Code as Text: Open Source Lessons for Geospatial Research and Education

Sergio J. Rey

Introduction

The open source revolution continues to have major impacts on science and education and the field of spatial analysis is no exception. A number of overviews of open source spatial analysis and geographic information science have recently appeared[1] and my intent here is not to provide a similar comprehensive coverage of this area but rather to expand upon a particular set of themes I have raised previously [11]. I do so by drawing on the lessons learned in the development and evolution of the PySAL project [13] as it has intersected with my teaching and research activities.

My central claim is that while open source has attracted much interest in geospatial education and research its potential to enhance our activities has been constrained by a lack of understanding of how open source communities function and the differences that exist between these communities and those found in the academic and scientific worlds. In broad terms, the excitement around open source in academia is dominated by the cost advantages Free/Libre Open Source Software (FLOSS) offers to our teaching and research missions. While these are important and very real, by focusing on these we miss the forest for the trees. The true value of open source is its potential to revolutionize and fundamentally enhance geospatial education and research. I argue that this will only be possible if instead of seeing

[1]For overviews see [4, 8, 10, 11, 15, 16].

S.J. Rey (✉)
Center for Geographical Information Science, School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ, USA
e-mail: Sergio.Rey@asu.edu; srey@asu.edu

© Springer International Publishing AG 2018
J.-C. Thill, S. Dragicevic (eds.), *GeoComputational Analysis and Modeling of Regional Systems*, Advances in Geographic Information Science,
DOI 10.1007/978-3-319-59511-5_2

open source code as only a tool to do research, we see the *code as text* and an object of research as well as a pathway to better geospatial education and research.

I begin with an overview of PySAL, discussing its origins, development, and current state, and I share several lessons I've learned as an open source developer living inside academia. Next, I focus on the role of open source in geospatial education. I then discuss the impacts of open source for research in geospatial analysis. I conclude the paper by identifying key areas of future concern and opportunity.

PySAL

Origins

PySAL's lineage can be traced back to two earlier projects, Space-Time Analysis of Regional Systems (STARS) [14] a package I developed with my students at San Diego State University, and PySpace which was Luc Anselin's project developed at the University of Illinois, Urbana Champaign [1]. STARS was designed to handle exploratory space-time data analysis while PySpace focused on spatial econometrics. Although they had different foci, the two projects relied on similar data structures (primarily spatial weights matrices), certain algorithms and statistical tests.

Collaboration between the project directors led to the realization that by pooling the development activities of our two teams, we could move these common features into a single library, rather than continuing to duplicate efforts. Additionally, such collaboration could allow for a more focused and optimized implementation of the core shared components and free up time for each of the respective projects to specialize on features that were unique to the individual package.

A second motivation for creating the library was that, at the time, spatial analysis was largely absent in the Python scientific computing community. There were some early efforts of packages for data integration (ogr), map projections (pyproj) and geoprocessing (shapely), but at the higher end of the spatial analysis stack there were no Python packages to support exploratory spatial data analysis and spatial econometrics. As Python was having major impacts elsewhere in scientific computing, and was starting to make in roads in GIS as reflected by ESRI's adoption of Python as a scripting language, we wanted to both contribute to further adoption of Python within the GIScience community but also fill the void of missing spatial analytical tools in the wider Python scientific computing portfolio.

Initial discussions about the design of the library laid out a comprehensive coverage of many areas of spatial analysis that not only included the feature sets in STARS and PySpace but an expanded vision to cover a broad set of methods, data structures and algorithms in the spatial analysis toolkit. Parallel to this coverage of the components in the library we also felt that the library should support a number of different types of uses. As a Python library PySAL could be used through imports at the Python interpretor to facilitate interactive computing at the command line. A second intended delivery mechanism was to use PySAL to develop rich desktop clients in the mode of GeoDa and STARS. Here the analytical engine of the application would be based on methods from PySAL, while advanced graphics toolkits, such as WxPython, could be used to implement fully interactive and dynamic graphics. A third way we envisioned the library being used was to build plugins or toolkits for other packages, for example ArcGIS, QGIS or GRASS. The fourth delivery mechanism we identified was to provide access to PySAL through distributed web processing services [9].

Early on in the implementation of the library we began to realize the advantages that adopting Python for this project would offer. Python is a dynamically typed scripting language which lends itself nicely to rapid prototyping of ideas which radically shortens the distance between the germ of an idea and its articulation in working code. Python also has a clean syntax which facilitates collaboration by making the implementation of algorithms and spatial analytical methods transparent as the code becomes text, a point I return to later.

Components

Figure 1 displays a schematic from the early design of PySAL. The key departure point for development of the library was the spatial weights module. Spatial weights are central to many areas of spatial analysis as they formalize the notion of neighbor relations governing potential spatial interaction between locations. Having efficient data structures to store, create, operate, and transform spatial weights is critical to the entire library and thus the weights module became the dominant focus early in PySAL's implementation.

A second building block in the library is the computational geometry module which provides a number of algorithms and data structures for spatial tessellations (Voronoi), minimum spanning trees, convex hulls, binning and R-trees which are required by several of the other modules in PySAL. For example, the construction of contiguity based spatial weights from shapefiles uses R-trees, or distance based weights using K-nearest neighbor algorithms relies on KD-trees.

The clustering module provides methods used to carry out spatially constrained regionalization as in the case of defining neighborhoods in geodemographics and urban analysis, or aggregating spatial units to satisfy some minimum threshold value when estimating disease rates in spatial epidemiology. The exploratory spatial data analysis module implements methods for global and local spatial autocorrelation analysis which includes enhancements to deal with rates and spatial smoothing. Methods for spatial dynamics form the fifth PySAL module and extend the class of Markov based methods from STARS to include LISA Markov chains, conditional and joint spatial Markov chains, and directional space-time indicators. Finally,

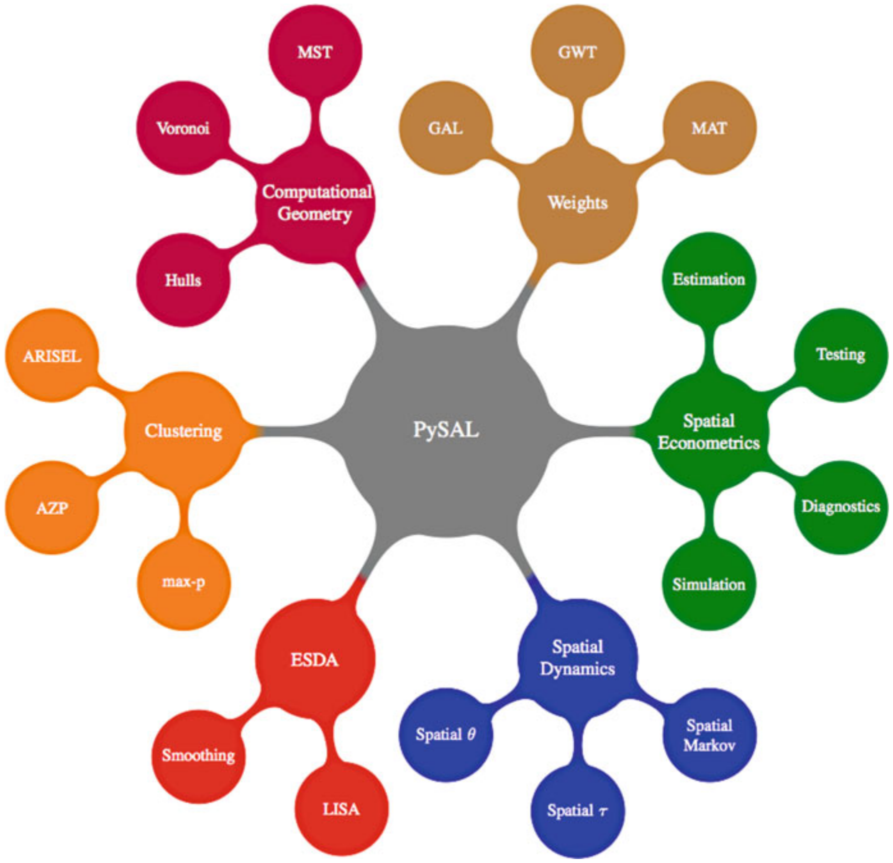

Fig. 1 PySAL components

methods for the testing, estimation, and validation of spatial regression models are contained in the spatial econometrics module.

The first official release of PySAL was in July 2010. We placed this release under the Berkeley Software Distribution (BSD) license since one of our goals for PySAL was to contribute the scientific computing stack in Python and BSD was the dominant and preferred license in this community. At the time of writing PySAL is in its 9th stable release (1.8), with the next formal release scheduled for January 2015. The project is housed at GitHub.[2] Since the first official release PySAL has been downloaded over 50,000 times and we are quite pleased with the reception of the library for the community. An important reflection of this reception is the inclusion of PySAL as a featured package in the leading Python distributions for scientific computing Anaconda Python Distribution [5] and Enthought Canopy [6].

[2]https://github.com/pysal.

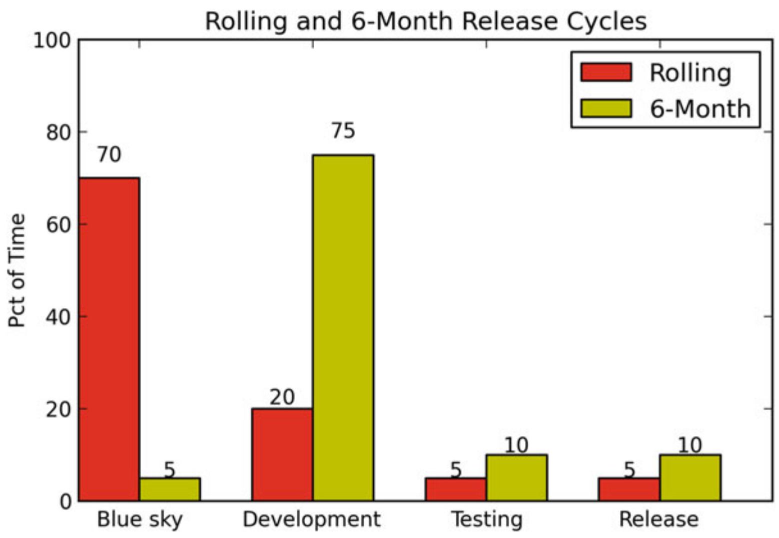

Fig. 2 Effort distribution under flexible and time based (release cycle) schemes

Lessons for Academic Open Source Developers

PySAL was born, and remains housed, in academia and the experience of managing an open source project inside of a university setting has been a rewarding and, at times, challenging experience. Because I expect more open source projects to find homes in academia in the future, I think it may be valuable to share a number of lessons that I've learned as an open source developer working inside academia.

Beginning with the first release (version 1.0) in July 2010 we adopted a 6 month release cycle. These cycles consist of several phases that begin with a 2 week period of considering PySAL Enhancement Proposals[3] (PEPs), followed by a developer vote on the PEPs that get priority for the current release cycle. The next phase of the cycle is the development component which lasts 14 weeks. The final month of the cycle is a feature freeze where attention shifts to bug fixes, testing, documentation and preparation of the release.

Prior to adopting the formal 6 month release cycle, work on PySAL and related projects (STARS, PySpace) followed no formal release schedule. We put out releases whenever we felt they were ready for public consumption, or in order to meet a deadline for a grant deliverable. I think this more flexible, less structured approach towards releases is more common in academic open source projects than the formal time-based scheme, and it is interesting to contrast the two (Fig. 2).

[3]We borrowed the PEP concept from the Python Enhancement Proposal model used in the development of Python: http://legacy.python.org/dev/peps/.

When operating under the informal release scheme, we (the developers) spent a large portion of the time engaged in design discussions about the new features, algorithms and related code architecture. These were very enjoyable discussions to be part of as they often generate a great deal of excitement and many fascinating ideas about what we could do. As an analogy to a research project, I would liken this to the phase where you are sitting with a potential collaborator discussing possible ideas over a beer. Here the sky is the limit and you can easily get caught up in the excitement of what is possible.

The excitement, however, can seem to evaporate when it comes time to actually write the proposal and, if successful, start to implement your grand ideas that not so long ago seemed so beautiful and seductive but now, in the face of the realities of pending deadlines and implementation challenges, take on a less enticing aura.

Given that the design and discussion phase was so engaging, while the implementation phase is much more like "work", it is not surprising that in the informal release scheme we spent a lot more time in the former and less in the latter. Unfortunately this meant that a lot of the ideas that drove the exciting design discussions never materialized in actual code.

When we shifted to the time-based release cycle we designed the lengths of the different phases to counteract this weakness. Basically we adopted the mantra of:

> You can do anything, but you can't do everything.
> *David Allen*

In other words, the blue sky discussions (i.e., you can do anything) were limited to the first 2 weeks of the cycle, and had to be formulated in a PEP which are essentially problem statements for the new feature. Voting on and prioritizing the PEPs for the release cycle reflected the recognition that we could not, in fact, do everything we might wish to do, and therefore we simply had to choose among all of our children which ones would be our favorites for the cycle.

The results of moving from the flexible to time-based release cycle have been very positive. The regular release schedule gives our users the ability to plan for any changes that they may want to make in light of new pending features in the library. For the developers it allows us to focus our energies on a subset of the features, resulting in more complete and better implementations of these in the release. Essentially we are sacrificing breadth for depth here.

In terms of the length of the release cycle, I think 6 months hits the sweet spot for an open source project inside academia, as we can align the ending of the cycle to coincide with the winter and summer breaks so that those release deadlines can be given full attention. Shorter cycles would mean release deadlines would compete against other deadlines during the academic year, while longer cycles (say 1 year) would slow the development of the project since new features could only be added once a year.

Our experiences with PySAL are likely similar to those of other projects that are housed inside academia. As is becoming increasingly recognized, much research code that is used in science is far from what could be called production quality software. Often it is written to get results for a particular project and then the researcher moves on to the next paper. What is typically missing are critical features such as documentation and testing that ensure the code could be used by others for purposes of replication and reproducibility.

Often the finger is pointed at the publish or perish dilemma as the main pressure leading to the rather poor state of research code. Less recognized, but arguably as important, is that most academics learn programming on the job rather than through a formalized sequence of courses in a degree program. As a result, even if the pressures to publish were absent and the researcher had time to document and test the code, they often lack the understanding of how to do so. To be fair, this lack of software engineering skills on the part of academic researchers could also be laid against many open source and proprietary projects outside of academia—many developers are self-taught. Equipping researchers with proper software development skills is a critical need that I return to below.

Lessons for Education

> It goes against the grain of modern education to teach students to program. What fun is there to making plans, acquiring discipline, organizing thoughts, devoting attention to detail, and learning to be self critical.
> *Alan Perlis*

Open source software and practices can have major empowering impacts on pedagogy. The free availability of the software offers a number of advantages in lab based courses. No longer are the students constrained to working in the school laboratory as they can now use the software installed on their own personal laptops, or home desktops, to complete exercises. This also allows for a greater degree of exploration and discovery by the student working by themselves and at their own pace.

These represent *potential* pedagogical wins for open source in geospatial education. My recent personal experience is that we still have far to go before these benefits are fully realized. During the fall of 2011 in my introductory course in GIScience, I decided to use QGIS as the software for the lab component in place of our traditional package of ArcGIS. This was something I had contemplated doing for quite sometime, but I always held back as the feature set and polish of QGIS were not yet at the stage where I felt comfortable doing so. By fall 2011, this had changed as the development of QGIS had reached an impressive state.

To my surprise this switch was less than well received by the students. Emblematic of the main complaint was the following comment I received on an anonymous teaching evaluation:

> I took this course as I heard we would be taught ArcGIS. I don't care about the science and the algorithms underneath the software, I want a job when I leave this class.
> *Anonymous student evaluation*

While there were a minority of students who told me they appreciated the introduction to an open source alternative, the vast majority of the students were not happy about the switch. In addition, I received push-back from some of my colleagues who were concerned that not covering ArcGIS threatened relationships with community internship partners that had been carefully cultivated over the years.

I was completely blindsided by these responses and felt a mixture of disappointment and puzzlement. In hindsight, I admit that these potential negative impacts never entered my decision making calculus. At the same time, while I now see that these are pressing concerns, they also raise some important questions regarding the role of geospatial education. On the one hand, the current demands in the labor market for students trained in ArcGIS reflects the reality that previous generations of students we have trained in this software are now in key positions in these agencies and companies. Additionally, many of these agencies have invested much time and resources in their GIS infrastructures and are understandably conservative regarding any changes. But, what about the future? Is our task to train students for today's labor market or to equip them with the skill sets and knowledge so that they are ready for, and can create, the future geospatial labor market?

A second general lesson for geospatial education concerns graduate education and the seemingly ironic situation of an embarrassment of riches in terms of freely available high quality programming tools for geospatial research on the one hand and, on the other, a general lack of desire to do any programming. I believe this stems from the challenges facing geography graduate students in that they not only need to acquire knowledge of substantive and methodological areas of the discipline but also somehow become proficient in programming. We have done a fairly poor job on the latter with solutions ranging from recommending introductory courses in computer science departments to learning on the job as part of a research project. The former is rather inefficient as my experience is geography students taking most introductory computer science classes come away without any idea of how to apply core concepts to geographical problems. The mentoring approach scores higher on this point, but does not really scale well.

There are several possible ways to address these issues. One approach I have adopted is to create a new course entitled "Geocomputation" that blends together both a primer on Python and open source tools, such as Git and text editors (Vim), together with formal lectures on a selection of spatial algorithms and their application to course projects. Open source tools, while very powerful, can have steep learning curves and a key motivation for this course is to flatten these curves. Thus far the course has been very well received as it equips the students with skill sets that are directly useful to their own thesis and dissertation research.

The course also offers a path to integrate PySAL into the curriculum as the library offers a rich set of possible topics to both use in lecture as well as to form core components of student projects. Stepping into an existing project gives the student hands-on experience with a large scale research code project, and rather than having to develop their own projects from scratch, they can choose from the ever expanding list of feature requests (and hopefully declining bug reports) for PySAL as their project topics.

It is here that I have seen the impact of the two key freedoms associated with FLOSS on geospatial education. The "free as in beer" freedom has already been alluded to since the students are free to download the software. This also is becoming increasingly important to educational institutions in an era of tightening budgets. The second freedom derives from the "free as in speech" aspect of FLOSS which means the code is now available for reading. Here seeing the code as text is enhanced in powerful ways by the free as in speech nature of FLOSS software and the use of Python. Students are empowered to think about the computational concepts and, due to the interpreted nature of Python and its clean syntax, they are unencumbered by the technical issues of compiling and linking that would be encountered in other languages hindering the learning process.

> You think you know when you can learn, are more sure when you can write, even more when you can teach, but certain when you can program.
> *Alan Perlis*

By coming to see the code as text, rather than as a black box, students' engagement with the fundamental concepts is deepened in a way that is simply not possible with closed source software. In my geocomputation course I endeavor to have the students come to see geospatial methods as not only tools they can use in their own research, but as possible subjects for research. For too long now the view in most geography departments has been that spatial analysis was something you use to do research, rather than something you do as research. We have only given scant attention towards nurturing the next generation of geospatial researchers who will produce the future advances in our fields. To facilitate the latter we have to affect a mind shift to see code as text.

There is, however, only enough demand at my own institution to offer a course like Geocomputation no more than every other year which leaves students entering our graduate program in the off year at a disadvantage as acquiring these skills early on in their studies is clearly desirable. A recent development in so called massively open and on-line courses (MOOC) offers some interesting possibilities in this regard. My experience in teaching PySAL workshops is that there is ample demand for these types of courses in the broader community. Offering such a course in the mode of a MOOC provides a mechanism to attract a staggering number of participants and could be a way to allow students at my own institution the possibility of taking the course each year. While there has been much debate about the impact of MOOCs on higher education, I am excited by the potential to reinvent the role of pedagogy at large research institutions as now it becomes possible to turn what could currently be seen as a boutique course into a staple offering.

Another attractive possibility for open source geospatial education can be seen in the Software Carpentry initiative [18]. Software Carpentry grew out of the recognition that most research scientists lacked basic software skills for scientific computing. Founded in 1988, the mission of Software Carpentry is to teach scientists basic lab skills for scientific computing through concentrated 2-day boot camps that cover topics such as shells, editors, code repositories and other technologies that help scientists become more efficient in their research computing.

Taken these lessons together I think that the reality of the situation of open source and geographic education is currently rather mixed. At the undergraduate level the impact has been much more limited than I would have originally believed, due mainly to the institutional factors raised above. The situation is more evolved at the graduate level. Here I've seen several instances where access to the source code in PySAL has enabled a motivated graduate student to gain a deeper understanding of a particular spatial analytical method. In hindsight, this mixed success may also suggest that a certain level of training and education may be required before the benefits of open source software can be experienced by students. I am optimistic that as the MOOC concept continues to evolve and Software Carpentry increases its outreach, our ability to engage more fully with the undergraduate population will be enhanced.

Lessons for Research

Analogies are often drawn between the logic of open source communities and the basic way science evolves. The notion of peer review is central to both arenas. Moreover, the ability to build on the contributions of others, as in the case of standing on the shoulders of giants, plays a central role in both open source and science. Finally, there are well accepted standards of behavior and norms in both communities. While these analogies have a ring of truth to them on the surface, closer inspection of each reveals subtle but important differences that may suggest the communities are more different than one might expect. Below I discuss a number of lessons the geospatial research community may draw from the open source world.

Open source has also had major impacts on research in GIScience. In the US this is clearly seen in research proposals to federal agencies as increasingly there are requirements that publicly funded projects include data and results management components so that subsequent research projects can replicate and extend funded projects. Having served on review panels for some of these agencies, a clear trend is that open source has been relied upon by many scientists to respond to these requirements. It should be emphasized that open source software offers clear advantages when it comes to replication as there are no longer any "black boxes" that conceal the implementation of a particular method or algorithm [19].

Peer review, while critically important to both science and open source development, works differently in these two communities. In the case of scientific journals, article reviews are typically done in a double-blind fashion to ensure candor from the referees. I've been an editor of a journal for 15 years, and have been impressed by the quality of reviews and the contributions these make towards often improving the original submission. Given that journal refereeing accounts for essentially zero in tenure and promotion cases, the fact that reviews are done at all, not to mention so well, is simply amazing to me. In the open source world by contrast, peer review is entirely open. The code commits a developer makes, the bug reports a user reports, feature requests, documentation contributions, and a host of other activities are all done in public view. This means that the individuals making those contributions are recognized and given credit. This is quite different from the scientists who spends several hours reviewing and, ultimately, improving a manuscript, since her comments only, and not name, are known to the author and wider scientific community. I think there are wonderful opportunities for academic publishing to learn from open source peer review processes.[4]

Another encouraging development can be seen in the evolving nature of the relationship between spatial scientists using open source code in their research and the development of that code. In the initial phase of adoption of open source in GIScience, the number of users of open source code dwarfed the number of developers of that code, and the intersection of users and developers was minimal. This situation is changing as there are important synergies between the two groups reflected in feature requests from users driving the development of the software. In other words, the distinction between user and developer is beginning to blur as users are coming to play more important roles in open source projects. Rather than being seen as end users or consumers, scientists adopting open source code in their research are increasing being viewed as collaborators in the open source development process [17].

This shift in collaboration will have major positive impacts on both the quality of future open source spatial analysis code as well as in the nature of the way geospatial research is conducted. One of the longstanding criticisms of open source code is that it can be "developer-centric" in the sense that only the developers understand and can make use of the code which is otherwise opaque to the end user. By integrating the research scientist into the development process, developers can be sensitized to the needs of the wider user community and improve the "user-centric" nature of the code. With regard to its impact on the practice of geospatial research and science, the open source model increases the likelihood that the scientific questions lead the way forward and the software itself is enhanced or modified to address these questions. This is in contrast to the proprietary world where the core programs themselves are not malleable. In the past this has led to the choice of research question being constrained by the functionality provided by the software.

The black box nature of proprietary spatial analysis software can mean that changes in APIs, data formats, and related design issues can break backwards compatibility yet, due to vendor lock-in, the costs of this breakage are largely borne by the community of users who are faced with the question of upgrading to the new version or finding an alternative. For the vendor, the clear gains are in a new revenue stream related to the upgraded version of the software. In some cases there

[4]A related development is the rise of open access journals and the open science movement. A full discussion of these is beyond the current scope, but can be found in [12].

may be legitimate debates as to whether the changes in the software reflect true enhancements to the software or not, but the impact is the same.[5]

This is not to say that similar changes in an open source project's code base do not happen. They can and they do. However, in our development of PySAL we have paid close attention to backward compatibility as we add new features and we are loath to break things. Moreover, users have access to the source code and can modify it to suite their own needs when there are changes in the code base. In the extreme case, the project could even be forked if development went in directions at odds with the wishes of our end users. Taken together I think that while open source projects have code bases that clearly evolve more rapidly than is the case of proprietary packages, the nature of community norms is such that the negative impacts of these changes are minimized.

To be sure these are all very positive developments. Yet, for the academic engaged in open source software development there are a number of challenges. A chief one regards the academic evaluation and promotion system which places heavy emphasis on scientific publications. Development, maintenance and documentation of an open source spatial analysis package requires a significant investment in one's time and this cuts into time that could go towards writing and submitting journal articles, books and proposals for funding. For a package that becomes widely adopted there is the possibility that scientists who use the package in their own research take care to cite the package, but my hunch is this is done less often than one would hope. There have been positive developments in this regard with journals such as *Journal of Statistical Software* that provide an outlet dedicated to developments in statistical software. It also reflects a shift in attitudes towards scientific software in that it is seen as scholarly work that should come under peer-review. In other words, the code is indeed viewed as text.

One often overlooked challenge that universities pose to open source developers is that these institutions are fairly conservative and slow to adapt to change. This was brought home to me in a very vivid way early on in my experience as an open source developer. As is customary when receiving a grant from a federal agency, I was called into a meeting with the chief technology officer (CTO) of my university when I received funding for the initial development of STARS from the U.S. National Science Foundation. The conversation went something like the following:

CTO: "Has the software that is being developed in this grant been licensed?"
Me: "Yes, it builds on code that I have placed under the GPL."
CTO: "What is the GPL?"

I was amazed that in 2004 the CTO of a major research university had not heard of the GPL, but in hindsight I think it reflects the simple fact that institutional change occurs at a slower pace than technological change.

[5]For an example of this debate see the discussion and comments on the geodatabase thread at http://blogs.esri.com/esri/arcgis/2008/05/30/five-reasons-why-you-should-be-using-the-file-geodatabase/.

It can also be very difficult to secure funding in support of software development. In part this reflects the dominant view that production of analytical tools is not research, but rather something used in research. What gets funded is published research—text matters, code doesn't, and code isn't viewed as text. In this context one strategy we have adopted is to support parts of PySAL development through funding related to particular substantive projects. This requires having an infrastructure for the meta project that can keep the bigger picture in mind, while responding to the requirements of particular funded projects. In one sense the situation is not much different from that faced by most research active academicians attempting to juggle multiple on-going projects together with the next round of proposal writing. PySAL does, however, provide an overarching umbrella that can tie all these pieces together and, at least conceptually, allow one to see how future opportunities might be integrated into a research agenda.

An additional challenge for the use of open source code in geospatial research is that code itself is not enough. There is somewhat of a "build it and they will come" mentality at work in the all too common practice of new analytical methods developed as part of a research effort being made available as source code. In theory, this should allow other researchers to use the code and apply the new methods. In practice, however, there is often a great deal of heterogeneity in the quality of documentation accompanying the code as well as learning curves to install the code and any dependencies which may limit the dissemination and impact of the new methods. In other words, there can be a substantial gap between what is research code supporting a particular article, and production code that supports the use of the software by a wider audience. Here again, the incentive to the original creator of the new method was the creation of the method itself, the code is often a means to that particular ends. Indeed the competitive nature of academic research can result in a reluctance to release code since it may enable competitors to close the gap with the researcher-developer.[6]

There can also be a substantial gap between the amount and quality of testing that research code is subject to compared to the more extensive set of regression and integration tests that are viewed as a necessary part of an open source project. Those tests play a critical role in ensuring that new changes to the software do not introduce errors elsewhere in the code base, and this relies on the ability of the existing code to replicate a set of known results. Reproducibility is also a central pillar of the scientific process, yet it is highly ironic that much of the source code that generates new scientific results is rarely subject to even minimal software quality control measures. Adopting open source practices in the development of scientific research code could do much to improve the situation.

[6] See [3] for arguments as to why this reluctance may be misplaced.

Conclusion

Although the lessons outlined above treated development, education and research separately, this was for the purposes of exposition only. There are clearly strong potential synergies between these activities. At the same time, there are some challenges that can hamper our ability to exploit these synergies. One of our overriding goals in the development of PySAL has been to keep the level of code readability as high as possible, and here we have relied on the clear syntax of the Python language. We have always felt that the code can serve as a powerful source of information for students interested to learn the exact manner in which a spatial analytical method was implemented. While we have by and large kept to this goal, we have encountered tensions along the way. Keeping the code readable has required that we limit the number of third party libraries that PySAL requires. These libraries are often written in lower level languages such as C, C++, or Fortran and can offer substantial speed gains over pure Python implementations. At the same time the lower level code can be more difficult for the newly initiated spatial analyst to decipher. Faced with this trade-off, we have chosen pedagogy over speed.

As the number of open source spatial analysis projects within academia continues to grow, a difference in attitudes towards collaboration in the open source world versus academia is starting to emerge. The attitude of "not invented here" appears to be more prevalent in academia relative to what I have experienced in the broader open source community. In part, this reflects the pressures that open source academicians face in that citation of their work is critical for their own career advancement. This is, however, unfortunate as opportunities for combining these different tool sets through different forms of interoperability are lost.

The notion of a scientific work-flow has gained much attention in the cyberinfrastructure community, however progress in the implementations of architectures to support these work-flows faces a fundamental problem in that there are many areas of spatial analysis where we lack a consensus on the proper sequence of tools, or even choice of an individual tool. Paradoxically, the problem is not one of a scarcity of tools but rather abundance as users face a bewildering array of software packages. However, many of these are closed source which means their black box characteristic has hindered a deeper understanding of the methods enabled by the software. Open source provides a way to shed light on this area and will be critical in facilitating open discussions about methodological work-flows in spatial analysis [2].

Scientific geospatial analysis offers an important vetting framework—code can be evaluated for its scientific soundness through the formal peer review outlet of journals. As mentioned before, this can stand in contrast to the more open peer review process in open source where the comments of the community can reflect a heterogeneous mix of perspectives and levels of expertise. The rise of open source spatial analysis and tools has played a major role in the dissemination of these technologies beyond the halls of academia. As [7] has noted, this has shifted the educational mission from how to train professionals in the use of these technologies towards cultivating a more fundamental understanding of GIScience principles. In the end it is these principles that are paramount; the software and tools can be seen as a means to these ends. But how that software is built can have profound impacts on scientific and educational outcomes.

Acknowledgements A previous version of this paper was presented at the Open Source Geospatial Research and Education Symposium, Yverdon-les-Bains, Switzerland. I thank Ron Dorn, Olivier Ertz, Janet Franklin, and Stephane Joost for constructive feedback on this work. I have also benefited from the comments of the editors and anonymous reviewers. This research was partially supported by NSF Awards OCI-1047917 and SES-1421935.

References

1. Anselin L, Le Gallo J (2004) Panel data spatial econometrics with PySpace. Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, IL
2. Anselin L, Rey SJ (2012) Spatial econometrics in an age of cyberGIScience. Int J Geogr Inf Sci 26:2211–2226
3. Barnes N (2010) Publish your computer code: it is good enough. Nature 467(7317):753–753
4. Brovelli M, Mitasova H, Neteler M, Raghavan V (2012) Free and open source desktop and Web GIS solutions. Appl Geomatics 4(2):65–66
5. Contiuum Analytics (2014) Anaconda scientific python distribution. https://store.continuum.io/cshop/anaconda/
6. Enthought (2014) Enthought canopy https://www.enthought.com/products/canopy/.
7. Goodchild M (2011) Twenty years of progress: GIScience in 2010. J Spat Inf Sci 1:3–20
8. Hu Q, Chen Y, Zhou Y, Xiong C (2009) An overview on open source GIS software with its typical applications. Geomatics World 1:2009–01. http://en.cnki.com.cn/Article_en/CJFDTotal-CHRK200901014.htm
9. Open Geospatial Consortium (2012) Web Processing Service. http://www.refractions.net/expertise/whitepapers/opensourcesurvey/survey-open-source-2007-12.p
10. Ramsey P (2007) The state of Open Source GIS. In: Free and Open Source Software for Geospatial (FOSS4G) conference
11. Rey SJ (2009) Show me the code: Spatial analysis and open source. J Geogr Syst 11:191–207
12. Rey SJ (2014) Open regional science. Ann Reg Sci 52:825–837
13. Rey SJ, Anselin L (2010) PySAL: A Python library of spatial analytical methods. In: Fischer MM, Getis A (eds) Handbook of applied spatial analysis. Springer, Berlin, pp 175–193
14. Rey SJ, Janikas MV (2006) STARS: space-time analysis of regional systems. Geogr Anal 38(1):67–86
15. Steiniger S, Hay G (2009) Free and open source geographic information tools for landscape ecology. Eco Inform 4(4):183–195
16. Steiniger S, Hunter AJ (2013) The 2012 free and open source GIS software map - a guide to facilitate research, development, and adoption. Comput Environ Urban Syst 39:136–150
17. Wang S, Wilkins-Diehr N, Nyerges T (2012) CyberGIS - towards synergistic advancement of cyberinfrastructure and GIScience: a workshop summary. J Spat Inf Sci 4:124–148
18. Wilson G (2006) Software carpentry: getting scientists to write better code by making them more productive. Comput Sci Eng 8(6):66–69
19. Yalta AT, Yalta AY (2010) Should economists use open source software for doing research? Comput Econ 35:371–394