Facets of R

Special invited paper on "The Future of R"

by John M. Chambers

We are seeing today a widespread, and welcome, tendency for non-computer-specialists among statisticians and others to write collections of R functions that organize and communicate their work. Along with the flood of software sometimes comes an attitude that one need only learn, or teach, a sort of basic how-to-write-the-function level of R programming, beyond which most of the detail is unimportant or can be absorbed without much discussion. As delusions go, this one is not very objectionable if it encourages participation. Nevertheless, a delusion it is. In fact, functions are only one of a variety of important facets that R has acquired by intent or circumstance during the three-plus decades of the history of the software and of its predecessor S. To create valuable and trustworthy software using R often requires an understanding of some of these facets and their interrelations. This paper identifies six facets, discussing where they came from, how they support or conflict with each other, and what implications they have for the future of programming with R.

Facets

Any software system that has endured and retained a reasonably happy user community will likely have some distinguishing characteristics that form its style. The characteristics of different systems are usually not totally unrelated, at least among systems serving roughly similar goals—in computing as in other fields, the set of truly distinct concepts is not that large. But in the mix of different characteristics and in the details of how they work together lies much of the flavor of a particular system.

Understanding such characteristics, including a bit of the historical background, can be helpful in making better use of the software. It can also guide thinking about directions for future improvements of the system itself.

The R software and the S software that preceded it reflect a rather large range of characteristics, resulting in software that might be termed rich or messy according to one's taste. Since R has become a very widely used environment for applications and research in data analysis, understanding something of these characteristics may be helpful to the community.

This paper considers six characteristics, which we will call *facets*. They characterize R as:

1. *an interface* to computational procedures of many kinds;
2. *interactive*, hands-on in real time;
3. *functional* in its model of programming;
4. *object-oriented*, "everything is an object";
5. *modular*, built from standardized pieces; and,
6. *collaborative*, a world-wide, open-source effort.

None of these facets is original to R, but while many systems emphasize one or two of them, R continues to reflect them all, resulting in a programming model that is indeed rich but sometimes messy.

The thousands of R packages available from CRAN, BioConductor, R-Forge, and other repositories, the uncounted other software contributions from groups and individuals, and the many citations in the scientific literature all testify to the useful computations created within the R model. Understanding how the various facets arose and how they can work together may help us improve and extend that software.

We introduce the facets more or less chronologically, in three pairs that entered into the software during successive intervals of roughly a decade. To provide some context, here is a brief chronology, with some references. The S project began in our statistics research group at Bell Labs in 1976, evolved into a generally licensed system through the 1980s and continues in the S+ software, currently owned by TIBCO Software Inc. The history of S is summarized in the Appendix to Chambers (2008); the standard books introducing still-relevant versions of S include Becker et al. (1988) (no longer in print), Chambers and Hastie (1992), and Chambers (1998). R was announced to the world in Ihaka and Gentleman (1996) and evolved fairly soon to be developed and managed by the R-core group and other contributors. Its history is harder to pin down, partly because the story is very much still happening and partly because, as a collaborative open-source project, individual responsibility is sometimes hard to attribute; in addition, not all the important new ideas have been described separately by their authors. The `http://www.r-project.org` site points to the on-line manuals, as well as a list of over 70 related books and other material. The manuals are invaluable as a technical resource, but short on background information. Articles in *R News*, the predecessor of this journal, give descriptions of some of the key contributions, such as namespaces (Tierney, 2003), and internationalization (Ripley, 2005).

An interactive interface

The goals for the first version of S in 1976 already combined two facets of computing usually kept apart, interactive computing and the development of procedural software for scientific applications.

One goal was a better interface to new computational procedures for scientific data analysis, usually implemented then as Fortran subroutines. Bell Labs statistics research had collected and written a variety of these while at the same time numerical libraries and published algorithm collections were establishing many of the essential computational techniques. These procedures were the essence of the actual data analysis; the initial goal was essentially a better interface to procedural computation. A graphic from the first plans for S (Chambers, 2008, Appendix, page 476) illustrated this.

But that interface was to be *interactive*, specifically via a language used by the data analyst communicating in real time with the software. At this time, a few systems had pointed the way for interactive scientific computing, but most of them were highly specialized with limited programming facilities; that is, a limited ability for the user to express novel computations. The most striking exception was APL, created by Kenneth Iverson (1962), an operator-based interactive language with heavy emphasis on general multiway arrays. APL introduced conventions that now seem obvious; for example, the result of evaluating a user's expression was either an assignment or a printed display of the computed object (rather than expecting users to type a print command). But APL at the time had no notion of an interface to procedures, which along with a limited, if exotic, syntax caused us to reject it, although S adopted a number of features from it, such as its approach to general multidimensional arrays.

From its first implementation, the S software combined these two facets: users carried out data analysis interactively by writing expressions in the S language, but much of the programming to extend the system was via new Fortran subroutines accompanied by interfaces to call them from S. The interfaces were written in a special language that was compiled into Fortran.

The result was already a mixed system that had much of the tension between human and machine efficiency that still stimulates debate among R users and programmers. At the same time, the flexibility from that same mixture helped the software to grow and adapt with a growing user community.

Later versions of the system replaced the interface language with individual functions providing interfaces to C or Fortran, but the combination of an interactive language and an interface to compiled procedures remains a key feature of R. Increasingly, interfaces to other languages and systems have been added as well, for example to database systems, spreadsheets and user interface software.

Functional and object-oriented programming

During the 1980s the topics of *functional programming* and *object-oriented programming* stimulated growing interest in the computer science literature. At the same time, I was exploring possibilities for the next S, perhaps "after S". These ideas were eventually blended with other work to produce the "new S", later called Version 3 of S, or S3, (Becker et al., 1988).

As before, user expressions were still interpreted, but were now parsed into objects representing the language elements, mainly function calls. The functions were also objects, the result of parsing expressions in the language that defined the function. As the motto expressed it, "Everything is an object". The procedural interface facet was not abandoned, however, but from the user's perspective it was now defined by functions that provided an interface to a C or Fortran subroutine.

The new programming model was functional in two senses. It largely followed the functional programming paradigm, which defined programming as the evaluation of function calls, with the value uniquely determined by the objects supplied as arguments and with no external side-effects. Not all functions followed this strictly, however.

The new version was functional also in the sense that nearly everything that happened in evaluation was a function call. Evaluation in R is even more functional in that language components such as assignment and loops are formally function calls internally, reflecting in part the influence of Lisp on the initial implementation of R.

Subsequently, the programing model was extended to include classes of objects and methods that specialized functions according to the classes of their arguments. That extension itself came in two stages. First, a thin layer of additional code implemented classes and methods by two simple mechanisms: objects could have an attribute defining their class as a character string or a vector of multiple strings; and an explicitly invoked method dispatch function would match the class of the function's first argument against methods, which were simply saved function objects with a class string appended to the object's name. In the second stage, the version of the system known as S4 provided formal class definitions, stored as a special metadata object, and similarly formal method definitions associated with *generic* functions, also defined via object classes (Chambers, 2008, Chapters 9 and 10, for the R version).

The functional and object-oriented facets were combined in S and R, but they appear more frequently in separate, incompatible languages. Typical object-oriented languages such as C++ or Java are not functional; instead, their programming model is of methods associated with a class rather than with a function and invoked on an object. Because the object is treated as a reference, the methods can usually modify the object, again quite a different model from that leading to R. The functional use of methods is more complicated, but richer and indeed necessary to support the essential role of functions. A few other languages do combine functional structure with methods, such as Dylan, (Shalit, 1996, chapters 5 and 6), and comparisons have proved useful for R, but came after the initial design was in place.

Modular design and collaborative support

Our third pair of facets arrives with R. The paper by Ihaka and Gentleman (1996) introduced a statistical software system, characterized later as "not unlike S", but distinguished by some ideas intended to improve on the earlier system. A facet of R related to some of those ideas and to important later developments is its approach to *modularity*. In a number of fields, including architecture and computer science, modules are units designed to be useful in themselves and to fit together easily to create a desired result, such as a building or a software system. R has two very important levels of modules: functions and packages.

Functions are an obvious modular unit, especially functions as objects. R extended the modularity of functions by providing them with an *environment*, usually the environment in which the function object was created. Objects assigned in this environment can be accessed by name in a call to the function, and even modified, by stepping outside the strict functional programming model. Functions sharing an environment can thus be used together as a unit. The programming technique resulting is usually called programming with closures in R; for a discussion and comparison with other techniques, see (Chambers, 2008, section 5.4). Because the evaluator searches for external objects in the function's environment, dependencies on external objects can be controlled through that environment. The namespace mechanism (Tierney, 2003), an important contribution to trustworthy programming with R, uses this technique to help ensure consistent behavior of functions in a package.

The larger module introduced by R is probably the most important for the system's growing use, the package. As it has evolved, and continues to evolve, an R package is a module combining in most cases R functions, their documentation, and possibly data objects and/or code in other languages.

While S always had the idea of collections of software for distribution, somewhat formalized in S4 as chapters, the R package greatly extends and improves the ability to share computing results as effective modules. The mechanism benefits from some essential tools provided in R to create, develop, test, and distribute packages. Among many benefits for users, these tools help programmers create software that can be installed and used on the three main operating systems for computing with data—Windows, Linux, and Mac OS X.

The sixth facet for our discussion is that R is a *collaborative* enterprise, which a large group of people share in and contribute to. Central to the collaborative facet is, of course, that R is a freely available, open-source system, both the core R software and most of the packages built upon it. As most readers of this journal will be aware, the combination of the collaborative efforts and the package mechanism have enormously increased the availability of techniques for data analysis, especially the results of new research in statistics and its applications. The package management tools, in fact, illustrate the essential role of collaboration. Another highly collaborative contribution has facilitated use of R internationally (Ripley, 2005), both by the use of locales in computations and by the contribution of translations for R documentation and messages.

While the collaborative facet of R is the least technical, it may eventually have the most profound implications. The growth of the collaborative community involved is unprecedented. For example, a plot in Fox (2008) shows that the number of packages in the CRAN repository exhibited literal exponential growth from 2001 to 2008. The current size of this and other repositories implies at a conservative estimate that several hundreds of contributors have mastered quite a bit of technical expertise and satisfied some considerable formal requirements to offer their efforts freely to the worldwide scientific community.

This facet does have some technical implications. One is that, being an open-source system, R is generally able to incorporate other open-source software and does indeed do so in many areas. In contrast, proprietary systems are more likely to build extensions internally, either to maintain competitive advantage or to avoid the free distribution requirements of some open-source licenses. Looking for quality open-source software to extend the system should continue to be a favored strategy for R development.

On the downside, a large collaborative enterprise with a general practice of making collective decisions has a natural tendency towards conservatism. Radical changes do threaten to break currently working features. The future benefits they might bring will often not be sufficiently persuasive. The very success of R, combined with its collaborative facet, poses a challenge to cultivate the "next big step" in software for data analysis.

Implications

A number of specific implications of the facets of R, and of their interaction, have been noted in previous sections. Further implications are suggested in considering the current state of R and its future possibilities.

Many reasons can be suggested for R's increasing popularity. Certainly part of the picture is a productive interaction among the interface facet, the modularity of packages, and the collaborative, open-source approach, all in an interactive mode. From the very beginning, S was not viewed as a set of procedures (the model for SAS, for example) but as an interactive system to support the implementation of new procedures. The growth in packages noted previously reflects the same view.

The emphasis on interfaces has never diminished and the range of possible procedures across the interface has expanded greatly. That R's growth has been especially strong in academic and other research environments is at least partly due to the ability to write interfaces to existing and new software of many forms. The evolution of the package as an effective module and of the repositories for contributed software from the community have resulted in many new interfaces.

The facets also have implications for the possible future of R. The hope is that R can continue to support the implementation and communication of important techniques for data analysis, contributed and used by a growing community. At the same time, those interested in new directions for statistical computing will hope that R or some future alternative engages a variety of challenges for computing with data. The collaborative community seems the only likely generator of the new features required but, as noted in the previous section, combining the new with maintenance of a popular current collaborative system will not be easy.

The encouragement for new packages generally and for interfaces to other software in particular are relevant for this question also. Changes that might be difficult if applied internally to the core R implementation can often be supplied, or at least approximated, by packages. Two examples of enhancements considered in discussions are an optional object model using mutable references and computations using multiple threads. In both cases, it can be argued that adding the facility to R could extend its applicability and performance. But a substantial programming effort and knowledge of the current implementation would be required to modify the internal object model or to make the core code thread-safe. Issues of back compatibility are likely to arise as well. Meanwhile, less ambitious approaches in the form of packages that approximate the desired behavior have been written. For the second example especially, these may interface to other software designed to provide this facility.

Opinions may reasonably differ on whether these "workarounds" are a good thing or not. One may feel that the total effort devoted to multiple approximate solutions would have been more productive if coordinated and applied to improving the core implementation. In practice, however, the nature of the collaborative effort supporting R makes it much easier to obtain a modest effort from one or a few individuals than to organize and execute a larger project. Perhaps the growth of R will encourage the community to push for such projects, with a new view of funding and organization. Meanwhile, we can hope that the growing communication facilities in the community will assess and improve on the existing efforts. Particularly helpful facilities in this context include the CRAN task views (`http://cran.r-project.org/web/views/`), the mailing lists and publications such as this journal.

Bibliography

R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language*. Chapman & Hall, London, 1988.

J. M. Chambers. *Programming with Data*. Springer, New York, 1998. URL `http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/`. ISBN 0-387-98503-4.

J. M. Chambers. *Software for Data Analysis: Programming with R*. Springer, New York, 2008. ISBN 978-0-387-75935-7.

J. M. Chambers and T. J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992. ISBN 9780412830402.

J. Fox. Editorial. *R News*, 8(2):1–2, October 2008. URL `http://CRAN.R-project.org/doc/Rnews/`.

R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

K. E. Iverson. *A Programming Language*. Wiley, 1962.

B. D. Ripley. Internationalization features of R 2.1.0. *R News*, 5(1):2–7, May 2005. URL `http://CRAN.R-project.org/doc/Rnews/`.

A. Shalit. *The Dylan Reference Manual*. Addison-Wesley Developers Press, Reading, Mass., 1996. ISBN 0-201-44211-6.

L. Tierney. Name space management for R. *R News*, 3(1):2–6, June 2003. URL `http://CRAN.R-project.org/doc/Rnews/`.

John M. Chambers
Department of Statistics
Stanford University
USA
`jmc@r-project.org`