Bioconductor Project
Bioconductor Project Working Papers

Year 2004 *Paper* 1

Bioconductor: Open software development for computational biology and bioinformatics

Robert C. Gentleman, *Department of Biostatistical Sciences, Dana Farber Cancer Institute*
Vincent J. Carey, *Channing Laboratory, Brigham and Women's Hospital*
Douglas J. Bates, *Department of Statistics, University of Wisconsin, Madison*
Benjamin M. Bolstad, *Division of Biostatistics, University of California, Berkeley*
Marcel Dettling, *Seminar for Statistics, ETH, Zurich, CH*
Sandrine Dudoit, *Division of Biostatistics, University of California, Berkeley*
Byron Ellis, *Department of Statistics, Harvard University*
Laurent Gautier, *Center for Biological Sequence Analysis, Technical University of Denmark, DK*
Yongchao Ge, *Department of Biomathematical Sciences, Mount Sinai School of Medicine*
Jeff Gentry, *Department of Biostatistical Sciences, Dana Farber Cancer Institute*
Kurt Hornik, *Computational Statistics Group, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, AT*
Torsten Hothorn, *Institut fuer Medizininformatik, Biometrie und Epidemiologie, Friedrich-Alexander-Universitat Erlangen-Nurnberg, DE*
Wolfgang Huber, *Department for Molecular Genome Analysis (B050), German Cancer Research Center, Heidelberg, DE*
Stefano Iacus, *Department of Economics, University of Milan, IT*
Rafael Irizarry, *Department of Biostatistics, Johns Hopkins University*
Friedrich Leisch, *Institut für Statistik und Wahrscheinlichkeitstheorie, Technische Universität Wien, AT*
Cheng Li, *Department of Biostatistical Sciences, Dana Farber Cancer Institute*
Martin Maechler, *Seminar for Statistics, ETH, Zurich, CH*
Anthony J. Rossini, *Department of Medical Education and Biomedical Informatics, University of Washington*

Guenther Sawitzki, *Statistisches Labor, Institut fuer Angewandte Mathematik, DE*
Colin Smith, *Department of Molecular Biology, The Scripps Research Institute, San Diego*
Gordon K. Smyth, *Division of Genetics and Bioinformatics, The Walter and Eliza Hall Institute of Medical Research, Melbourne, AU*
Luke Tierney, *Department of Statistics and Actuarial Science, University of Iowa*
Yee Hwa Yang, *Center for Bioinformatics and Molecular Biostatistics, Univerisity of California, San Francisco*
Jianhua Zhang, *Department of Biostatistical Sciences, Dana Farber Cancer Institute*

Bioconductor: Open software development for computational biology and bioinformatics

Robert C. Gentleman, Vincent J. Carey, Douglas J. Bates, Benjamin M. Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Guenther Sawitzki, Colin Smith, Gordon K. Smyth, Luke Tierney, Yee Hwa Yang, and Jianhua Zhang

Abstract

The Bioconductor project is an initiative for the collaborative creation of extensible software for computational biology and bioinformatics. We detail some of the design decisions, software paradigms and operational strategies that have allowed a small number of researchers to provide a wide variety of innovative, extensible, software solutions in a relatively short time. The use of an object oriented programming paradigm, the adoption and development of a software package system, designing by contract, distributed development and collaboration with other projects are elements of this project's success. Individually, each of these concepts are useful and important but when combined they have provided a strong basis for rapid development and deployment of innovative and flexible research software for scientific computation. A primary objective of this initiative is achievement of total remote reproducibility of novel algorithmic research results.

Software

Bioconductor: Open software development for computational biology and bioinformatics

Robert Gentleman[1], Vincent J. Carey[2], Douglas J. Bates[3], Ben Bolstad[4], Marcel Dettling[5], Sandrine Dudoit[4], Byron Ellis[6], Laurent Gautier[7], Y. Ge[4], Jeff Gentry[1], Kurt Hornik[8], Torsten Hothorn[8], Wolfgang Huber[9], Stefano Iacus[10], Rafael Irizarry[11], Friedrich Leisch[8], Cheng Li[1], Martin Maechler[5], Anthony J. Rossini[12], Guenther Sawitzki[13], Colin Smith[14], Gordon Smyth[15], Luke Tierney[16], Jean Y. H. Yang[17], and Jianhua Zhang[1]

Addresses: [1]Department of Biostatistical Sciences, Dana Farber Cancer Institute, 44 Binney Street, Boston 02115, USA. [2]Channing Laboratory, Brigham and Women's Hospital, 75 Francis Street, Boston 02115, USA. [3]Department of Statistics, University of Wisconsin-Madison, 1210 W Dayton St., Madison, WI 53706, USA. [4]Division of Biostatistics, University of California, Berkeley, 140 Warren Hall, Berkeley, CA 94720-7360, USA. [5] Seminar for Statistics, ETH, Zürich, CH-8092, CH. [6]Department of Statistics, Harvard University, 1 Oxford St., Cambridge, MA 02138, USA. [7]Center for Biological Sequence Analysis, Technical University of Denmark, Building 208, Lyngby 2800, DK. [8]Institut für Statistik und Wahrscheinlichkeitstheorie, TU Wien, Wiedner Hauptstrasse 8-10/1071, Wien 1040, AT. [9]Department for Molecular Genome Analysis (B050), DKFZ (German Cancer Research Center), Im Neuenheimer Feld 580, 69120 Heidelberg, DE. [10]Department of Economics, University of Milan, 23 Via Mercalli, I-20123, Milan, Italy. [11]Department of Biostatistics, Johns Hopkins University, 615 N. Wolfe St. E3035, Baltimore, MD 21205, USA. [12]Department of Medical Education and Biomedical Informatics, University of Washington, Box 357240, 1959 NE Pacific, Seattle, WA 98195, USA. [13]Statistisches Labor, Institut fuer Angewandte Mathematik, Im Neuenheimer Feld 294, D 69120, Heidelberg, DE. [14]Department of Molecular Biology, The Scripps Research Institute, 10550 North Torrey Pines Road, TPC-28, La Jolla, CA 92037, USA. [15]Division of Genetics and Bioinformatics, The Walter and Eliza Hall Institute of Medical Research, 1G Royal Parade, Parkville, Vic 3052, Australia. [16]Department of Statistics and Actuarial Science, University of Iowa, 241 Schaeffer Hall, Iowa City, IA 52242, USA. [17]Center for Bioinformatics and Molecular Biostatistics, Univerisity of California, San Francisco, 500 Parnassus Ave, San Francisco 94143-0560, USA.

Correspondence: Robert Gentleman. E-mail: rgentlem@jimmy.harvard.edu

Abstract

The Bioconductor project is an initiative for the collaborative creation of extensible software for computational biology and bioinformatics. We detail some of the design decisions, software paradigms and operational strategies that have allowed a small number of researchers to provide a wide variety of innovative, extensible, software solutions in a relatively short time. The use of an object oriented programming paradigm, the adoption and development of a software package system, designing by contract, distributed development and collaboration with other projects are elements of this project's success. Individually, each of these concepts are useful and important but when combined they have provided a strong basis for rapid development and deployment of innovative and flexible research software for scientific computation. A primary objective of this initiative is achievement of total remote reproducibility of novel algorithmic research results.

Background

The Bioconductor project is an initiative for the collaborative creation of extensible software for computational biology and bioinformatics (CBB). Biology, molecular biology in particular, is undergoing two related transformations. First, there is a growing awareness of the computational nature of many biological processes and that computational and statistical models can be used to great benefit. Second, developments in high throughput data acquisition induce requirements for computational and statistical sophistication at each stage of the biological research pipeline. The main goal of the Bioconductor Project is creation of a durable and flexible software development and deployment environment that meets these new conceptual, computational, and inferential challenges. We strive to reduce barriers to entry to research in CBB. A key aim is simplification of the processes by which statistical researchers can explore and interact fruitfully with data resources and algorithms of CBB, and by which working biologists obtain access to and use state-of-the-art statistical methods for accurate inference in CBB.

Among the many challenges that arise for both statisticians and biologists are tasks of data acquisition, data management, data transformation, data modeling, combining different data sources, machine learning and developing new modeling strategies suitable to CBB. Fundamental to all of these tasks is the need for software; ideas alone cannot solve the substantial problems that arise. In this paper we consider problems particular to CBB as well as more general issues that arise when beginning a relatively large open source scientific software project. We hope that some of the strategies used by the Bioconductor project will be useful to other software initiatives.

The primary motivations for an open source computing environment for statistical genomics are transparency, pursuit of reproducibility, and efficiency of development.

- *Transparency.* High-throughput methodologies in CBB are extremely complex, and many steps are involved in the conversion of information from low-level information structures (e.g., microarray scan images) to statistical databases of expression measures coupled with design and phenotype data. It is not possible to say *a priori* how sensitive the ultimate analyses are to variations or errors in the many steps in the pipeline. Credible work in this domain requires exposure of the entire process.

- *Pursuit of reproducibility.* Experimental protocols in molecular biology are fully published lists of ingredients and algorithms for creating specific substances or processes. Accuracy of an experimental claim can be checked by complete obedience to the protocol. This standard should be adopted for algorithmic work in CBB. Portable source code should accompany each published analysis, coupled with the data on which the analysis is based.

- *Efficiency of development.* By development we refer not only to the development of the specific computing resource but to the development of computing methods in CBB as a whole. Software and data resources in an open source environment can be read by interested investigators, and can be modified and extended to achieve new functionalities. Novices can use the open sources as

learning materials. This is particularly effective when good documentation protocols are established. The open source approach thus aids in recruitment and training of future generations of scientists and software developers.

The rest of this paper is devoted to describing the computing science methodology underlying Bioconductor. The main sections to follow detail design methods and specific coding and deployment approaches, describe specific unmet challenges, and review limitations and future aims. Because a large number of software components and projects are referenced, we have included a glossary of technical terms as a final section. Terms defined in the glossary are marked at first mention with a superscript asterisk.

Methodology

The software development strategy we have adopted has several precedents. In the mid-1980s Richard Stallman started the Free Software Foundation and the GNU* project as an attempt to provide a free and open implementation of the Unix operating system. One of the major motivations for the project was the idea that for researchers in computational sciences *"their creations/discoveries (software) should be available for everyone to test, justify, replicate and work on to boost further scientific innovation"* [1]. Together with the Linux kernel the GNU/Linux combination sparked the huge open source movement we know today. Open source software is no longer viewed with prejudice, it has been adopted by major information technology companies and has changed the way we think about computational sciences. A large body of literature exists on how to manage open source software projects, see [2] for both a good introduction and a comprehensive bibliography.

One of the key success factors of the Linux kernel is its modular design, which allows for independent and parallel development of code [3] in a virtual decentralised network [1]. Developers are not managed within the hierarchy of a company, but are directly responsible for parts of the project and interact directly (where necessary) to build a complex system [4]. Our organization and development model has attempted to follow these principles, as well as those that have evolved from the R project.

In this section, we review seven topics important to establishment of a scientific open source software project and discuss them from a CBB point of view: language selection, infrastructure resources, design strategies and commitments, distributed development and recruitment of developers, reuse of exogenous resources, publication and licensure of code, and documentation.

Language selection

CBB poses a wide range of challenges and any software development project will need to consider which specific aspects it will address. For the Bioconductor project we wanted to focus initially on bioinformatics problems. In particular we were interested in data management and analysis problems associated with DNA microarrays. This orientation necessitated a programming environment that had good numerical capabil-

ities, flexible visualization capabilities, access to databases and a wide range of statistical and mathematical algorithms. Our collective experience with R [5] suggested that its range of well-implemented statistical and visualization tools would decrease development and distribution time for robust software for CBB. We also note that R is gaining widespread usage within the CBB community independently of the Bioconductor Project. Many other bioinformatics projects and researchers have found R to be a good language and toolset with which to work. Examples include the Spot system (see glossary), MAANOVA [6], and dChip [7]. We now briefly enumerate features of the R software environment that are important motivations behind its selection.

Prototyping capabilities. R is a high level interpreted language in which one can easily and quickly prototype new computational methods. These methods may not run quickly in the interpreted implementation, and those that are successful and that get widely used will often need to be reimplemented to run faster. This is often a good compromise; we can explore lots of concepts easily and put more effort into those that are successful.

Packaging protocol. The R environment includes a well-established system for packaging together related software components and documentation. There is a great deal of support in the language for creating, testing, and distributing software in the form of *packages*. Using a package system lets us develop different software modules and distribute them with clear notions of protocol compliance, test-based validation, version identification, and package interdependencies. The packaging system has been adopted by hundreds of developers around the world and lies at the heart of the Comprehensive R Archive Network, where several hundred independent but interoperable packages addressing a wide range of statistical analysis and visualization objectives may be downloaded as open source.

Object-oriented programming support. The complexity of problems in CBB is often translated into a need for many different software tools to attack a single problem. Thus, many software packages are used for a single analysis. To secure reliable package interoperability, we have adopted a formal object oriented programming discipline, as encoded in the 'S4' system of formal classes and methods [8]. The Bioconductor project was an early adopter of the S4 discipline and was the motivation for a number of improvements (established by John Chambers) in object oriented programming for R.

WWW connectivity. Access to data from on-line sources is an essential part of most CBB projects. R has a well-developed and tested set of functions and packages that provide access to different databases and to web resources (via `http`, for example). There is also a package for dealing with XML*, available from the Omegahat Project, and an early version of a package for a SOAP client* [9], *SSOAP*, also available from the Omegahat Project. These are much in line with proposals made in [10] and have aided our work towards creating an environment in which the user perceives tight integration of diverse data, annotation and analysis resources.

Statistical simulation and modeling support. Among the statistical and numerical algorithms provided by R are its random number generators and machine learning algorithms. These have been well tested and are known to be reliable. The Bioconductor Project has been able to adapt these to the requirements in CBB with minimal effort. It is also worth noting that a number of innovations and extensions based on work of

researchers involved in the Bioconductor project have been flowing back to the authors of these packages.

Visualization support. Among the strengths of R are its data and model visualization capabilities. Like many other areas of R these capabilities are still evolving. We have been able to quickly develop plots to render genes at their chromosomal locations, a *heatmap* function, along with many other graphical tools. There are clear needs to make many of these plots interactive so that users can query them and navigate through them and our future plans involve such developments.

Support for concurrent computation. R has also been the basis for pathbreaking research in parallel statistical computing. Packages such as *snow* and *rpvm* simplify the development of portable interpreted code for computing on a Beowulf or similar computational cluster of workstations. These tools provide simple interfaces which allow for high-level experimentation in parallel computation by computing on functions and environments in concurrent R sessions on possibly heterogeneous machines. The *snow* package provides a higher level of abstraction which is independent of the communication technology such as MPI* or PVM*. Parallel random number generation [11], essential when distributing parts of stochastic simulations across a cluster, is managed by *rsprng*. Practical benefits and problems involved with programming parallel processes in R are described more fully in both [12] and [13].

Community. Perhaps the most important aspect of using R is its active user and developer communities. This is not a static language. R is undergoing major changes that focus on the changing technological landscape of scientific computing. Exposing biologists to these innovations and simultaneously exposing those involved in statistical computing to the needs of the CBB community has been very fruitful and we hope beneficial to both communities.

Infrastructure base

We began with the perspective that significant investment in software infrastructure would be necessary at the early stages. The first two years of the Bioconductor project have included significant effort in developing infrastructure in the form of reusable data structures and software/documentation modules (R packages). The focus on reusable software components is in sharp contrast to the *one-off* approach that is often adopted. In a one-off solution to a bioinformatics problem, code is written to obtain the answer to a given question. The code is not designed to work for variations on that question or to be adaptable for application to distinct questions, and may indeed only work on the specific dataset to which it was originally applied. A researcher who wishes to perform a kindred analysis must typically construct the tools from scratch. In this situation, the scientific standard of reproducibility of research is not met except via laborious re-invention. It is our hope that reuse, refinement and extension will become the primary software-related activities in bioinformatics. When reusable components are distributed on a sound platform, it becomes feasible to demand that a published novel analysis be accompanied by portable and open software tools that perform all the relevant calculations. This will facilitate direct reproducibility, and will increase the efficiency of research by making transparent the means to vary or extend the new computational method.

Two examples of the software infrastructure concepts described here are the `exprSet` class of the *Biobase* package, and the various Bioconductor metadata packages, for example *hgu95av2*.

An `exprSet` is a data structure that binds together array-based expression measurements with phenotype and administrative data for a collection of microarrays. Based on R `data.frame` and `list` structures, `exprSets` offer much convenience to programmers and analysts for gene filtering, constructing annotation-based subsets, and for other manipulations of microarray results. The `exprSet` design facilitates a three-tier architecture for providing analysis tools for new microarray platforms: low-level data are bridged to high-level analysis manipulations via the `exprSet` structure. The designer of analysis procedures can focus on the `exprSet` data representation and ignore low-level structures and processes; likewise the low-level process designer need not cater for any particular analysis data structure expectation, but need only program to the `exprSet` representation. This design has been accepted by several distinct microarray processing initiatives (*affy*, by Irizarry, Gautier and Bolstad, *marrayClasses*, by Dudoit and Yang, *limma*, by Smyth, Ritchie, Wettenhall and Thorn, and *exprDB*, a high-performance microarray data store package based on Berkeley DB* by Ellis). Here, "acceptance" of the design entails that key products of the packages can be formally coerced to instances of the `exprSet` class.

The *hgu95av2* package is one of a large collection of related packages that relate manufactured chip components to biological metadata concerning sequence, gene functionality, gene membership in pathways, and physical and administrative information about genes. The package includes a number of conventionally named hashed environments providing high-performance retrieval of metadata based on probe nomenclature, or retrieval of groups of probe names based on metadata specifications. Both types of information (metadata and probe name sets) can be used very fruitfully with `exprSets`: for example, a vector of probe names immediately serves to extract the expression values for the named probes, because the `exprSet` structure inherits the named extraction capacity of R `data.frames`.

Design strategies and commitments

Well-designed scientific software should reduce data complexity, ease access to modeling tools and support integrated access to diverse data resources at a variety of levels. Software infrastructure can form a basis for both good scientific practice (others should be able to easily replicate experimental results) and for innovation.

The adoption of *designing by contract*, *object-oriented programming*, *modularization*, *multiscale executable documentation*, and *automated resource distribution* are some of the basic software engineering strategies employed by the Bioconductor Project.

Designing by contract. While we do not employ formal contracting methodologies (e.g., Eiffel*) in our coding disciplines, the contracting metaphor is still useful in characterizing the approach to the creation of interoperable components in Bioconductor. As an example, consider the problem of facilitating analysis of expression data stored in a relational database, with the constraints that a) one wants to be able to work with the

data as one would any `exprSet` and b) one does not want to copy unneeded records into R at any time. Technically data access could occur in various ways, using ODBC* connections, DCOM* communications, or CORBA*, to name but a few. In a designing by contract discipline, the provider of `exprSet` functionality must deliver a specified set of functionalities. Whatever object the provider's code returns, it must satisfy the `exprSet` contract. Among other things, this means the object must respond to the application of functions `exprs` and `pData` with objects that satisfy the R `matrix` and `data.frame` contracts respectively. It follows that `exprs(x)[i,j]`, for example, will return the number encoding the expression level for the `i`th gene for the `j`th sample in the object `x`, no matter what the underlying representation of `x`. Here `i` and `j` need not denote numerical indices but can hold any vectors suitable for interrogating matrices via the square-bracket operator. Satisfaction of the contract obligations simplifies specification of analysis procedures, which can be written without any concern for the underlying representations for `exprSet` information.

A basic theme in R development is simplifying the means by which developers can state, follow, and verify satisfaction of design contracts of this sort. Environment features that support convenient inheritance of behaviors between related classes with minimal recoding are at a premium in this discipline.

Object-oriented programming. There are various approaches to the object-oriented programming methodology. We have encouraged, but do not require, use of the so-called S4 system of formal classes and methods in Bioconductor software. The S4 object paradigm (defined primarily by Chambers in *Programming with Data*, with modifications embodied in R) is similar to that of Common Lisp [14] and Dylan [15]. In this system, classes are defined to have specified structures (in terms of a set of typed *slots*) and inheritance relationships, and methods are defined both generically (to specify the basic contract and behavior) and specifically (to cater for objects of particular classes). Constraints can be given for objects intended to instantiate a given class, and objects can be checked for validity of contract satisfaction. The S4 system is a basic tool in carrying out the designing by contract discipline, and has proven quite effective.

Modularization. The notion that software should be designed as a system of interacting modules is fairly well-established. Modularization can occur at various levels of system structure. We strive for modularization at the data structure, R function, and R package levels. This means that data structures are designed to possess minimally sufficient content to play a meaningful role in efficient programming. The `exprSet` structure, for example, contains information on expression levels (`exprs` slot), variability (`se.exprs`), phenotype data (`phenoData` slot), and several types of metadata (slots `description`, `annotation`, and `notes`). The tight binding of phenotype data with expression data spares developers the need to track these two types of information separately. The `exprSet` structure explicitly excludes information on gene-related annotation (such as gene symbol or chromosome location) because these are potentially volatile and are not needed in many activities involving `exprSets`. Modularization at the R function level entails that functions are written to do one meaningful task and no more, and that documents (`help` pages) are available at the function level with

worked examples. This simplifies debugging and testing. Modularization at the package level entails that all packages include sufficient functionality and documentation to be used and understood in isolation from most other packages. Exceptions are formally encoded in files distributed with the package.

Multiscale and executable documentation. Accurate and thorough documentation is fundamental to effective software development and use, and must be created and maintained in a uniform fashion to have the greatest impact. We inherit from R a powerful system for small-scale documentation and unit testing in the form of the executable example sections in function-oriented manual pages. We have also introduced a new concept of large-scale documentation with the vignette concept. Vignettes go beyond typical *man page* documentation which generally focuses on documenting the behavior of a function or small group of functions. The purpose of a vignette is to describe in detail the processing steps required to perform a specific task, which generally involves multiple functions and may involve multiple packages. Users of a package have interactive access to all vignettes associated with that package.

The Sweave system [16] was adopted for creating and processing vignettes. Once these have been written users can interact with them on different levels. The transformed documents are provided in Adobe's portable document format (PDF) and access to the code chunks from within R is available through various functions in the *tools* package. However, new users will need a simpler interface. Our first offering in this area is the vignette explorer `vExplorer` which provides a widget that can be used to navigate the various code chunks. Each chunk is associated with a button and the code is displayed in a window, within the widget. When the user clicks on the button the code is evaluated and the output presented in a second window. Other buttons provide other functionality, such as access to the PDF version of the document. We plan to extend this tool greatly in the coming years and to integrate it closely with research into reproducible research.

Automated software distribution. The modularity commitment imposes a cost on users who are accustomed to integrated 'end to end' environments. Users of Bioconductor need to be familiar with the existence and functionality of a large number of packages. To diminish this cost, we have extended the packaging infrastructure of R/CRAN to better support the deployment and management of packages at the user level. Automatic updating of packages when new versions are available and tools that obtain all package dependencies automatically are among the features provided as part of the *reposTools* package in Bioconductor. Note that new methods in R package design and distribution include the provision of MD5 checksums with all packages, to help with verification that package contents have not been altered in transit.

In conclusion, these engineering commitments and developments have led to a reasonably harmonious set of tools for CBB. It is worth considering how the S language notion that "everything is an object" impacts our approach. We have made use of this notion in our commitment to contracting and object-oriented programming, and in the automated distribution of resources, in which package catalogs and biological metadata are all straightforward R objects. Packages and documents are not yet treatable

as R objects, and this leads to complications. We are actively studying methods for simplifying authoring and use of documentation in a multipackage environment with namespaces that allow symbol reuse, and for strengthening the connection between session image and package inventory in use, so that saved R images can be restored exactly to their functional state at session close.

Distributed development and recruitment of developers

Distributed development is the process by which individuals who are significantly geographically separated produce and extend a software project. This approach has been used by the R project for approximately 10 years. This was necessitated in this case by the fact no institution currently has sufficient numbers of researchers in this area to support a project of this magnitude. Distributed development facilitates the inclusion of a variety of viewpoints and experiences. Contributions from individuals outside the project led to the expansion of the core developer group. Membership in the core depends upon the willingness of the developer to adopt shared objectives and methods and to submerge personal objectives in preference to creation of software for the greater scientific community.

Distributed development requires the use of tools and strategies that allow different programmers to work approximately simultaneously on the same components of the project. Among the more important requirements is for a shared code base (or archive) that all members of the project can access and modify together with some form of version management system. We adopted the Concurrent Versions System (CVS*, [17]) and created a central archive, within this system, that all members of the team have access to.

Additional discipline is needed to ensure that changes by one programmer should not result in a failure of other code in the system. Within the R language software components are naturally broken into *packages*, with a formal protocol for package structure and content specified in the R Extensions manual [18]. Each package should represent a single coherent theme. By using well defined applications programming interfaces (APIs) developers of a package are free to modify their internal structures as long as they continue to provide the documented outputs.

We rely on the testing mechanisms supported by the R package testing system [18] to ensure coherent, non-regressive development. Each developer is responsible for documenting all functions she writes and for providing examples and possibly other scripts or sets of commands that test her code. Each developer is responsible for ensuring that all tests run successfully before committing his or her changes back to the central archive. Thus, the person who knows the code best writes the test programs, but all are responsible for running them and ensuring that changes they have made do not affect the code of others. In some cases changes by one author will necessitate change in the code and tests of others. Under the system we are using these situations are detected and dealt with when they occur in development, reducing the frequency with which error reports come from the field.

Members of the development team communicate via a private mailing list. In many cases they also use private email, telephone and meetings at conferences in order to engage in joint projects and to keep informed of the ideas of other members.

Reuse of exogenous resources

We now present three arguments in favor of using and adapting software from other projects rather than reimplementing or reinventing functionality. The first argument that we consider is that writing good software is a challenging problem and any reimplementation of existing algorithms should be avoided if possible. Standard tools and paradigms that have been proven and are well understood should be preferred over new untested approaches. All software contains bugs but well used and maintained software tends to contain fewer.

The second argument is that CBB is an enormous field and that progress will require the coordinated efforts of many projects and software developers. Thus, we will require structured paradigms for accessing data and algorithms written in other languages and systems. The more structured and integrated this functionality is the easier it will be to use and hence the more it will be used. As specific examples we consider our recent development of tools for working with graph or network structures. There are three main packages in Bioconductor of interacting with graphs. They are *graph*, *RBGL* and *Rgraphviz*. The first of these provides the class descriptions and basic infrastructure for dealing with graphs in R, the second provides access to algorithms on graphs, and the third to a rich collection of graph layout algorithms. The *graph* package was written from scratch for this project, but the other two are interfaces to rich libraries of software routines that have been created by other software projects, BOOST* [19] and GraphViz* respectively, both of which are very substantial projects with large code bases. We have no interest in replicating that work and will, wherever possible, simply access the functions and libraries produced by other projects.

There are many benefits from this approach for us and for the other projects. For bioinformatics and computational biology we gain rapid access to a variety of graph algorithms including graph layout and traversal. The developers in those communities gain a new user base and a new set of problems that they can consider. Gaining a new user base is often very useful as new users with previously unanticipated needs tend to expose weaknesses in design and implementation that more sophisticated or experienced users are often able to avoid.

In a similar vein, we plan to develop and encourage collaboration with other projects, including those organized through the Open Bioinformatics Foundation and the International Interoperability Consortium. We have not specifically concentrated on collaboration to this point in part because we have chosen areas for development that do not overlap significantly with the tools provided by those projects. In this case our philosophy remains one of developing interfaces to the software provided by those projects and not reimplementing their work. In some cases, other projects have recognized the potential gains for collaboration and have started developing interfaces for us to their systems, with the intent of making future contributions [20].

Another argument in favor of standardization and reuse of existing tools is best made with reference to a specific example. Consider the topic of markup and markup languages. For any specific problem one could quickly devise a markup that is sufficient for that problem. So why then should we adopt a standard such as XML? Among the reasons for this choice is the availability of programmers conversant with the paradigm, and hence lower training costs. A second reason is that the XML commu-

nity is growing and developing and we will get substantial technological improvements without having to initiate them. This is not unusual. Other areas of computational research are as vibrant as CBB and by coordinating and sharing ideas and innovations we simplify our own tasks while providing stimulus to these other areas.

Publication and licensure of code

Modern standards of scientific publication involve peer review and subsequent publication in a journal. Software publication is a slightly different process with limited involvement to date of formal peer review or official journal publication. We release software under an open source license as our main method of publication. We did this in the hopes that it would encourage reproducibility, extension and in general adherence to the scientific method. This decision also ensures that the code is open to public scrutiny and comment.

There are many other reasons for deciding to release software under an open source license. Some of these are:

- to provide full access to algorithms and their implementation;
- to provide to users the ability to fix bugs without waiting for the developer, and to extend and improve the supplied software;
- to encourage good scientific computing and statistical practice by exhibiting fully appropriate tools and instruction;
- to provide a workbench of tools that allow researchers to explore and expand the methods used to analyze biological data;
- to ensure that the international scientific community is the owner of the software tools needed to carry out research;
- to lead and encourage commercial support and development of those tools that are successful;
- to promote reproducible research by providing open and accessible tools with which to carry out that research.

Another consideration that arose when determining the form of publication was the need to allow an evolutionary aspect to our own software. There are many reasons for adopting a strategy that would permit us to extend and improve our software offerings over time. The field of CBB is relatively volatile and as new technologies are developed new software and inferential methods are needed. Further, software technology itself is evolving. Thus, we wanted to have a publication strategy that could accommodate changes in software at a variety of levels. We hope that that strategy will also encourage our users to think of software technology as a dynamic field rather than a static one and to therefore be on the lookout for innovations in this arena as well as in more traditional biological ones.

Our decision to release software in the form of R packages is an important part of this consideration. Packages are easy to distribute, they have version numbers and

define an API. A coordinated *release* of all Bioconductor packages occurs twice every year. At any given time there is a *release* version of every package and a *development* version. The only changes allowed to be made on the release version are bug fixes and documentation improvements. This ensures that users will not encounter radical new behaviors in code obtained in the release version. All other changes such as enhancements or design changes are carried out on the development branch [see, e.g., 21].

Approximately six weeks before a release a major effort is taken to ensure that all packages on the development branch are coordinated and work well together. During that period extensive testing is carried out through peer review amongst the Bioconductor core. At release time all packages on the development branch that are included in the release change modes and are now released packages. Previous versions of these packages are deprecated in favor of the newly released versions. Simultaneously, a new development branch is made and the developers start to work on packages in the new branch. Note that these version-related administrative operations occur with little impact on developers. The release manager is responsible for package snapshot and file version modifications. The developers' source code base is fairly simple, and need not involve retention of multiple copies of any source code files, even though two versions are active at all times.

We would also like to point out that there are compelling arguments that can be made in favor of choosing different paradigms for software development and deployment. We are not attempting at this juncture to convince others to distribute software in this way, but rather elucidating our views and the reasons that we made our choice. Under a different set of conditions, or with different goals, it is entirely likely that we would have chosen a different model.

Challenges

We now consider four specific challenges that are raised by research in computational biology and bioinformatics: reproducibility, data evolution and complexity, training users, and responding to user needs.

Reproducible Research We would like to address the reproducibility of published work in CBB. Reproducibility is important in its own right, and is the standard for scientific discovery. Reproducibility is an important step in the process of incremental improvement or refinement. In most areas of science researchers continually improve and extend the results of others but for scientific computation this is generally the exception rather than the rule.

[22], referring to the work and philosophy of Claerbout, state the following principle:

> 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 that complete set of instructions that generated the figures.

There are substantial benefits that will come from enabling authors to publish not just an advertisement of their work but rather the work itself. A paradigm that fundamentally shifts publication of computational science from an advertisement of scholarship to the scholarship itself will be a welcome addition. Some of the concepts and tools that can be used in this regard are contained in [23] and [24].

When attempting to reimplement computational methodology from a published description many difficulties are encountered. In [25] the following points are made:

> Indeed the problem occurs wherever traditional methods of scientific publication are used to describe computational research. In a traditional article the author merely outlines the relevant computations: the limitations of a paper medium prohibit complete documentation including experimental data, parameter values and the author's programs. Consequently, the reader has painfully to reimplement the author's work before verifying and utilizing it.... The reader must spend valuable time merely rediscovering minutiae, which the author was unable to communicate conveniently.

The development of a system capable of supporting the convenient creation and distribution of reproducible research in CBB is a massive undertaking. Nevertheless, the Bioconductor Project has adopted practices and standards that assist in partial achievement of reproducible CBB.

Publication of the data from which articles are derived is becoming the norm in CBB. This practice provides one of the components needed for reproducible research – access to the data. The other major component that is needed is access to the software and the explicit set of instructions or commands that were used to transform the data to provide the outputs on which the conclusions of the paper rest. In this regard publishing in CBB has been less successful. It is easy to identify major publications in the most prestigious journals that provide sketchy or indecipherable characterizations of computational and inferential processes underlying basic conclusions. This problem could be eliminated if the data housed in public archives were accompanied by portable code and scripts that regenerate the article's figures and tables.

The combination of R's well-established platform independence with Bioconductor's packaging and documentation standards leads to a system in which distribution of data with working code and scripts can achieve most of the requirements of reproducible and replayable research in CBB. The steps leading to the creation of a table or figure can be clearly exposed in an Sweave document. An R user can export the code for modification or replay with variations on parameter settings, to check robustness of the reported calculations or to explore alternative analysis concepts.

Thus we believe that R and Bioconductor can provide a start along the path towards generally reproducible research in CBB. The infrastructure in R that is used to support replayability and remote robustness analysis could be implemented in other languages such as Perl* and Python*.All that is needed is some platform-independent format for binding together the data, software, scripts defining the analysis, and a document that can be rendered automatically to a conveniently readable account of the analysis steps and their outcomes. If the format is an R package, this package then constitutes a single distributable software element that embodies the computational science being published. This is precisely the *compendium concept* espoused in [23].

Data Evolution Meta-data are data about data and their definition depends on the perspective of the investigator. Meta-data for one investigator may well be experimental data for another. There are two major challenges that we will consider. First is the evolutionary nature of the meta-data. As new experiments are done and as our understanding of the biological processes involved increases the meta-data changes and evolves. The second major problem that concerns meta-data data is its complexity. We are trying to develop software tools that make it easier for data analysts and researchers to use the existing meta-data appropriately.

The constant changing and updating of the meta-data suggests that we must have a system or a collection process that ensures that any meta-data can be updated and the updates can be distributed. Users of our system will want access to the most recent versions. Our solution has been to place meta-data into R packages. These packages are built using a semi-automatic process [26] and are distributed (and updated) using the package distribution tools developed in the *reposTools* package. There is a natural way to apply version numbers so users can determine if their data are up to date or if necessary they can obtain older versions to verify particular analyses. Further, users can synchronize a variety of meta-data packages according to a common version of the data sources that they were constructed from.

There are a number of advantages that come from automating the process of building data packages. First, the modules are uniform to an extent that would not be possible if the packages were human written. This means that users of this technology need only become acquainted with one package to be acquainted with all such packages. Second, we can create many packages very quickly. Hence the labor savings are substantial. For microarray analyses all data packages should have the same information (chromosomal location, gene ontology categories, etc). The only difference between the packages is that each references only the specific set of genes (probes) that were assayed. This means that data analysts can easily switch from one type of chip to another. It also means that we can develop a single set of tools for manipulating the meta-data and improvements in those tools are available to all users immediately. Users are free to extend data packages with data from other, potentially proprietary, sources.

Treating the data in the same manner that we treat software has also had many advantages. On the server side we can use the same software distribution tools, indicating updates and improvements with version numbering. On the client side, the user does not need to learn about the storage or internal details of the data packages. They simply install them like other packages and then use them.

One issue that often arises is whether one should simply rely on on-line sources for meta-data. That is, given an identifier the user can potentially obtain more up to date information by querying the appropriate data bases. The data packages we are proposing cannot be as current. There are, however, some disadvantages to the approach of accessing all resources on-line. First, users are not always on-line, they are not always aware of all applicable information sources and the investment in person-time to obtain such information can be high. There are also issues of reproducibility that are intractable since the owners of the web resources are free to update and modify their offerings at will. Some, but not all, of these difficulties can be alleviated if the data are available in a *web services* format.

Another argument that can be made in favor of our approach, in this context, is that

it allows the person constructing the data packages to amalgamate disparate information from a number of sources. In building meta-data packages for Bioconductor, we find that some data are available from different sources and under those circumstances we look for consensus, if possible. The process is quite sophisticated and is detailed in the *AnnBuilder* package and paper [26].

Training Most of the projects in CBB require a combination of skills from biology, computer science, and statistics. Since the field is new and there has been little specialized training in this area it seems that there is some substantial benefit to be had from paying attention to training. From the perspective of the Bioconductor project many of our potential users are unfamiliar with the R language and generally are scientifically more aligned with one discipline than all three. It is therefore important that we produce documentation for the software modules that is accessible to all. We have taken a two-pronged approach to this, we have developed substantial amounts of course material aimed at all the constituent disciplines and we have developed a system for interactive use of software and documentation in the form of *vignettes* and more generally in the form of navigable documents with dynamic content.

Course materials have been developed and refined over the past two to three years. Several members of the Bioconductor development team have taught courses and subsequently refined the material, based on success and feedback. The materials developed are modular and are freely distributed, although restrictions on publication are made. The focus of the materials is the introduction and use of software developed as part of the Bioconductor project but that is not a requirement and merely reflects our own specific purposes and goals.

In this area we feel that we would benefit greatly from contributions from those with more experience in technical document authoring. There are likely to be strategies, concepts and methodologies that are standard practice in that domain that we are largely unaware of. However, in the short-term, we rely on the students, our colleagues and the users of the Bioconductor system to both guide us and we hope that many will contribute. Others can easily make substantial contributions, even those with little or no programming skills. What is required is domain knowledge in one field of interest and the recognition of a problem that requires additional domain knowledge from another of the fields of interest.

Our experience has been that many of these new users often transform themselves into developers. Thus, our development of training materials and documentation needs to pay some attention to the needs of this group as well. There are many more software components than we can collectively produce. Attracting others to collaboratively write software is essential to success.

Responding to user needs The success of any software project rests on its ability to both provide solutions to the problems it is addressing and to attract a user community. Attracting a user community itself requires a method of distributing the software and providing sufficient training materials to allow potential users to explore the system and determine whether it is sufficient for their purposes. An alternate approach would be to develop a graphical user interface (GUI) that made interactions with the system

sufficiently self-explanatory that documentation was not needed. We note that this solution is generally more applicable to cases where the underlying software tasks are well defined and well known. In the present case, the software requirements (as well as the statistical and biological requirements) are constantly evolving. R is primarily command-line oriented and we have chosen to follow that paradigm at least for the first few years of development. We would of course welcome and collaborate with those whose goal was in GUI development but our own forays into this area are limited to the production of a handful of widgets that promote user interaction at specific points.

Users have experienced difficulties downloading and installing both R and the Bioconductor modules. Some of these difficulties have been caused by the users local environments (firewalls and a lack of direct access to the internet), by problems with our software (bugs) which arise in part because it is in general very difficult to adequately test software that interacts over the internet. We have, however, managed to help every user that was willing to persist get both R and Bioconductor properly installed. Another substantial difficulty that we had to overcome was to develop a system that allowed users to download not just the software package that they knew they wanted, but additionally and at the same time, all other software packages that it relies on. With Bioconductor software there is a much larger inter-reliance on software packages (including those that provide machine learning, biological meta-data and experimental data) than for most other uses of R and the R package system. The package, *reposTools* contains much of the necessary infrastructure for handling these tasks. It is a set of functions for dealing with R package repositories which are basically internet locations for collections of R packages.

Once the basic software is installed users will need access to documentation such as the training materials described above and other materials such as the vignettes, described in a previous section. Such materials are most valuable if the user can easily obtain and *run* the examples on their own computer. We note the obvious similarity with this problem and that described in the section on reproducible research. Again, we are in the enjoyable situation of having a paradigm and tools that can serve two purposes.

Discussion

We have detailed the approach to software development taken by the Bioconductor Project. Bioconductor has been operational for about two years now and in that time it has become a prominent software project for CBB. We argue that the success of the project is due to many factors. These include the choice of R as the main development language, the adoption of standard practices of software design and a belief that the creation of software infrastructure is an important and essential component of a successful project of this size.

The group dynamics have also been an important factor in the success of Bioconductor. A willingness to work together, to see that cooperation and coordination in software development yields substantial benefits for the developers and the users and encouraging others to join and contribute to the project are also major factors in our success.

To date the project provides the following resources:

- An online repository for obtaining software, data and meta-data, papers, and training materials.
- A development team that coordinates the discussion of software strategies and development.
- A user community that provides software testing, suggested improvements and self-help.
- More than 50 software packages, hundreds of meta-data packages and a number of experimental data packages.

At this point it is worth considering the future. While many of the packages we have developed have been aimed at particular problems there have been others that were designed to support future developments. And that future seems very interesting. Many of the new problems we are encountering in CBB are not easily addressed by technology transfer, but rather require new statistical methods and software tools. We hope that we can encourage more statisticians to become involved in this area of research and to orient themselves and their research to the mixture of methodology and software development that is necessary in this field.

In conclusion we would like to note that the Bioconductor Project has many developers, not all of whom are authors of this paper, and all have their own objectives and goals. The views presented here are not intended to be comprehensive nor prescriptive but rather to present our collective experiences and the author's shared goals. In a very simplified version these can be summarized in the view that coordinated cooperative software development is the appropriate mechanism for fostering good research in CBB.

List of abbreviations

CBB Computational biology and bioinformatics

Glossary

The following provides a description and some references for some of the software and standards referenced in the paper.

Berkeley DB Berkeley DB is made by Sleepycat Software, www.sleepycat.com and is widely-used application-specific data management software.

BOOST The Boost web site provides free peer-reviewed portable C++ source libraries, www.boost.org.

CORBA The Common Object Request Broker Architecture (CORBA) is a vendor-independent architecture and infrastructure that allows computer applications to

locate and provide different services over a network. More information is available from the Object Management Group www.omg.org.

CVS The concurrent version control system, http://www.cvshome.org, is an open source, widely used, software version control system.

DCOM The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. More information is available at http://www.microsoft.com/com/tech/dcom.asp.

Eiffel Eiffel is an object oriented programming language that emphasizes the production of robust software. See http://smarteiffel.loria.fr/ for more information.

GNU stands for "GNU's Not Unix" and is the Free Software Foundation's project to provide a free alternative of the Unix operating system. The project provides a huge set of programs for Unix systems, among them a complete development environment including the GNU C compiler, www.gnu.org.

GraphViz The Graph Visualization Project, GraphViz, provides a collection of tools for manipulating graph structures and generating graph layouts, www.graphviz.org.

MPI Message-Passing Interface is a standardized API for communication between processes located on different machines in a computational cluster. It is implemented by a number of academic groups (usually provided as open source software) and commercial vendors. See http://www.mpi-forum.org/ for details.

ODBC Open DataBase Connectivity is a database access standard developed by Microsoft. ODBC enables data between applications and databases. The standard is open and nonproprietary. Information and documentation is available from a variety of sources including the MDAC SDK from Microsoft.

Omegahat The Omegahat Project, www.omegahat.org is a joint project with the goal of providing a variety of open-source software for statistical applications.

PDF Portable Document Format is a standard for creating and distributing electronic documents. The format is owned/controlled by Adobe (url).

Perl Practical Extraction and Report Language (Perl) is a general high-level programming language that excels at text manipulation. See http://www.perl.org and the Comprehensive Perl Archive Network (CPAN) for available add-ons to the system.

PVM Parallel Virtual Machine is a message passing library for communication of data between processes in a computational cluster, for the development and ease of deployment of high-performance computing applications. See http://www.csm.ornl.gov/pvm/pvm_home.html

Python Python is an interpreted, object-oriented high-level programming language that can be compiled on some systems. See http://www.python.org.

R is an Open Source implementation of the S language, winner of the 1998 ACM Software Systems Award. R is similar to the commercial S implementation S-Plus (www.insightful.com). S is both a general programming language and an extensible interactive environment for data analysis and graphics. See http://www.r-project.org for information on the project and CRAN (the Comprehensive R Archive Network) http://cran.r-project.org for available software and packages.

SOAP is an acronym for the Simple Object Access Protocol which is an XML dialect for representing distributed or remote method calls between applications. It has become a very popular protocol for implementing Web services, using HTTP as the communication mechanism and XML as the data representation. See http://www.w3.org/TR/SOAP/ for more information.

Spot (http://spot.cmis.csiro.au) is an R-based system for microarray image analysis.

XML stands for the eXtensible Markup Language, a text-based markup mechanism for representing self-describing data. Its syntax is the same as the familiar HTML (the Hyper Text Markup Language). However, one can define new and arbitrary tags in XML to define new, specialized dialects for representing arbitrary data in a self-describing manner. XML documents are made up of nodes which are arranged hierarchically. A class of XML documents (i.e. a dialect) can be described symbolically via a Document Type Definition (DTD) which describes the possible relationships between different types of nodes, i.e. which nodes can be nested within other node types and in what order. This allows one to also validate XML documents according to this specification without actually interpreting the specific content. Schema are a newer way to provide information not just about the structure of the document, but also about the data types within XML nodes.

The W3 organization (http://www.w3.org) provides much of the standardization and specification of XML and its dialects. The Cover Pages Web site (http://www.coverpages.org provides information on using XML in a wide variety of different applications.

XSL (the eXtensible Stylesheet Language) is a specific XML dialect that is used to describe transformations that map an XML document to an other XML document or different format. Typically, an XSL transformer (XSLT) is used to apply a stylesheet to an XML document.

References

[1] Dafermos GN. Management and virtual decentralised networks: The Linux project. *First Monday*, 6(11), 2001.

[2] Hill B. Free software project management HOWTO, 2002.

[3] Torvalds L. The Linux edge. *Communications of the ACM*, 42(4):38–39, 1999.

[4] Raymond ES. The cathedral and the bazaar. *First Monday*, 3(3), 1998.

[5] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3.

[6] Wu H, Kerr MK, Cui X, and Churchill GA. Maanova: A software package for the analysis of spotted cdna microarray experiments. In G. Parmigiani, E. Garrett, R. Irizarry, and S. Zeger, editors, *The analysis of gene expression data: methods and software*, pages 313–341. Springer, NY, 2003.

[7] Li C and Wong WH. Model based analysis of oligonucleotide arrays: Expression index computation and outlier detection. *PNAS*, 98:31–36, 2001.

[8] Chambers JM. *Programming with Data: A Guide to the S Language*. Springer-Verlag New York, 1998.

[9] Box D, Ehnebuske D, Kakivaya G, Layman A, Mendelsohn N, Nielsen HF, and Thatte S. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/.

[10] Stein L. Creating a bioinformatics nation. *Nature*, 417:119–120, 2002.

[11] Mascagni M, Ceperley DM, and Srinivasan A. Sprng: A scalable library for parallel pseudorandom number generation. In H. Niederreiter and J. Spanier, editors, *Monte Carlo and Quasi-Monte Carlo Methods 1998*. Springer Verlag, Berlin, Germany, 2000.

[12] Rossini AJ, Tierney L, and Li M. Simple parallel statistical computing in R. *Submitted*, 2003. also: University of Washington Biostatistics Technical Report #193, http://www.bepress.com/uwbiostat/paper193.

[13] Li M and Rossini AJ. Rpvm: Cluster statistical computing in R. *RNews*, 1(3):4–7, 2001.

[14] Steele GL. *Common LISP: The Language*. Butterworth-Heinemann, 1990.

[15] Shalit A, Starbuck O, and Moon D. *Dylan Reference Manual*. Addison-Wesley, 1996.

[16] Leisch F. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physika Verlag, Heidelberg, Germany, 2002. ISBN 3-7908-1517-9.

[17] Purdy GN. *CVS Pocket Reference*. O'Reilly & Associates, 2000.

[18] R Development Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-04-6.

[19] Siek JG, Lee LQ, and Lumsdaine A. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2001.

[20] Mei H, Tarczy-Hornoch P, Mork P, Rossini AJ, Shaker R, and Donelson L. Expression array annotation using the biomediator biological data integration system and the bioconductor analytic platform. In *Proceedings of AMIA 2003*. American Medical Informatics Association, 2003.

[21] Raymond ES. Software release practice HOWTO, 2002.

[22] Buckheit J and Donoho DL. Wavelab and reproducible research. In A. Antoniadis, editor, *Wavelets and Statistics*. Springer-Verlag, 1995.

[23] Gentleman R and Temple Lang D. Statistical analyses and reproducible research. *Submitted*, 2003.

[24] Rossini AJ and Leisch F. Literate statistical practice. *Submitted*, 2002. also: University of Washington Biostatistics Technical Report #194, http://www.bepress.com/uwbiostat/paper194.

[25] Schwab M, Karrenbach M, and Claerbout J. Making scientific computations reproducible. Technical report, Stanford University, Stanford Exploration Project, 1996.

[26] Zhang J, Carey V, and Gentleman R. An extensible application for assembling annotation for genomic data. *Bioinformatics*, 19:155–56, 2003.