

informit.com

 Keywords
[Advanced Search](#) | [Search Help](#)
[Home](#) [Bookstore](#) [Safari Bookshelf](#) [Articles](#) [Reference Guides](#)
[Login](#) [My Account](#) [View Cart](#)

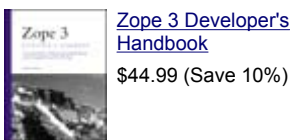
Site Help

[Home](#) > [Articles](#) > Interview with Stephan Richter, Open Source Zope Luminary, Developer, and Authority

Contents

1. CIPS Connections

Related Book


[Zope 3 Developer's Handbook](#)

\$44.99 (Save 10%)

You May Also Like

1. [Financial Advice for the Suddenly Unemployed](#)
By [Edie Milligan](#)
Jul 26, 2002
2. [Consulting: Prerequisites for Success](#)
By [Robert Bacal](#)
Aug 30, 2002
3. [Week 12: Memetic Marketing, Part 1 of 2: Sending Email in Your Web Pages](#)
By [Nick V. Flor](#)
Nov 16, 2001

[See All Related Articles](#)

Search Related Safari Books



Search electronic versions of over 1500 technical books:

Interview with Stephan Richter, Open Source Zope Luminary, Developer, and Authority

By [Stephen Ibaraki](#).Article is provided courtesy of [Sams](#).

Date: Mar 25, 2005.

Stephen Ibaraki interviews Zope guru Stephan Richter about his passion for physics, the evolution of Zope, and why Zope is superior to other systems.

 Other Articles By [Stephen Ibaraki](#).

Stephan Richter has been involved with Zope since 1999 including Zope community activities such as documentation, and organizing the first EuroZope conference, plus working with Zope solution providers, developing add-on products, publishing books on Zope and taking a lead on Zope 3 sub-projects.

Stephan is a Ph.D. student in Physics at Tufts University.

Discussion:

Q: Stephan, as a leader and well-known contributor within the Zope community, we are fortunate to have you with us. Thank you for sharing your expertise with our audience!

A: Thank you very much for the opportunity to spread the word about Zope, specifically Zope 3.

Q: Before we get into your Zope activities, let us explore your passion for physics. Describe your thoughts on numerical analysis of quantum lattice systems and quantum computing? Where do you see it all going in the future: say 5, 10, and 20 years?

A: This is a really exciting time for quantum computing, both theoretically and experimentally. As you may know, it has been proved that we can develop more efficient algorithms using quantum computers. To implement a quantum computer we need many thousand entangled quantum states (known as qbits). The current world record is to have 8 entangled qbits. So you can imagine that there is a lot of work to be done. Also, we currently do not have any cheap way of producing qbit devices. Most experiments are done near absolute zero temperature, a setup not suitable for an average person. Another important experimental challenge is to implement the CNOT (controlled NOT) gate, which has been proved to be a universal gate for quantum computers.

On the theoretical side physicists and mathematicians try to develop new algorithms, such as quantum lattice systems using quantum logic. However, theorists are also dealing with error correction, since not all algorithms in quantum computing have to give you an answer with 100% certainty.

Q: Can you comment on your future with teaching and research?



[Ads by Goooooogle](#)

[Save upto 40% on Books](#)

at Amazon.ca Free Shipping on orders over \$39
www.amazon.ca

[Books - eBay Canada](#)

Fresh daily! Rarities, 1st editions Great prices, from \$1. Affiliate.
www.eBay.ca

[Books Under \\$10](#)

24000 Books .01 Cent to \$9.99-Affil Need it Cheap? Find it on eBay.
search.ebay.com

[Studying For Exams?](#)

An 'A' Student Reveals How You Can Get A's With a Simple 6-Step System
www.infobook50.com

A: Honestly, I am not the research guy. While doing my Ph.D. I want to do a really great project and enjoy it. But I currently have no ambitions to do research later. To me the Ph.D. is simply a necessary milestone to do what I truly love: to teach. Tufts already gives me many opportunities to develop my teaching skills. Besides teaching recitations and labs, I have taught the introductory Physics course in Mechanics during the summer and the classical computing part of a Quantum Computation and Measurement course my adviser gave. I have also been vocal about improving the laboratory components of the introductory courses and hope that I will be part of the group restructuring them during the summer.

After my Ph.D. I hope that I will find a university that will hire me for my scientific teaching skills. I want to explore more methods of teaching. I am particularly interested in the non-conventional use of technology and toys, such as Lego(tm) blocks to demonstrate principles. Maybe this will become my future research area; who knows!

Q: Share with us your experiences with Jason Orendorff, who taught you Python.

A: Wow, you dug deep! I knew how to program Pascal (including the CGI extensions) and a little assembly by the time I entered college. It was all self taught, but good enough for some high-school-level science competitions. Jason and I met the first time in the computer labs of Christian Brothers University. He was a senior working in the system administration group and I was a freshmen. I proudly showed (well, bragged) some code to a friend of mine, when Jason came to the computer, looked over my shoulder and said: This code sucks! Well, you would think that we were both off to bad start, but I challenged his remark and we developed a strong friendship.

Later that year, Jason taught some Python classes to interested students, which I attended. However, Python was so different to me at this point that I had a hard time and I did not know much about the formal terminology in computer science. At the end of the first semester, I joined the system administration group, where I was given tasks to write some Web applications. During this time Jason coached me a lot and I was soon proficient enough in Python and Web application development to help computer science seniors with their final projects.

Over the years Jason not only taught me Python, but also object-oriented programming (with which I had great difficulty at the beginning), Java, Jython and much more. He also got me my first Web-application programming job at iXL(tm), one of those Internet Bubble companies. It was at iXL, when I heard about Zope. Thanks to Jason (and some other great people) I was able to finish college without taking out any loans or financially stressing my parents.

Q: How would you describe Zope and its evolution to various audiences?

A: (a) HTML Developer/Hobbyist - Zope was really never aimed at this target audience. However, as more products were developed, people could install Zope and the products they wanted without much hassle and have blogs, discussion boards, and much more in no time. With the development of high-level CMS systems, such as Plone, CPS, Silva and icoya, the entry level became even lower allowing this group of people to use Zope to its full potential.

In Zope 3, as new as it is, we have not addressed or targeted this crowd at all yet. However, I am hopeful that we will see high-level CMSs being developed in the next years that will make Zope 3 as accessible as its predecessor.

(b) Web Scripter - The scripter is a person that knows how to develop dynamic Web applications using scripting languages, such as PHP, ColdFusion, ASP, JSP and so on. It was the original and is the main target audience for Zope 2. Zope always tries to provide great development facilities to the scripter, such as Zope Page Templates, Python Scripts and SQL Methods. Using these three concepts alone, the scripter can build powerful SQL-based Web applications. Once the scripter becomes more familiar with Zope, other through-the-Web development features are available to him/her, such as ZClasses, which allow object development by clicking some buttons. All this functionality has been even further simplified in CMSs such as Plone.

In Zope 3 we have not addressed this audience at all, since we first wanted to provide a good experience to the audience below. However, there are already several goodies in the repository that will make the scripters heart beat faster. Such features include persistent Python modules, persistent schema-based content types, and SQL expressions for Zope Page Templates (more generally, TALES).

(c) Python Developer - While not one of the target audiences, it felt always natural to develop and extend Zope through Python code. A huge amount of Python products have been developed for Zope 2 this way. A lot of documentation has been developed and newcomers can look at the existing code for help. The main critique Zope 2 gets from Python developers is that its APIs are big and that it is unnecessarily hard to incorporate other 3rd party Python packages.

Zope 3 is all about the Python developer. Its goal was to make it as easy as possible to develop new Python-based extensions and incorporate 3rd party Python packages. And I think we have succeeded. Thanks to the component architecture, any component in Zope 3 can be easily removed and replaced by a custom implementation, allowing one to customize/extend Zope 3 as little or as much as one desires.

Q: What is the situation with Zope worldwide?

A: This is a really difficult question. Before giving an answer, let me make an analogy. Zope could be compared to Linux, where Zope itself is really just the kernel. The add-on products could be compared to tools based on the kernel. The high-level CMSs are almost like distributions that contain the kernel plus many add-on products. Then I, as a Zope 3 core developer, am a Linux kernel hacker that works on the next generation kernel.

Thus, it is very difficult to see the whole picture. The Zope community has grown beyond the sight of a single individual so that my answer will be based on the tidbits of information I have. I am personally most familiar with the

situation in the US and Europe.

Since the US is far more conservative in comparison to Europe, the acceptance of Zope grows slowly, but steadily. Companies, such as Zope Corporation, show that one can compete successfully against the giants, such as IBM Websphere, BEA WebLogic and Vignette. Also, the deployment of high volume sites, like CBS New York and boston.com, give Zope more and more credibility. Plone, one of the CMSs built atop Zope, also increases the visibility of Zope in the corporate environment due to the numerous Plone-based sites and its mentioning in the media. Another indicator of Zope's success is the fact that several Zope-originated technologies, such as the TAL/TALES standard and the Plone CSS style sheet, have been ported and implemented by many other software projects.

In Europe, Zope is much more widely accepted and the biggest competitor is PHP. The French and German government, for example, have both deployed numerous Zope Web applications. Plone, CPS, Silva and icoya took really off and many of the systems integrate well with other software commonly used in Europe, such as SAP. Community life is also much healthier. There are constantly meetings and small conferences all over Europe and Zope solution providers often share projects.

Q: You believe in the real application development capability of Zope. Can you comment on this? List ten ways it is superior to other systems or frameworks!

A: I had to dig on Google to find the context I said this in (which is different than what I thought). In the original context I was referring to the fact that I can develop Web sites using Zope very much the same way I would like a GUI widget-based application. I can separate presentation from logic from data and do so in an object-oriented fashion. If you develop Web applications only using scripting languages, such as PHP or ASP, it is very hard to separate the concerns let alone develop object-oriented code. The end result is that scripted Web pages are unmanageable.

Here is the context I thought of when I read this question. Zope is often considered a Web-only application framework or even worse a Web-CMS-framework. I very much dislike this categorization, since it unnecessarily restricts Zope's capabilities. Zope, in my opinion, is a network-application framework which can be used to develop standalone applications or simply to glue many different services together. With Zope 3 we hope to have created an application framework that is even well-suited for desktop application development.

I don't think it is necessary to list ten ways in which Zope is superior. They would eventually all boil down to the following:

1. Zope is written in Python! The best decision the original creators of Zope ever did was to choose Python as their programming language of choice. Python is ideal for rapid application development, since one does not have to worry about static typing or compiling while still having all the benefits of a powerful, object-oriented, modern programming language that comes with batteries included.
2. Object Publishing - In Zope the path of a URL is interpreted as a path to an object. This process is known as traversal and the rules on how the path is traversed are completely pluggable. Once the object is found, a method is called on the object by a component known as the Object Request Broker (ORB). The ORB knows what method to call - by inspecting the request - and with which arguments to call it. This allows the programmer to think about the application from a functionality point of view instead of the perspective of HTML pages (presentation).
3. Zope Object Database (ZODB) - Zope comes with its own object database that is able to store any Python object. It has many advanced features, such as transaction support. Over the years the ZODB has been proved to be very scalable, especially in combination with load distributing software like the Zope Enterprise Objects (ZEO), which allows it to spread the ZODB over several servers.
4. Free Software - At least in comparison to its commercial counter-parts, Zope being published under the Zope Public License 2.1 - a certified Free Software license - is a real advantage. People do not have to worry about licensing issues, have access to the source code and can reuse code for other projects. Several packages originally developed for Zope are now used by many other Python applications.

Q: Can you provide some good case studies of Zope and illustrate how it is being used?

A: I have not worked on any end-user applications in the last three years, so it is hard for me to come up with particular projects in which Zope was utilized. However, there are several very popular products that run very prominent sites.

Once Zope reached some maturity, the Content Management Framework (CMF) was developed which serves as the foundation for many of the Content Management Systems developed with Zope. As mentioned before, the most prominent one is Plone (www.plone.org), which was used for some NASA and Lufthansa Web sites. CPS (www.cps-project.org) has been developed by Nuxeo, which has implemented many government sites. icoya (www.icoya.com), developed by struktur, has been deployed on several international sites. One CMS I did some work for is Silva (www.infrae.com/products/silva) by Infrae, which was originally developed to be an XML-based system to manage the university catalog of Erasmus University in Rotterdam. There I implemented a tool that could convert Silva XML documents to publishable MS Word documents. This functionality has since been further developed to also allow to convert Word documents to Silva XML files.

But also Zope 3 has some success stories to report. Many Mark Shuttleworth's projects, like the SchoolTool (www.schooltool.org), use Zope 3 as the framework of choice. SchoolTool is meant to be a free school management software. The first component, SchoolBell, a calendaring application will be released in March 2005. Another application sponsored by Mark is LaunchPad, which will be a package management software for the new Ubuntu Linux distribution.

Q: What are the current flaws in Zope and how can they be addressed currently and resolved in the future?

A: As with all other successful software, Zope 2 shows signs of design flaws. Here are some of them:

1. Multiple Inheritance - Python, unlike many other object-oriented languages, allows for multiple inheritance, which is often seen as one of Python's killer features and is thus heavily used in Zope 2. However, as Zope 2 became more popular and more functionality was added, Zope 2's APIs became bloated to a point where it is very hard to implement any object from scratch. You could say that we discovered with Zope 2 the practical limits of multiple inheritance.
2. Internationalization - Zope was originally developed in the US and internationalization was not on the minds of the developers. As Zope became more popular, it quickly spread to Europe and special character had to be available. The problem was patched by using encodings, but it was too late to make the Zope core unicode-aware.
3. Implicit Acquisition - Zope was one of the first frameworks to implement acquisition, which is basically inheritance of object instances. For example, if attribute x is not found on the current instance, it knows how to look up this attribute in the parent object. This process happened automatically in every object that inherited a certain class. Unfortunately, this feature caused some unwanted side effects, which were very hard to debug.
4. Documentation - While there are many books available for Zope these days, people still complain about the lack of documentation. I personally believe it is simply the lack of comprehensive API and independent package documentation that still bothers people. Unfortunately, due to the reasons in (1) it is very hard to develop this level of documentation now.

The good news is that all these issues have been well-addressed in Zope 3! In fact the API bloating problem was the main motivation to start the development of Zope 3. The problem with multiple inheritance is that it creates a tightly-coupled relationship between the various object layers that cannot be decoupled. For Zope 3 we developed the Component Architecture, which is based on objects having very small APIs, which communicate via very well defined interfaces. Any component can be easily replaced by another custom implementation as long as it properly implements the required interface.

Zope 3 was also implemented using only unicode in the core. In fact, the first public developer meeting (sprint) was aimed at this issue. Acquisition, being a very useful concept, has not been abandoned but is now available in explicit form only. One must specifically request an attribute to be acquired. The last issue was solved by changing the development model. Zope 3 makes heavy use of interfaces, which are not only used to define the API, but also to document it. Additionally, many packages contain README.txt files that outline the functionality of the package. The code examples in those files are guaranteed to work, since they also serve as executable tests. Finally, early on I started to write a book that has been updated and expanded while Zope 3 was developed constantly receiving community input about possible improvements, technical inaccuracies and typos.

Q: Detail your work on Zope 3.

A: As one of the core developers I have been involved in many aspects of Zope 3. Here are some of my major contributions:

- Internationalization/Localization: Actually, the way I got involved in the Zope 3 development was by attending the first Zope 3 sprint that discussed the need for internationalization. I have since then implemented all the translation facilities, including message catalogs for various languages and providing locale information based on the ICU XML data files. I have also developed many of the tools to ease the localization process and helped translating Zope 3 to German, my mother tongue.
- Coding Style Guide: Another issue with Zope 2 is that there is no style guideline, which makes the API totally unpredictable. Early on, we decided we would not make this mistake again and developed a comprehensive coding style guide that must be followed by all developers. I enforce the guide by constantly reading through the checkin messages and changed code to detect non-conformances. I have sometimes been referred to as the checkin police by Jim Fulton.
- Documentation: Due to my original involvement in the documentation process for Zope 2, I realized early the need to provide good and complete documentation. Thus I wrote Zope 3's Online Help system and the API documentation tool that are distributed with the Zope 3 package. I am also gardening the Zope 3 development Wiki as much as time allows and authored the Zope 3 Developer's Handbook. Recently documentation has also been made a coding style. No new or restructured package can be checked into the version control without an appropriate README.txt file that demonstrates the functionality of the package.
- Server/Publisher: Early on, I reimplemented the Medusa server code and then built a new HTTP, FTP and XML-RPC atop of this code. Finally I reinterpreted the Publisher code to the new component-based framework.
- Release Management: With the help of Fred Drake and Tim Peters, I have been responsible for the the release of Zope X3.0. This included fixing outstanding bugs, making packages and testing the distribution. I also served more or less officially as the press contact.
- Zope 3 Applications: After about one year of development, the community became wary of waiting and called Zope 3 an academic experiment that would not be usable for real-world applications. To disprove this opinion, I ported the successful ZWiki product to Zope 3 and the criticism died down. Later I also wrote a bug tracker and a message board application (for the book).

Q: Your book, Zope 3 Developers Handbook, is so highly recommended. Share ten special and especially valuable tips from the book.

A: Thanks. I think the expectations are not only so high because of the content, but due to the quality of the material. And this is not thanks to me being a careful writer, but due to the community that criticized the content long before the contract with Sams was signed. Thus I split the tips into 5 tips for other authors and 5 technical Zope 3 tips:

Author Tips:

1. When I was in college, one of my programming professors always presented a code example before discussing any theory. He called it example-driven teaching. I loved his approach and preferred it much over the often abstractly taught mathematics classes. Since then I have worked on applying this method to my writings and thus this book. Every chapter starts with a clear task, then dives right into the code and at the end commonly explains some of greater design and reasoning.
2. I often get very frustrated when books do not contain full code examples and leave you in the dark about the material they do not cover. In my book you will always find the complete code and the actual files are available via the Zope source repository.
3. It is very important to integrate the development community into your projects. The community members will not only report typos and technical inaccuracies, but also tell you about parts of the book they had a difficult time understanding, where explanation is missing and what parts they are interested in. Overall, I think the quality and usefulness of the book at least doubled thanks to the community.
4. Of course, if you share your book with the community before publication, you need to find a publisher that is willing to publish the book under some sort of Open Content license. My book is published under one of the Creative Commons licenses, which permits you to use the book personally, but does not allow unauthorized commercial use and alteration. However, I have written consent from SAMS to keep updating the book online and allow people to make translations. The code developed for the book is all available via the Zope Public License (ZPL) and can thus be freely used and reused, as the ZPL is a free-software certified license.
5. Unfortunately, due to space constraints and quality concerns, not all material made it into the paper version of the book. It is thus important that you have good online resources available, where such material and other updates are available. For example, the glossary, which was excluded from the book, is still available online and is constantly updated to reflect the latest advances.

Zope 3 Tips:

1. Something that you might notice while looking through the book is that there is an awful lot of text about writing tests. And this is intentional. Zope 3 follows the eXtreme Programming (XP) paradigm, which mandates all code to be tested. And we want to project this paradigm onto all people that develop applications based on Zope 3. Writing tests is one of the most important steps during a development cycle and usually takes about 60-70% of the entire programming effort. But it is worth it! Your code will be better!
2. Over time we have noticed that content components should have almost no methods. They only keep data. Any functionality based on this data is better implemented using an adapter (see below).
3. Keep your APIs for a particular functionality as small as possible. You can do this by using adapters to extend an object's functionality. The added benefit is that people will be able to reuse code more frequently and unwanted behavior can easily be replaced without any major operation.
4. If you use Zope 3, never write an HTML form by hand! Zope 3 comes with a great schema and automated form generation mechanism that can be customized as much or as little as one desires. The book demonstrates this feature in several places.
5. When developing Web applications, you always want to associate meta-data with an object, for example a publication date with an article. Many developers would probably store the publication date as part of the real data, which is actually not the right thing to do. Zope 3 has a very effective framework for storing meta-data called Annotations. If an object is declared to be annotatable, then one can associate any type and amount of data with this object using annotations. They are truly your friend and the book covers various aspects of using them.

Q: You are a sprint believer and in learning from each other? Describe the process.

A: Yes! Sprints are 3-5 day programming sessions, when developers get together to work on a particular software. Over the last three years 28 sprints all around the world were conducted to boost the development of Zope 3. The greatest benefit for the participants, however, is the new tricks you learn from other people. During the sprint, programmers always work in pairs; one is driving (i.e. typing) and the other makes sure that no typos are made and that the algorithm is okay. During these sessions you learn all sorts of things; alone working with Jim Fulton has multiplied my knowledge of Emacs many times and I have also learned many new Python techniques that I was not aware of before.

Q: Describe three of your most interesting Zope projects and share five helpful lessons with our audience.

A: There are many interesting projects I have done using Zope. When I was still using Zope 2, I developed an object-relational mapping framework which was later used in a huge Austrian E-Commerce site. For my senior project in college I developed an online testing tool for the Physics department, which featured many science-specific features such as flexible numerical questions, animated sketches/graphs and mathematical input using LaTeX. One of my Zope 3 projects that is used commercially now is the Lucene text-index, where I developed a simple XML-RPC based bridge between Zope 3 and the Java-based Lucene text index.

Lessons learned:

1. I often tend to overdesign my software by abstracting it before understanding the problem domain well. One of the things I learned during the Zope 3 development is to always only implement an API that is sufficient to cover your use cases. You can abstract the software later as more use cases arise.
2. Don't worry about interoperability with other systems, if it is not required. If you keep your APIs small, it will be easy to write adapters and other components that manage compatibility with other systems.
3. As you code, you need to write documentation. In Zope 3 we have managed to combine documentation and testing into one process. All code examples in our documentation files are actually also executable tests. Good documentation is key to the success of your software.
4. Try to avoid reinventing the wheel as much as possible. I agree that it is often much more difficult to adjust your design ideas to existing standards or conform to the limits of existing software, but any software you do not have to write you do not have to maintain either. And contributing to another software project can be very rewarding as well.
5. When I first developed Zope applications, I believed that Zope's object database would not scale and used relational databases. Today I realize that the ZODB is extremely powerful and allows me to develop

applications in a much more natural way. I am not saying that there is not a place for RDBs, but in a lot of Web applications they are simply not necessary, contrary to what most other people try to tell you.

Q: What factors contribute to the success of an Open Source conference?

A: After three successful EuroZope conferences, it all merged to the even more successful EuroPython conference series. I only was involved in the organization of the first EuroZope conference that was held during Linuxtag. I think the following factors apply to any open-source community-based conference.

The most important part is that you, as the main organizer, are excited about the conference and have a lot of time organizing it. The second most important thing is that you have to involve the community in the organization process. It is amazing how much work can be delegated, if you ask the right way. Also, if you organize a conference for the first time, email people personally asking them to give a talk. Bulk E-mails are usually less well received. Also, if you have a small community, jump on the wagon of a larger event, such as the Linuxtag, which can provide you rooms for no or a small fee. Finally, publicize the event well and often. With all the technology at our fingertips this is no problem and frequent noise helps people to remember.

Q: Do you have additional books planned in the medium term?

A: Well, there is currently nothing in the pipeline and I am a bit out of writing energy. But I know that in the not too distant future my fingers will start to itch and I will pick up writing again. I am really hoping to find a topic to write about in Physics or Computer Science. I certainly have plenty of ideas for another one or two Zope 3 books.

Q: Heres a common question, I ask leaders in the Open Source movement. With your knowledge of key trends in Open Source, what ten developments should we be following and why?

A: Wow, I think this is a really tricky question.

1. Fight Software Patents. The longer they are around, the worse the situation will get. It will eventually become so bad that I must be afraid with any line of code that I check in, whether I violate a patent. If we want innovation in the IT industry, we have to get rid of software patents.
2. Raise public awareness of Free Software. I think that free and open software is one of the greatest social movements in recent years. I am not advocating that all software has to be free as in beer, but the freedom part of free can only enhance innovation and security.
3. Support the RIAA (wink). Before I receive death threats, let me make my point. I am totally against illegal use of any intellectual property, be it music, books or software. I am a programmer; I want my rights be respected and be paid for the work I do, if I choose not to give it away. If people think that the recording industry is unfair and makes too much money, then they should send letters to their favorite musicians and urge them to release their music under open content licenses, and not start illegally downloading it. Sams actually told me that the free availability of a book online does not make a difference in their sales.
4. Protect Privacy. With all those new and scary technologies, such as DRM, arriving, Open Source software will play a vital role in protecting the individual's right to privacy. My greatest fear is that the big chip designers might end up developing chips that only work on Windows and other commercial OSs, because Linux does not implement the software-side of the DRM-based hardware. I realize that the chance of this happening is slim (since the community is usually cleverer than any single company), but it is still a concern.
5. Make Open-Source Operating Systems more user friendly. While many Linux distributions are heavily improving on their configuration utilities, many tasks are still too hard. For example, I just got a new Sony VAIO, which comes with a dual-head graphics card. I had to manually edit the xorg.conf file to activate the second head. Try the same task in Windows - it is idiot proof. There the dual-head is automatically detected and you can complete all the configuration using one screen.
6. Collaboration. While choice and competition is healthy, the Open Source community needs to define and abide to standards. I think that the freedesktop.org and OASIS projects are steps in the right direction. We need more of these types of efforts.
7. Opening of hardware specification. This is sort of an anti-trend. Currently it becomes harder and harder to get good specifications for a piece of hardware. It would be good, if the hardware industry would turn around and be more forthcoming, so that Linux users can have good drivers for their hardware.
8. Interoperability. I have the greatest respect for projects like WINE, Samba and OpenOffice-MS-Office Filters. I cannot imagine how painful it must be to reverse-engineer these complex and undocumented frameworks. However they are essential for the continued success of Linux and other free operating systems as they provide migration paths for large companies.
9. Penetration into niche markets. Linux has made it finally into the PDA/cellphone market, which is really good news. If Linux is expanding in this and other similar markets, it will really help the adoption to the Linux desktop and server as well.
10. Government acceptance. As mentioned before, in Europe governments are very supportive of Open Source. I believe that Open Source software will eventually make governments less vulnerable to attacks and lower the IT costs, which will help me, the tax payer. The money is much better spent in scientific research. ;-)

Q: If you were doing this interview, what three questions would you ask of someone in your position and what would be your answers?

A:

Q1: What is most rewarding to an Open Source developer?

A1: It is a great feeling to know that many people use your software.

Q2: What else do you like besides physics and programming?

A2: I love to discover new branches of music. In fact, as I am answering these questions I am

listening to the Toronto Jazz radio station, jazz.fm.

Q3: Where are you originally from?

A3: I grew up in a small village in East Germany near Dresden. I came to the US as an exchange student in 1996.

Q: Now select five topics of your choosing and share your views.

A: Honestly, I have exhausted everything that I wanted to say, because your questions were very comprehensive.

Q: Stephan, we will continue to follow your work in physics, Zope, and the open source community. Thank you again for your time, and consideration in doing this interview.

A: Thank you for this great opportunity to talk about these topics. If people have questions concerning the points I raised, feel free to contact me at: stephan.richter@tufts.edu

Make a New Comment

You must [login](#) in order to post a comment.

[About](#) | [Legal Notice](#) | [Privacy Policy](#) | [Press](#) | [Jobs](#) | [Write For Us](#) | [Contact Us](#) | [Advertise](#) | [Site Map](#)

© 2005 Pearson Education, InformIT. All rights reserved.
800 East 96th Street Indianapolis, Indiana 46240

informit network