Journal of Statistical Software
June 2011, Volume 42, Issue 2. http://www.jstatsoft.org/

Nineteen Ways of Looking at Statistical Software

Micah Altman
Harvard University

Simon Jackman
Stanford University

Abstract

We identify principles and practices for writing and publishing statistical software with maximum benefit to the scholarly community.

Keywords: statistical computation, programming methods.

1. Introduction

People who read journals like this are continually creating snippets of code. We can hardly avoid it. Anyone who performs statistical analysis on a regular basis naturally encounters repetitive tasks that beg for automation, data that needs to be prepared in new ways for analysis, and models that cannot be estimated well with canned statistical packages. So we write code – to automate tasks, manipulate data, extend existing methods of analyses, and to create new ones.

Most of this code is never seen by anyone else. Much of it evaporates soon after its task is completed. Without doubt, a good portion of code deserves this fate. However the rest is useful, and in general continues to persist for a time, while gradually ossifying or mutating until eventually, either lifeless or monstrous, it is buried in an unmarked grave. This is a lost opportunity – with a small additional effort this code could be shared, and lead long healthy lives in service to the community.

Exemplifying the potential value of statistical software, many of the packages included in this special volume, and the work they enabled, have already received substantial scholarly recognition: **wnominate** (Poole *et al.* 2011) is a modern port and update of Poole and Rosenthal's **NOMINATE** package, which won the *Statistical Software Award* from the Society for Political Methodology in 2009, has been used for hundreds of published articles. The authors of **Synth** (Abadie *et al.* 2011) received the *Gosnell Prize* from the Society for Political Methodlogy for the development and application of the methods that are implemented in that package. The authors of **MatchIt** (Ho *et al.* 2011) won the *Warren Miller Prize* for their article describing

the method implemented in that package. **BARD** (Altman and McDonald 2011) received the *Best Research Software* from the Information Technology and Politics section of the *American Political Science Association* in 2009, and is now being used to support public redistricting contests to promote transparency in government. And the other packages in this volume have supported many other research articles and other software development efforts.

We write this article to identify some principles for writing statistical code that benefits the community. This is based on our own experience writing statistical code professionally, our study of practices in the field of software engineering, and having directly observed many others creating software. These principles represent *good* practice, not necessarily *common* practice.

2. Six motivations for writing statistical software

1. *Understand your problem.* When you solve a problem by writing software you test both your knowledge of both the problem and the adequacy of your proposed method of solving it. As Knuth (1974) put it: "It has been often said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e., express it as an algorithm."

2. *Solve a problem.* Start from a real problem that you need to solve. It doesn't have to be a big problem, but it should be one for which a good solution does not already exist. Before you write code do some research: Check books, documentation, and software archives that could contain solutions to your problem. It is tempting to start writing immediately, or to dismiss existing solutions as inadequate, and many developers follow this impulse. (This is one of the reasons why the majority of software projects started with in SourceForge and other software archives are abandoned before reaching a stable a release.) But take a long look. Software development tends to involve hidden complexities that are revealed only after a substantial amount of design work has been done, and code has been written. And it is common to find *if* you've studied the code and documentation closely, and communicated with the code's maintainers, that existing open-source solutions can be adapted or extended. So, avoid building a solution from scratch if an adequate solution exists, which can be used or improved: When existing software fails to do what you want, or is too inaccurate, slow, tedious to use, or awkward to integrate into your larger research workflow, it is then time to build.

3. *Do good.* Code that solves your problem could often be useful to others. This is especially likely whenever you implement a statistical analysis that is not available in canned statistical packages. Think about the problem that your code solves – is it unique, or are there other problems like it that real people are actively trying to solve? Can your code solve these problems too? Can it be extended easily? Change your code if you can easily solve more real problems for real people by doing so.

4. *Get credit.* For many of us, credit is our most valued currency. Making your code useful to and available to others is an excellent way of making it possible for people to try a method that you have developed, and to aid in the replication and extension of work

with which you are involved. And work that is replicable is more likely to be used and cited more (Gleditsch *et al.* 2003).

For example, King et al.'s (2000) tremendously influential article, which eloquently advocated the use of simulation to improve the interpretation and presentation of statistical analyses, comprised simulation methods already well-known by methodologists. It had a striking impact in large part because the authors simultaneously developed, discussed, demonstrated, and distributed the **CLARIFY** package, which made it easy for other members of the discipline to apply these techniques; the software was later published separately as Tomz *et al.* (2003).

Citing software, particularly when using more complex models, is a best practice, and is essential to ensuring the replicability of research results (Altman *et al.* 2004). Encourage users to cite your software directly by supplying a clear citation for your work. For instance, when programming in R (R Development Core Team 2011), you should provide a `CITATION` file that clearly documents how your package should be cited.

5. *Make money.* Greed is good. Most of us, however, will not get rich selling statistical software. (And we are more interested in doing good research than in making money – which weighs against obscuring our analytic methods in a closed-source product.) But statistical programming can contribute to your economic well-being in other ways: it can allow you more easily to fulfill consulting requests which involve applying a method, market services based on new methods, teach commercial courses, enhance or constitute the goal of a grant proposal, and so forth.

6. *Do reproducible research.* Many details of research are embedded in the preparation of data and implementation of methods. When one publishes an article about empirical research, or an article about a new or complex method without a corresponding open implementation, one leaves much of the underlying scholarship hidden. Buckheit and Donoho (1995) — summarizing the insights in Schwab *et al.* (2000) — have formulated this principle pithily: "An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures." de Leeuw (1996) dubs this "Claerbout's principle" and notes that the same principle can be applied to all forms of teaching and publications that rely on computations.

   Tools such as `Sweave` (Leisch 2002), **odfWeave** (Kuhn 2006), **SASweave** (Lenth and Højsgaard 2007), **VIStrails** (Silva *et al.* 2007), **StatWeave** (Leisch 2002), and **The Dataverse Network System** (King 2007) are particularly useful for binding together data, software, and publications in support of reproducible research.

3. Eight ways to wake your code more useful

Zeileis (2006) succinctly summarizes many of the desirable features of implementation of an econometric procedure: numerically reliable, computational efficient, flexible, extensible, and reflecting the features of the conceptual model. To this we would add transparency, usability, and robustness as goals for your software development.

Applied in moderation, all of the following techniques will aid in obtaining these goals, and in the prevention of ossification, confusion, and uncontrolled mutation. These techniques are taken from recognized best practices in the field of software engineering, and some are still far from common in statistical code. These techniques will make your code easier to use, lend more confidence to the results, and enable others to add to it without disruption. And even if you never share it with anyone outside of your office – your code will still benefit from becoming more reliable, easier to understand, to maintain and to extend.

1. *Choose algorithms appropriately.* To make correct inferences requires the convergence of relevant statistical theory, informative data, well-chosen algorithms, and correct encoding of those algorithms using a programming language. The most sophisticated algorithm need not be used to solve the problem, as long as it is not horribly inefficient, and is demonstrably accurate enough to produce an answer to the required tolerances. However, avoid algorithms with known performance and numerical problems, algorithms with unknown accuracy, and software libraries using unknown algorithms.

   Most of the time, it is both convenient and less error-prone to use algorithms already implemented in well-tested software libraries. A good guide in general to choosing both algorithms and libraries is Skienna (2008); see Cormen *et al.* (2001) for a broad overview of general algorithms; Gentle (2005) and Galassi *et al.* (2009) on general algorithms for statistical programming (also see `http://maltman.hmdc.harvard.edu/numal/resources/` for a list of useful software libraries); Gentle (2007) for algorithms involving matrix algebra and statistics; and Nocedal and Wright (2000) for optimization algorithms used in statistical programming).

2. *Design programs for use and reuse.* A number of programming techniques are well-understood to lead to more durable, reusable code: Modular organization of files, clean separation of functionality into components and classes, interface encapsulation, naming & style conventions, and consistency applied to all levels of design and implementation. See Gamma *et al.* (1994), Booch *et al.* (2007) and McConnell (2004) for a detailed summary of methods relevant to (respectively) high, medium, and low-level software design.

   Furthermore, you will benefit by becoming familiar with the conventions and idioms that have been developed within the communities that will be using your work. A little time spent in preparation for writing code, skimming previous discussions on the developers' and users' mailing lists, and examining the code from other projects, will save a great deal of time later in debugging, rewriting, and documentation.

3. *Program defensively.* Users (and other programmers) are imperfect. They don't always read documentation, and when they do they don't always understand it. At some point, your program is going to be asked to do the impossible. When the impossible happens, your program should not explode, or worse, produce plausible nonsense, but should instead complain noticeably and informatively. In particular, you should adopt at least the following defensive programming techniques:

   Check all inputs values received by every external interface (this includes any public library procedure or object method you supply). Provide reasonable default values for inputs where possible. Document the valid range of inputs for your code, and check

these. Explicitly check that the output your program or function is producing is itself valid. (Check outputs even if you know this should be true by construction. Sometimes limits in theory, algorithm, or implementation causes the 'impossible' to occur.).

Report any problems the code encounters. For non-fatal problems, warn the user through the system's standard warning facility. For fatal problems, throw an exception (and document which exceptions your code will throw).

Avoid coding for a single system. Although outside of large-scale commercial development and large open-source projects with an extensive build and testing environment, it is probably not possible to make sure your program runs correctly on all other platforms and system configurations (such as the Estonian version of Windows XP Home Edition, service pack 3a) one should avoid inasmuch as possible the assumption that system on which your program runs is identical to your own. Avoid using system-dependent values, and make full use of system-independent interfaces.

For more detail on these techniques see McConnell (2004). A major finding of applied research in software engineering research is that these techniques lead to the production of code with drastically fewer errors. Not only will following these methods prevent problems from being falsely attributed to you and your code, they will protect you from many of your own errors.

4. *Write tests early.* You should provide tests that verify correct output based on known input. This makes it possible for others to use and extend the code with confidence. A number of modern software engineering techniques go so far as to advocate that complete sets of tests should be developed prior to any coding, then used as an indicator of the completeness of the implementations; e.g., Martin (2002).

   Tests can be designed for different purposes. Most tests of software falls into one of four categories. The first category tests "high-level" functionality of the program as it appears in the domain of use. Such "high-level" tests supply the software with known inputs and compare the results to output known (or defined) to be correct.[1] "Unit" tests are similar, except they are defined in terms of smaller operational units within the software, such as components, classes, methods, or functions. "Load" tests are designed to reveal the operation of software under inputs of increasing size in the domain of use (e.g., increasing numbers of variables, bytes, observations, number of connections). These three tests evaluate external behavior of the program, In addition, "lint" tests detect constructs in the source code that are generally associated with defects.

   Testing tools make it easier to find defects and to identify the source of aberrant behavior. As important they make it easier to write code in the first place. They do this by helping to more rigorously define and verify the behavior of the program. And by catching defects earlier in the development process.

   There are many tools available for automating tests. For example, in R writing can be as easy as specifying a set of code to run, and then providing a copy of the expected output, and putting these in the appropriate package directory. Valuable tools for testing and

[1] High level tests are also known as "benchmark", "acceptance", and "regression" tests when the known inputs correspond to canonical inputs, a set of inputs pre-defined by a requirements document, or the inputs that provoke a previously recognized bug.

debugging in R include **codetools** (Tierney 2011), **debug** (Bravington 2011), and **RUnit** (Burger *et al.* 2010).

5. *Measure accuracy ("trust but verify").* Unlike most software, which simply fails to produce results when broken or used improperly, statistical code almost always produces output with some gloss of plausibility. Thus, stable algorithms, conservative implementation, and full documentation are vital to producing trustworthy statistical software. Formal benchmarks, testing in extended precision environments, and sensitivity analyses are critical. For details on testing and programming for numerically reliable software see Altman *et al.* (2004) and Higham (2002). Users of R may find **accuracy** (Altman *et al.* 2007) and **GMP** (Granlund 2010) useful tools in this regard.

6. *Provide an open-source license.* To put it simply, give people *permission* to use your software. Code that you write is automatically copyrighted, and without a license others cannot have confidence that they can reuse the code, or even share the results that it produces. We recommend using the standard GPL license for most statistical code, since it since it allows others to use, reuse, and extend the code itself. For a guide to the other types of open source licenses see Rosen (2004). Using an open-source license maximizes the usefulness and influence of your work. And, after doing all of the work we describe above, you want people to use your code, don't you?

7. *Document your software, and how it changes.* Insufficiently documented software is indistinguishable from magic.[2] It also goes without saying (although we'll say it anyway) that if you expect others (or yourself, after you have moved on to future projects, and, inevitably, forgotten the details of the current one) to use your software, you should document how and why to use it. What is often overlooked, is documentation that systematically explains when *not* to use the software. Documentation should include known limitations of the output and implementation, such as degradation in accuracy outside a certain range of inputs. All algorithms and implementations are limited, and experienced (or hard-bitten) users are justly suspicious of programs that do not admit to their own limitations. See Altman *et al.* (2004) for dramatic examples of what can go wrong when these considerations are omitted.

   Using a version control system is a complement to external documentation – it allows you to document changes to your software, and to easily revisit earlier versions. Inevitably, at some point in your project you are going to discover that because of some change that was made to your software, something important that used to work no longer does. Version control provides a safety net, since you can use it to restore any previous version – particularly those that worked. Version control also provides a diagnostic tool, since you can use it to see exactly what was changed between the working and non-working versions. Finally, version control supports replication – since it ensures that any version of the code used to produced a published analysis remains available.

8. *Listen to users of your software.* Pay close attention to whether people use it, what they do with it, and where they encounter problems. Release incrementally in order to respond to and gather user feedback; see Martin (2002) for one effective strategy along these lines. Listen to other developers. Pay attention to suggestions your users and

[2] Magic is no longer generally viewed as sound basis for inference, although see Abramson (1997) for an opposing view.

make it easy for them to use the software and to contact you about it. Use the standard mechanisms for the community, and "go" to where users are already: e.g., rather than forcing them to use your registration mechanism, etc, you should subscribe to existing forums where users and potential users already go to ask questions.

4. Two places to share statistical software

While posting the code to your personal web site is the simplest way to share your code, and has the virtue of making preliminary work quickly available, there are better alternatives:

1. *Deposit your program in a high-quality software repository.* Programs distributed through ad-hoc (individual, or institutional) web sites are analogous to self-circulated drafts of manuscripts. High-quality software repositories, while not equivalent to peer review, provides a number of features that together establish a higher level of reliability and authority. Code repositories make it easier to find your software; frequently require some minimal degree of standardization in packaging and documentation; prevent the software from being withdrawn arbitrarily once deposited; require that the source code be supplied whenever a binary is distributed; and require that a software license be clearly indicated (and often that that license be in conformance with open-source standards). In addition, code archives generally provide a common infrastructure for versioning code and accessing previous versions, for reporting bugs, and for recording and making available the comments from other users on the software.

   Repositories providing the most extensive programming support, of which CRAN and CPAN are shining examples, also provide an infrastructure for installing building and testing your software on a variety of platforms: This eases installation, expands use of your software, increases portability, and can help to uncover software flaws.

2. *Publish your software in a peer-reviewed outlet.* In addition to the *Journal of Statistical Software*, journals such as *Computational Statistics & Data Analysis*, *Journal of Statistical Computation and Simulation*, *ACM Transactions on Mathematical Software* and more than a dozen others regularly publish statistical software or algorithms, and related articles. (See http://www.hmdc.harvard.edu/micah_altman/numal/resources/ for a regularly maintained list of publication outlets.)

5. Three ways software contributes to the study of politics

1. *Contributions to substantive understanding of political science.* Although methodology is a large and important component of political science, political science remains a substantively oriented field. The point of methodology is to contribute to our collective efforts to develop a science of politics. Software written by and for political scientists is an increasingly important component of that which we call "political methodology". And like other methodological innovations, the utility of our efforts with respect to software development is measured in terms of how it contributes to the scientific understanding

of politics. To this end, statistical software can be considered valuable in so far as it enables political scientists to better extract substantive content from data.

2. *Contributions to methodology.* Models are not fully useful until there exists a way for non-methodologists in the relevant discipline to apply them. Often, seeing how a theoretical statistical model can be applied to real data, and how that statistical model can be implemented, is neither obvious nor easy. Software is the vehicle through which new statistical models are made operational, or how existing models can be made to run better, faster, or to shed more light on substantive problems of interest.

3. *Contributions to teaching and learning.* Open software does more than enable application of methods, it also opens a window into the black box of computing. Students are given an opportunity to more deeply understand what is involved in analysis.

   Furthermore, open software provides another valuable way of communicating complex statistical and methodological concepts. Software packages are rapidly becoming an expected accompaniment for new books in these area. For example, **car** (Fox and Weisberg 2011), **BaM** (Gill 2010), and **AER** (Kleiber and Zeileis 2008) as well as being useful in their own right, were all created to accompany recent books. These, and more than a dozen more, are all available on CRAN.

Political methodology has grown from an almost unacknowledged niche to a robust field of political science. This is attested to by the existence of a top-ranked journal, *Political Analysis*; a high-quality annual conference; tremendous growth in the number of teaching and research positions devoted to it; and continued support from a large group of APSA members (Box-Steffensmeier and Sokhey 2007).

Statistical software seems poised to grow in a similar way. Commercial software has often fallen far short of keeping up with advances in political methodology (or with statistical methods in general). Commercial software vendors can be slow to add methods that while perhaps statistically more appropriate, are not in current demand, or that trade performance for robustness and accuracy (Stromberg 2004). In contrast, communities of methodologies, developing open code – sometimes in partnership with commercial companies such as Stata – have begun to fill the gap.

Writers of statistical software for political analysis contribute to a number of different communities. Other political scientists benefit from having sophisticated statistical models available for their own research. Students benefit from the opportunity to see examine *precisely* how results are generated. And the applied statistics community benefits from the development of new algorithms for statistical analysis. These can be large contributions, and a movement is well underway to recognize these contributions with the academic incentives of citation and peer review.

Acknowledgments

We thank Achim Zeileis and Thomas Yee for their helpful comments, and we apologize to Eliot Weinberger for parodying his title “19 Ways of Looking at Wang Wei” and to Arthur C. Clarke for paraphrasing his famous maxim in our recommendation on documentation.

References

Abadie A, Diamond A, Hainmueller J (2011). "**Synth**: An R Package for Synthetic Control Methods in Comparative Case Studies." *Journal of Statistical Software*, **42**(13), 1–17. URL http://www.jstatsoft.org/v42/i13/.

Abramson PR (1997). "Probing Well Beyond the Bounds of Conventional Wisdom." *American Journal of Political Science*, **41**, 675–682.

Altman M, Gill J, McDonald MP (2004). *Numerical Issues in Statistical Computing for the Social Scientist*. John Wiley & Sons, New York.

Altman M, Gill J, McDonald MP (2007). "**accuracy**: Tools for Accurate and Reliable Statistical Computing." *Journal of Statistical Software*, **21**(1), 1–30. URL http://www.jstatsoft.org/v21/i01/.

Altman M, McDonald MP (2011). "**BARD**: Better Automated Redistricting." *Journal of Statistical Software*, **42**(4), 1–28. URL http://www.jstatsoft.org/v42/i04/.

Booch G, Maksimchuk RA, Engel MW, Young BJ, Conallen J, Houston KA (2007). *Object-Oriented Analysis and Design with Applications*. 3rd edition. Addison-Wesley Professional, Boston.

Box-Steffensmeier JM, Sokhey AE (2007). "A Dynamic Labor Market: How Political Science is Opening up to Methodologists, and How Methodologists are Opening up Political Science." *PS: Political Science and Politics*, **15**, 125–129.

Bravington MV (2011). ***debug**: MVB's Debugger for R*. R package version 1.2.4, URL http://CRAN.R-project.org/package=debug.

Buckheit JB, Donoho DL (1995). "WaveLab and Reproducible Research." In A Antoniadis, G Oppenheim (eds.), *Wavelets in Statistics*, Lecture Notes in Statistics, pp. 55–82. Springer-Verlag, New York.

Burger M, Juenemann K, Koenig T (2010). ***RUnit**: R Unit Test Framework*. R package version 0.4.26, URL http://CRAN.R-project.org/package=RUnit.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001). *Introduction to Algorithms*. 2nd edition. MIT Press, Cambridge.

de Leeuw J (1996). "Reproducible Research: The Bottom Line." *Technical Report 301*, Department of Statistics, University of California, Los Angeles. URL http://repositories.cdlib.org/uclastat/papers/2001031101/.

Fox J, Weisberg S (2011). *An R Companion to Applied Regression*. 2nd edition. Sage, Thousand Oaks, CA.

Galassi M, Davies J, Theiler J, Gough B, Jungman G, Alken P, Booth M, Rossi F (2009). *Gnu Scientific Library Reference Manual*. 3rd edition. Network Theory Limited, Bristol. ISBN 978-0-9546120-7-8, URL http://www.gnu.org/software/gsl/.

Gamma E, Helm R, Johnson R, Vlissides JM (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Boston.

Gentle JE (2005). *Elements of Computational Statistics*. Springer-Verlag, New York.

Gentle JE (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer-Verlag, New York.

Gill J (2010). ***BaM**: Functions and Datasets for Books by Jeff Gill*. R package version 0.99, URL http://CRAN.R-project.org/package=BaM.

Gleditsch NP, Metelits C, Strand H (2003). "Posting Your Data: Will You Be Scooped or Will You Be Famous?" *International Studies Perspectives*, **4**, 89–97.

Granlund T (2010). ***GMP**: The GNU Multiple Precision Library, Version 5.0.1*. URL http://GMPlib.org/.

Higham NJ (2002). *Accuracy and Stability of Numerical Algorithms*. 2nd edition. SIAM Press, Philadelphia.

Ho DE, Imai K, King G, Stuart EA (2011). "**MatchIt**: Nonparametric Preprocessing for Parametric Causal Inference." *Journal of Statistical Software*, **42**(8), 1–28. URL http://www.jstatsoft.org/v42/i08/.

King G (2007). "An Introduction to the Dataverse Network as an Infrastructure for Data Sharing." *Sociological Methods and Research*, **32**(2), 173–199.

King G, Tomz M, Wittenberg J (2000). "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science*, **44**, 241–55.

Kleiber C, Zeileis A (2008). *Applied Econometrics with R*. Springer-Verlag, New York.

Knuth DE (1974). "Computer Science and Its Relation to Mathematics." *American Mathematical Monthly*, **81**, 323–343.

Kuhn M (2006). "**Sweave** and the Open Document Format – The **odfWeave** Package." *R News*, **6**(4), 2–8. URL http://CRAN.R-project.org/doc/Rnews/.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *COMPSTAT 2002 – Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg.

Lenth RV, Højsgaard S (2007). "**SASweave**: Literate Programming Using SAS." *Journal of Statistical Software*, **19**(8), 1–20. URL http://www.jstatsoft.org/v19/i08/.

Martin RC (2002). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, New Jersey.

McConnell S (2004). *Code Complete*. 2nd edition. Microsoft Press, Redmond.

Nocedal J, Wright S (2000). *Numerical Optimization*. Springer-Verlag, New York.

Poole K, Lewis J, Lo J, Carroll R (2011). "Scaling Roll Call Votes with **wnominate** in R." *Journal of Statistical Software*, **42**(14), 1–21. URL http://www.jstatsoft.org/v42/i14/.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rosen L (2004). *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, New Jersey.

Schwab M, Karrenbach N, Claerbout J (2000). "Making Scientific Computations Reproducible." *Computing in Science & Engineering*, **2**, 61–67. URL http://sepwww.stanford.edu/research/redoc/.

Silva C, Freire J, Callahan S (2007). "Provenance for Visualizations: Reproducibility and Beyond." *IEEE Computing in Science & Engineering*, **9**(1), 82–89.

Skienna SS (2008). *Algorithm Design Manual*. 2nd edition. Springer-Verlag, New York.

Stromberg A (2004). "Why Write Statistical Software? The Case of Robust Statistical Methods." *Journal of Statistical Software*, **10**(5), 1–8. URL http://www.jstatsoft.org/v10/i05.

Tierney L (2011). ***codetools**: Code Analysis Tools for R*. R package version 0.2-8, URL http://CRAN.R-project.org/package=codetools.

Tomz M, Wittenberg J, King G (2003). "**CLARIFY**: Software for Interpreting and Presenting Statistical Results." *Journal of Statistical Software*, **10**, 1–30. URL http://www.jstatsoft.org/v08/i01.

Zeileis A (2006). "Implementing a Class of Structural Change Tests: An Econometric Computing Approach." *Computational Statistics & Data Analysis*, **50**(11), 2987–3008.

Affiliation:

Micah Altman
Institute for Quantitative Social Science
Harvard University
1737 Cambridge Street, K 325
Cambridge, MA, 02138, United States of America
E-mail: micah_altman@harvard.edu
URL: http://www.hmdc.harvard.edu/micah_altman/

Simon Jackman
Department of Political Science
Encina Hall, Stanford University

Stanford, California 94305-6044, United States of America
E-mail: jackman@stanford.edu
URL: http://jackman.stanford.edu/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 42, Issue 2 *Submitted:* 2007-02-07
June 2011 *Accepted:* 2008-04-30