned from fewer than 300 function points 10 years ago to more than 5,000 function points today.

Changes in military software can be caused by changing mission requirements or hardware platforms. In addition, a significant number of military applications must interface with such a complex web of other applications that many changes in system A are due to changes in systems B or C or N, which share data with system A.

Measuring the rate of change

The function point metric has proven to be useful for exploring the impact and cost of creeping requirements. The function point is a synthetic metric derived from five external attributes of software systems: inputs, outputs, inquiries, logical files, and interfaces.

To use function points to measure creeping requirements, first size the application when the requirements are first considered to be firm. Then size the application at the end of development. For example, a project might have 100 initial function points and 125 at delivery. This provides a direct measurement of the volume of creep.

Because the function point count is based on features requested and agreed to by the users or clients, it is unlikely that the development team would be able to add function points.

Of course, requirements changes will change the volume of source code as well, but while function points can be calculated from the requirements themselves, code size is a secondary or derivative factor that is harder to determine early in development.

By analyzing the evolution of requirements during development, you can show the approximate monthly rate of change. The changes are shown from the point at which the requirements are initially defined through the design and development phases of the software projects. The changes are expressed as a percentage change to the function point total of the original requirements specification.

There is a high margin of error in this data, but even so it is useful to measure the rate of change: Table 1 shows the monthly rate of change for the five domains.

Another way to express the change is to look at the average volume of change between the original requirements and the delivered application in terms of typical growth patterns derived from function point totals. Military software has a much more sluggish development cycle than civilian projects, which allows time for more changes to accumulate. Unfortunately, average values can be misleading. The maximum growth rate observed in many cases has exceeded 100 percent. One IBM systems software project I observed grew by more than 270 percent. Table 2 shows the cumulative growth rates for the five domains.

There are some seeming contradictions between the data expressed in terms of monthly change rates and the overall volume of change. The differences are due to the fact that schedules vary among the five domains. Although not the topic of this column, the speed with which software projects can be developed varies by domain. From fastest to slowest, the order is

1. contract or outsource software.
2. information systems.
3. commercial software.
4. systems software.
5. military software.

Table 1. Monthly requirements change for five software types. (source: *Applied Software Measurement,* Mc-Graw Hill, 1996)

Software type	Monthly rate of requirements change to function point total
Contract or outsource software	1.0%
Information systems software	1.5%
Systems software	2.0%
Military software	2.0%
Commercial software	3.5%

Table 2. Cumulative requirements change for five software types. (source: *Applied Software Measurement,* Mc-Graw Hill, 1996)

Software type	Requirements change to total function points
Contract or outsource software	14.45%
Information systems software	17.95%
Systems software	21.50%
Military software	25.54%
Commercial software	19.75%

Stabilizing requirements

Several technologies can either reduce the rate at which requirements change or make the changes less disruptive.

JOINT APPLICATION DESIGN. JAD is a method for developing requirements in which user representatives and development representatives work together with a facilitator to produce a joint requirements specification. JAD, which originated in Canada in the 1970s, is now very common in information systems development. Books, training, and consulting groups that offer JAD facilitation are also very common. Compared to the old adversarial style of requirements development, JAD can cut creeping requirements by almost half.

PROTOTYPES. Many changes occur after clients and users see an application's interface and output. So it is obvious that building early prototypes can help move some changes to the front of the development cycle. Prototypes can reduce requirements creep and can be combined with other approaches such as JAD. By themselves, prototypes can reduce requirements creep by somewhere between 10 and 25 percent.

RAPID APPLICATION DEVELOPMENT. For applications with fewer than about 1,000 function points, RAD schedules are somewhat shorter than conventional development. Finishing projects sooner obviously reduces the window of opportunity for changing requirements, so any way to shorten the schedule will also reduce requirements creep. There is insufficient data at present to know the exact impact of RAD on creeping requirements, but on the basis of preliminary observations, I estimate that RAD will reduce requirements changes by about 10 percent.

REQUIREMENTS INSPECTIONS. The classic formal inspection process can be applied to requirements as well as specifications and code. Requirements inspections are used more often for systems software than for information systems or commercial software. Inspections significantly reduce the rate of requirements creep because they find errors and inconsistencies. Inspections can reduce requirements creep by about 30 percent.

COST-PER-FUNCTION-POINT CONTRACTS. Several outsource vendors are exploring the use of cost-per-function-point contracts. This approach allows vendors to use a sliding scale, so the cost per function point rises for late requirements changes. This approach is too new and too experimental to judge the overall effectiveness against requirements creep, but the preliminary results are favorable. Because the US Internal Revenue Service is considering using the cost per function point to evaluate the

LOGGING OFF

This is the 27th and final column in the Software Challenges series. I thank the many readers who took the time to write or call me. Thanks also to the editing and production staff of *Computer*, who never failed to do an excellent job and meet their deadlines too. This is an example those of us in the software industry might hope to emulate.

Software continues to be a challenging occupation, but improvements do occur and may even be accelerating. The recent advances in software measurement and metrics capabilities derived from functional metrics are now making things visible that were difficult to see during the dark ages of lines of code. Topics such as requirements changes, quality, defect-removal rates, object-oriented programming, reuse, visual languages, and many others are now capable of solid empirical research.

Although this column is my last, I'm still writing as fast as possible. The second edition of *Applied Software Measurement* is nearing release, and I have two more books underway for late 1996 and early 1997 on software quality and software cost estimating. Juggling three books at once is as much as I can handle and still keep up with my regular work.

—*Capers Jones*

taxable value of software, cost per function point is starting to become a significant business metric.

QUALITY FUNCTION DEPLOYMENT. QFD is a way to analyze requirements in terms of user quality needs. QFD resembles JAD, but focuses on quality requirements rather than general features. There is insufficient data to judge the overall reduction in requirements creep, because QFD is just moving into the software world from the hardware world.

CHANGE-CONTROL BOARDS. A change-control board is a group of managers, client representatives, and technical personnel who decide which change to accept and which to reject. Change-control boards are often encountered in the military and systems-software domains, especially for systems in excess of 10,000 function points. These boards are comparatively rare for information systems, contract and outsource projects, and commercial software. In these domains project managers serve as de facto change-control boards. There is insufficient data to evaluate the effectiveness of formal change-control boards, but on the basis of my experience I estimate that change-control boards can reduce the volume of changes made during the initial development of large systems by about 25 percent.

CHANGE- AND CONFIGURATION-MANAGEMENT SYSTEMS. There are many commercial change-management systems on the market. These tools do not reduce the rate of change, but they greatly facilitate the speed with which changes can be processed. Hence they reduce the overall costs of change management. Modern change-management tools cover more than source code, facilitating changes to specifications, test libraries, code, and even user manuals.

These tools can also facilitate requirements traceability and help show the effect of change across multiple deliverables. Automating change control can reduce the effort of manually tracking changes by more than 70 percent and greatly reduce the probabilities of two serious software problems: *bad fixes* and *overlapping updates*. A bad fix is a repair that inadvertently injects a new error. An overlapping update is a mutually incompatible modification made by two or more developers to the same code.

Conclusion

Creeping user requirements have been troublesome since the software industry began, and they are a significant factor in at least half of the projects I and my colleagues have analyzed. There is no quick, perfectly effective cure. But now that you can measure the rate of creep, you can explore technologies to either reduce the rate or increase the pace of change.