The anatomy of successful computational biology software

Stephen Altschul[1], Barry Demchak[2], Richard Durbin[3], Robert Gentleman[4], Martin Krzywinski[5], Heng Li[6], Anton Nekrutenko[7], James Robinson[6], Wayne Rasband[8], James Taylor[9] & Cole Trapnell[10]

Creators of software widely used in computational biology discuss the factors that contributed to their success

The year was 1989 and Stephen Altschul had a problem. Sam Karlin, the brilliant mathematician whose help he needed, was so convinced of the power of a mathematically tractable but biologically constrained measure of protein sequence similarity that he would not listen to Altschul (or anyone else for that matter). So Altschul essentially tricked him into solving the problem stymying the field of computational biology by posing it in terms of pure mathematics, devoid of any reference to biology. The treat from that trick became known as the Karlin-Altschul statistics that are a key part of BLAST, arguably the most successful piece of computational biology software of all time.

Nature Biotechnology spoke with Altschul and several other originators of computational biology software programs widely used today (**Table 1**). The conversations explored what makes certain software tools successful, the unique challenges of developing them for biological research and how the field of computational biology, as a whole, can move research agendas forward. What follows is an edited compilation of interviews.

[1]National Center for Biotechnology Information, Bethesda, Maryland. [2]University of California San Diego, La Jolla, California. [3]Wellcome Trust Sanger Institute, Hinxton, UK. [4]Genentech, South San Francisco, California. [5]British Columbia Cancer Research Centre, Vancouver, Canada. [6]Broad Institute, Cambridge, Massachusetts. [7]Penn State University, University Park, Pennsylvania. [8]National Institute of Mental Health, Bethesda, Maryland. [9]Emory University, Atlanta, Georgia. [10]Harvard University, Cambridge, Massachusetts.

What factors determine whether scientific software is successful?

Stephen Altschul: BLAST was the first program to assign rigorous statistics to useful scores of local sequence alignments. Before then people had derived many different scoring systems, and it wasn't clear why any should have a particular advantage. I had made a conjecture that every scoring system that people proposed using was implicitly a log-odds scoring system with particular 'target frequencies', and that the best scoring system would be one where the target frequencies were those you observed in accurate alignments of real proteins.

Stephen Altschul co-developed BLAST.

It was the mathematician Sam Karlin who proved this conjecture and derived the formula for calculating the statistics of the scores [E-values] output by BLAST. This was the gravy to the algorithmic innovations of David Lipman, Gene Myers, Webb Miller and Warren Gish that yielded BLAST's unprecedented combination of sensitivity and speed.

Another great aspect of the popularity of BLAST was that over time it was seamlessly linked to NCBI's sequence and literature databases, which were updated daily. When we developed BLAST, the databases available were in relatively poor shape. In many instances, you had to wait for over a year between the publication of a paper and when its sequences appeared in a database. A lot of very talented and dedicated people worked to construct the infrastructure at NCBI that allowed you to search up-to-date databases online.

Cole Trapnell: Probably the most important thing is that Cufflinks, Bowtie (which is mainly Ben Langmead's work) and TopHat were in large part at the right place at the right time. We were stepping into fields that were poised to explode, but which really had a vacuum in terms of usable tools. You get two things from being first. One is a startup user base. The second is the opportunity to learn directly from people what the right way, or one useful way, to do the analysis would be.

Cole Trapnell developed the Tophat/Cufflinks suite of short-read analysis tools.

Heng Li (developed MAQ, BWA, SAMtools and other genomics tools): I agree timing is important. When MAQ came out, there was no other software that could do integrated mapping and SNP [single-nucleotide polymorphism] calling. BWA was among the first batch of Burrows-Wheeler–based aligners (BWA, Bowtie and SOAP2 were all developed at about the same time). Similarly, SAMtools was the first generic SNP caller that worked with any aligner, as long as the aligner output SAM format.

Robert Gentleman is co-creator of the R language for statistical analyses.

Robert Gentleman: The real big success of R, I think, was around the package system. Anybody that wanted to could write a package to carry out a particular analysis. At the same time, this system allowed the standard R language to be developed, designed

and driven forward by a core group of people.

For Bioconductor, which provides tools in R for analyzing genomic data, interoperability was essential to its success. We defined a handful of data structures that we expected people to use. For instance, if everybody puts their gene expression data into the same kind of box, it doesn't matter how the data came about, but that box is the same and can be used by analytic tools. Really, I think it's data structures that drive interoperability.

Wayne Rasband: Several factors have contributed to the usability of ImageJ. First, it has a relatively simple graphical user interface, similar to popular desktop software, such as Photoshop. Second, there is a large community of users and developers willing to answer questions, contribute plugins and macros, and find and fix bugs. Third, because it is written in Java, ImageJ runs on Linux, Macs and Windows. And fourth, ImageJ is open source, so users can inspect, modify and fix the source code.

Wayne Rasband developed the ImageJ image analysis software.

Richard Durbin: I think a key thing is that software or a data format does a clean job correctly, that it works. For the software I've been involved with, I think support isn't a critical thing, in a strange way. Rather, it's the lack of need for support that's important.

Richard Durbin led the development of many tools and data standards in genomics.

Also, I have always wanted to use the software I developed, or my group has wanted to use it to do our own job. John Sulston taught me to try to write something that does the best possible job for yourself and enables others to see what you would want to get out of the data. Don't think that you'll produce one version for yourself and then somehow have a different tool for others. For example, the people who built the *C. elegans* physical maps in the '80s made the same software used to build the maps available to external scientists so they could look at the evidence and the data. It's important, I think, to generate data at the same time as writing software for those data, being driven by problems that are at hand.

Does scientific software development differ from that for other types of software?

Durbin: Scientific software often requires quite a strong insight—that is, algorithmic development. The algorithm implements novel ideas, is based on deep scientific understanding of data and the problem, and takes a step beyond what has been done previously. In contrast, a lot of commercial software is doing specific cases of fairly straightforward things—book-keeping and moving things around and so on.

Gentleman: I have found that real hardcore software engineers tend to worry about problems that are just not existent in our space. They keep wanting to write clean, shiny software, when you know that the software that you're using today is not the software you're going to be using this time next year. At Genentech (S. San Francisco, California), we develop testing and deployment paradigms that are on somewhat shorter cycles.

Table 1 Software for the ages

Software	Purpose	Creators	Key capabilities	Year released	Citations[a]
BLAST	Sequence alignment	Stephen Altschul, Warren Gish, Gene Myers, Webb Miller, David Lipman	First program to provide statistics for sequence alignment, combination of sensitivity and speed	1990	35,617
R	Statistical analyses	Robert Gentleman, Ross Ihaka	Interactive statistical analysis, extendable by packages	1996	N/A
ImageJ	Image analysis	Wayne Rasband	Flexibility and extensibility	1997	N/A
Cytoscape	Network visualization and analysis	Trey Ideker *et al.*	Extendable by plugins	2003	2,374
Bioconductor	Analysis of genomic data	Robert Gentleman *et al.*	Built on R, provides tools to enhance reproducibility of research	2004	3,517
Galaxy	Web-based analysis platform	Anton Nekrutenko, James Taylor	Provides easy access to high-performance computing	2005	309[b]
MAQ	Short-read mapping	Heng Li, Richard Durbin	Integrated read mapping and SNP calling, introduced mapping quality scores	2008	1,027
Bowtie	Short-read mapping	Ben Langmead, Cole Trapnell, Mihai Pop, Steven Salzberg	Fast alignment allowing gaps and mismatches based on Burrows-Wheeler Transform	2009	1,871
Tophat	RNA-seq read mapping	Cole Trapnell, Lior Pachter, Steven Salzberg	Discovery of novel splice sites	2009	817
BWA	Short-read mapping	Heng Li, Richard Durbin	Fast alignment allowing gaps and mismatches based on Burrows-Wheeler Transform	2009	1,556
Circos	Data visualization	Martin Krzywinski *et al.*	Compact representation of similarities and differences arising from comparison between genomes	2009	431
SAMtools	Short-read data format and utilities	Heng Li, Richard Durbin	Storage of large nucleotide sequence alignments	2009	1,551
Cufflinks	RNA-seq analysis	Cole Trapnell, Steven Salzberg, Barbara Wold, Lior Pachter	Transcript assembly and quantification	2010	710
IGV	Short-read data visualization	James Robinson *et al.*	Scalability, real-time data exploration	2011	335

[a]Citations in Web of Science as of September 2013 of the first paper describing the software. [b]A comprehensive list of publications (1,150 as of September 2013) related to Galaxy is at http://www.citeulike.org/group/16008/order/group_rating. N/A, paper not available in Web of Science.

James Taylor developed the Galaxy platform.

James Taylor: A lot of traditional software engineering is about how to build software effectively with large teams, whereas the way most scientific software is developed is (and should be) different. Scientific software is often developed by one or a handful of people.

Barry Demchak (lead Cytoscape software architect): The status quo of software development in the 1990s is where computational biologists are today—for loops, variables and function calls. Computer science has moved on, particularly in three areas: functional programming, service-oriented architectures and domain-specific languages.

What misconceptions does the research community have about software development or use?

Trapnell: I think the biggest misconception is that something you put on the Internet for people to use is a finished product. Each version of Cufflinks and Tophat, for instance, offered performance improvements and bug fixes. But they often also had substantial new features that actually reflected a new understanding about what's going on in the computer science and the mathematics and the statistics that are attached to RNA-seq. Really fundamental stuff. The way that translates into the software that people use is that they download one version and they run the analysis and then they upgrade and then the results change, sometimes a little bit, sometimes a lot. That creates the impression, which I think is the wrong impression, that one or both of those sets of results is just totally wrong. People don't, I think, frequently understand just how much those programs are research projects that are continually evolving.

James Robinson: Perhaps that the software is more sophisticated than it actually is, leading to too much faith in the results without critical thinking. For analysis software, such as mutation calling, it's important to know at some level what the algorithms are, the biases in them, what they assume and how they fail. Visualizing algorithmic output with a critical eye in a program, such as IGV or the UCSC browser, can help with this. However, it is also important to understand what the developers of the visualization have chosen to emphasize, through the use of color and other techniques, and what they have chosen to de-emphasize.

James Robinson developed the Integrative Genomics Viewer.

Martin Krzywinski: In a way, a fixed visualization is only one answer. It's one projection, one encoding, one view of the data. Depending on the complexity of the data and the number of dimensions, there are many views. We have to accept that what we're seeing is akin to a shadow on the wall. An object can cast many different shadows, depending on its shape. We can't look at the shadow and say that that's the object. We have to remember that that's the shadow of the object and that the object has some higher dimensional properties.

Martin Krzywinski developed the Circos data visualization tool.

Li: People not doing the computational work tend to think that you can write a program very fast. That, I think, is frankly not true. It takes a lot of time to implement a prototype. Then it actually takes a lot of time to really make it better.

Demchak: Quite often, users don't appreciate the opportunities. Noncomputational biologists don't know when to complain about the status quo. With modest amounts of computational consulting, long or impossible jobs can become much shorter or richer.

Are too many new software tools developed that ultimately don't get used?

Durbin: This is kind of a debate of top down versus bottom up. In science, always there are lots of people looking at the same thing in different ways. There are people trying out all sorts of crazy things. It's extremely successful to not have top-down control. It can look a little bit redundant when you have a person write yet another read mapper, but sometimes things will be influential. New ideas will come. Sometimes things can be relevant to individual projects. I think for sure things are done inefficiently. I accept that. It's a bit like evolution. Random mutation and testing is very powerful.

Trapnell: Maybe the way to look at it is the software that gets produced is, in a sense, the piece of supporting data for those papers, and isn't necessarily even meant for adoption by a community. It's more a vehicle for producing data to argue that a computational method is sound or that it has the properties that are being claimed.

Gentleman: I'll point to the 'bump hunting' tools for finding peaks in [chromatin immunoprecipitation] ChIP-seq data. There must be a hundred of those. Why are there so many? They either all work equally well, and it doesn't matter which one you use. Or, each one of them does something that's a little bit different, and we simply have not figured out how to decide which one is best. I argue that it's more the latter than the former. What's missing is, 'How do we generate large data sets with enormous numbers of false positives and false negatives?' You need a sufficiently big and complicated data set, where you know the truth, to understand whether one method is better than another, whether I'm getting exactly the same answer but I'm just getting it faster or whether I'm getting different answers that are both flawed. Those sorts of things are part of what can help you drive from a diversity of computational tools down to a relative few that work better.

Taylor: I don't think there are good incentives for contributing to and improving existing software instead of inventing something new. The latter is more likely to be publishable. There is also a problem with discovering software that exists; often people reinvent the wheel just because they don't know any better. Good repositories for software and best practice workflows, especially if citable, would be a start.

Anton Nekrutenko (co-creator of Galaxy): This is the key idea behind the Galaxy Tool Shed, our app store. So far, it contains about 2,700 tools. The goal of this mechanism is to make it easy to try each tool and then vote on which ones perform well.

How is the field of computational biology evolving?

Durbin: Now there are a lot of strong, young, faculty members who label themselves as computational analysts, yet very often want wet-lab space. They're not content just working off data sets that come from other people. They want to be involved in data generation and experimental design and mainstreaming computation as a valid research tool. Just as the boundaries of biochemistry and cell biology have kind of blurred, I think the same will be true of computational biology. It's going to be alongside biochemistry, or molecular biology or microscopy as a core component.

Box 1 Does every new biology PhD student need to learn how to program?

Durbin: That's a little like asking is it good for a molecular biologist to know chemistry. I would say that computation is now as important to biology as chemistry is. Both are useful background knowledge. Data manipulation and use of information are part of the technology of biology research now. Knowing how to program also gives people some idea about what's going on inside data analysis. It helps them appreciate what they can and can't expect from data analysis software.

Trapnell: It's probably not just that experimental biologists need to program, but it's also enormously helpful when computational folks learn how to do experiments. For me, for example, coming from a computer science background, the opposite way of thinking was hard to learn. How do I learn to argue with wet-lab data? How do I learn what to trust, what to distrust, how to cross-validate things? That's a radically different way of thinking when you're used to proofs and writing code and validating it on a computer.

Krzywinski: To some, the answer might be "no" because that's left to the experts, to the people downstairs who sit in front of a computer. But a similar question would be: does every graduate student in biology need to learn grammar? Clearly, yes. Do they all need to learn to speak? Clearly, yes. We just don't leave it to the literature experts. That's because we need to communicate. Do students need to tie their shoes? Yes. It has now come to the point where using a computer is as essential as brushing your teeth. If you want some kind of a competitive edge, you're going to want to make as much use of that computer as you can. The complexity of the task at hand will mean that canned solutions don't exist. It means that if you're using a canned solution, you're not at the edge of research.

Robinson: Yes. Even if they don't program in their research, they will have to use software and likely will communicate with software developers. It helps tremendously to have some basic knowledge. Additionally, in the research environment, the ability to do basic tasks in the Linux/Unix environment is essential.

Rasband: All scientists should learn how to program.

Nekrutenko: Many people can learn how to program in C, but they still write horrific code that nobody can understand. Most of the biology graduate students who can program, they're more dangerous than people who cannot program because they produce these things. It's horrible, but that's what you expect from a new field. It will change, and it needs to change just through graduate education. For example, at Penn State, we have a scientific programming course designed for life science people with sequence analysis in mind. It builds on 'software carpentry' [http://software-carpentry.org/] by teaching people that you need to version your software. You need to write tests. All these skills, that's the missing part.

Trapnell: If you break down past work in computational biology, there've been a couple of historically really rich areas. One of them stems from sequence alignment. From there, you get paleogenetics and certain molecular evolution studies. Then with microarrays, you had the advent of genomics as a measurement science, where you're actually trying to measure something about what's happening in some samples. With DNA sequencing, we are seeing the convergence of those two things. You get all of the possibilities in terms of statements you might make with alignment and the stuff that follows from that, but you also get all the crazy statistical issues and numerical analysis problems that arise when you're making quantitative measurements of biological activity. I think the end-stage result is that, now, sequencing is used not as a cataloging technology, but really as a routine, day-to-day measurement technology. That just reorients, I think, the baseline computational skill set that everybody needs in order to deal with that kind of data (**Box 1**). The computational folks need to learn more about statistics. The biology folks need to understand basic computation in order to even be able to communicate with the biostatistics crowd.

What are the emerging trends in computational biology and software tools?

Robinson: The emergence of sequencing, and even very high–resolution SNP and expression platforms, means that virtually all computational biologists working with human samples now need to understand and deal with implications of individual identification and privacy.

Trapnell: The areas of computer science that will be needed to solve these privacy issues are ones that biologists have never even been exposed to. It has to do with secret sharing and public-key cryptography and all these other areas that we just never worry about because they don't come up. Now they're coming up in a huge way. So I would expect that that is going to be a major driver of a lot of serious computational work.

Taylor: Crowdsourcing is potentially a deep trend. We've seen a lot with people having success with verification of results. If we can develop infrastructures to allow greater participation and to take advantage of large communities, the decision-making capabilities of groups is going to be a continuing trend.

Gentleman: At Genentech, we have petabytes of data, but it's not 'big data' like at Amazon or Walmart or in the airline industry. Our problem is that we have lots of little, tiny files that have all sorts of complicated information in them, and a few big files with complicated information in them. How to deal with that is a different problem. We start with data in one format. We run it through a very complex set of transformations in a very complicated computing environment. Tracking it, knowing which output of which version of which tool we're actually going to use, and being sure that something that we started actually finished—those are the really complex cases for us. Those sorts of organizational details are challenging to get right, but I think most academics don't have the problem on the same scale.

Krzywinski: In terms of data visualization, the idea that we can show all the data that we are collecting is long gone. We now need to look at the differences in the data sets, and help the user focus on the things that are important. Differences, and differences of differences, are now the data. In addition, you cannot dump that output from a program on a user, otherwise they will become lost in this sea of detail. I think what software needs is a series of output filters that can be used to select for the level of detail in the output.

Durbin: I once heard Nathan Myhrvold and Sydney Brenner talk about "exponential technologies," which were all characterized by providing an exponential increase in information. Sequencing is like that. In the future, I think we will get into cell biology, away from the genome. I think it's going to be done through high-throughput data acquisition from instrumentation of cell biological measurements of some sort. I'm not exactly sure how, but I'm interested, and excited by that.

Corrected after print 9 May 2014.

Erratum: The anatomy of successful computational biology software

Stephen Altschul, Barry Demchak, Richard Durbin, Robert Gentleman, Martin Krzywinski, Heng Li, Anton Nekrutenko, James Robinson, Wayne Rasband, James Taylor & Cole Trapnell

***Nat. Biotechnol.* 31, 894–897 (2013); published online 8 October 2013; corrected after print 9 May 2014**

In the version of this article initially published, in Table 1, Steven Salzberg should have been listed as the second, and not the last, of the creators of the Cufflinks software. The error has been corrected in the HTML and PDF versions of the article.