**BEFORE THE**
**SECURITIES AND EXCHANGE COMMISSION**
**WASHINGTON, DC**

| | | |
|---|---|---|
| Proposed Rule: | ) | |
| Asset Backed Securities | ) | File No. S7-08-10 |
| Revisions to Regulation AB | ) | |

# COMMENTS OF THE ASSOCIATION FOR COMPETITIVE TECHNOLOGY

Communications with respect to this document should be sent to:

Morgan Reed
Executive Director
ASSOCIATION FOR COMPETITIVE TECHNOLOGY
1401 K St. NW, Suite 502
Washington, DC 20005
Tel: (202) 331-2130

August 2, 2010

## I. SUMMARY AND INTRODUCTION

The Association for Competitive Technology ("ACT") hereby submits these comments in response to the Securities and Exchange Commission's ("SEC's") Request for Comments on its Proposed Rule in File No. S7-08-10, published on May 3, 2010.

ACT's members applaud the SEC's efforts to increase transparency in the Asset-Backed Securities markets. ACT wishes only to comment on the Commission's proposal to require filing of a waterfall computer model in the Python programming language. This proposal however, does raise some concerns to the extent that the Commission's proposal extends beyond the issue of how to improve the quality of financial disclosures, and potentially into the realm of technology architecture.

## II. ABOUT THE ASSOCIATION FOR COMPETITIVE TECHNOLOGY

The Association for Competitive Technology (ACT) is an international advocacy and education organization for developers of software applications and IT services. We represent over 3,000 small and mid-size IT firms throughout the world and advocate for public policies that help our members leverage their intellectual assets to raise capital, create jobs and innovate.

Our community leaders are not political spokesmen—they are engineers. The workings of the SEC are mostly foreign to software developers—but the proposed rule requiring filing a Waterfall Computer Program is a notable exception. ACT draws upon its membership's technical expertise and business concerns to inspire and inform its comments.

ACT was started by a small group of information technology entrepreneurs who

felt their interests were not being represented in government. Today, ACT is still run by entrepreneurs from the industry who intimately understand how the regulatory environment affects business decision-making and strategy.

## III. THE PROPOSED RULE DOES NOT ACHIEVE THE SEC'S STATED GOAL

The SEC's proposed rule is intended to improve disclosure standards in the Asset-Backed Securities ("ABS") market. As part of this effort, the SEC has proposed that ABS issuers file a computer program that models the flow of funds collected on a pool of assets in a transaction to different tranches in an ABS offering.[1] The stated goal is that investors be able to determine "the results of applying the cash flows on the pool assets to the waterfall under different interest rate, prepayment, default and loss-given-default assumptions to determine the likely amount and timing of cash distributions on . . . the ABS."[2] The SEC has identified four requirements for this type of market analysis:

- Loan-level information about the assets, which the SEC is proposing to require issuers to file in a standardized, computer-readable format such as XML;[3]
- A computer program that calculates the contractual cash flows for each tranche of the ABS based on the presumed cash flows of the underlying pool assets – the waterfall model;
- Additional computer models that generate inputs for the computer simulation (such as interest rate, prepayment, loss and loss-given-default models); and
- A computer system that combines the three elements above into a model that allows investors to calculate the value of ABS tranches based on their own assumptions about the behavior of the underlying pool assets combined with the waterfall of the ABS.[4]

---

[1] *Proposed Rules*, § 229.1113, 75 Fed. Reg. 23328, 23429 (May 3, 2010).

[2] *Id.* at 23378.

[3] ACT expresses no opinion on this proposed rule.

[4] *Proposed Rules*, 75 Fed. Reg. at 23378.

The SEC's proposed rules address the first two requirements, and believes that the waterfall computer program could be run "in combination with other internally developed or commercially available vendor interest rate, prepayment, default and loss-given-default models, cash flow engines, or computational services."[5] Under the proposed requirement, the waterfall computer program would have to "provide the user with the ability to programmatically input the user's own assumptions regarding the future performance and cash flows from the pool assets" fleshed out by those internally developed or commercially available cash flow engines or services.[6]

However, without a standard specification and application programming interface ("API") for the waterfall computer program, investors would be unable to take advantage of this purported ability to programmatically input their assumptions, for all practical purposes. Given that the SEC identified 343 unique ABS sponsors during the most recent five-year period,[7] investors would have to manage many different programming interfaces in order to avail themselves of this intended benefit. Some standardization and reuse could occur if sponsors purchase waterfall computer program templates from the same vendor(s) or collaborate on a standard, but the proposed rule is silent on this point.

Most importantly, the proposed rule does nothing to address the primary problem

---

[5] *Id.* at 23379.

[6] *Id.*

[7] *Id.* at 23403. The SEC's statement that the number of unique sponsors should be averaged out over a five year period is an unsupported assumption that gives an unrealistic picture of the costs and burdens of its proposed rules. This statement assumes that a sponsor in one year would not issue ABS securities in another year. This is an untenable position. Without more data, the average number of unique sponsors in each year could more easily be over 300 than the 57 sponsors arrived at by the SEC's simplistic division.

that it was intended to fix: to "convey [waterfall formulas and] information to investors in a form that is both more accurate and more useful to them for data analysis than a textual description of the waterfall."[8] The results of the waterfall under the investor's own assumptions are simply one more series of data points to add to the sample results most issuers currently provide. A Monte Carlo simulation provides a further series of results, but still rely on the assumptions used for input. There is also a not-insignificant risk that a program will produce differing results on various implementations of Python or whatever programming language is mandated.

The SEC asserts that the disclosure of the waterfall formulas and rules in a Python computer program would provide investors with a more comprehensible statement of these formulas and rules than plain text. While this may be true for financial programmers, it is unlikely to be true for the bankers and investors (as well as their lawyers) who must actually agree on the deal. The belief in greater waterfall transparency in code is called into question by a look at a sample Python waterfall program. For example, the sample waterfall program included in the July 8, 2010 Comment to the Proposed Rules by Richard Careaga[9] to demonstrate how a simple waterfall could be programmed cannot be considered legible by non-programmers.

Furthermore, an oft-stated goal of open-source software is that a "thousand eyes" can look at the code to find bugs and anomalies in the implementation, so that even those users who cannot read the code themselves will benefit from those who can.[10] This principle would not likely work in the financial arena, particularly with the

---

[8] *Id.* at 23379.

[9] Available at http://sec.gov/comments/s7-08-10/s70810-41.pdf.

[10] *See* Linus' Law, Wikipedia, *available at* http://en.wikipedia.org/wiki/Linus%27s_law.

strict liability of issuers under the securities laws. If an investor were to find an anomaly in the waterfall rules and formulas, it may see that information as a proprietary advantage rather than reporting the anomaly.

A better solution may be to require disclosure of the waterfall rules and formulas to investors in a format other than narrative description or inclusion in a program. For example, some experts have proposed that the waterfall structure be filed and available to investors in XBRL format.[11] ACT member Steve Smith suggests two other alternatives:

- *An Excel document could contain all of the data as well as the formulae necessary, and most likely would not require the end-user to install anything on their machine*
- *The SEC could simply create a calculator "in the cloud" such that any/all investors could use a single canonical web-based (or web service based) tool*[12]

ACT expresses no opinion on the exact format that disclosure of the waterfall structure of an ABS should take, other than to comment that there are many options for conveying waterfall structure information.

## IV. THE SEC'S PROPOSAL WOULD OPERATE AS A TECHNOLOGY MANDATE.

The SEC's mission is to "to protect investors, maintain fair, orderly, and efficient [securities] markets, and facilitate capital formation."[13] The SEC's mission does not extend to regulation of the technology marketplace, or regulation of the tools used to

---

[11] December 9, 2009 Letter of Philip D. Moyer, Edgar Online, *Extension of Filing Accommodation for Static Pool Information in Filing With Respect to Asset-Backed Securities*, File No. S7-23-09, at 12.

[12] http://stevesmithblog.com/blog/government-mandates-and-programming-languages/

[13] http://www.sec.gov/about/whatwedo.shtml

analyze investments. The proposed rule mandating that ABS issuers file a waterfall computer program written in the Python programming language would interfere with the normal operation of the marketplace for financial analysis tools. Unless there is a compelling arugment requiring a contrary approach, the SEC should not determine winners and losers in the financial technology marketplace.

### A. The SEC Should Not Mandate Use of Any One Programming Language.

Successful technology standards are those created by the market, rather than those imposed on the market. By that measure, while Python is a great programming language with many beneficial aspects and some market success, several other languages continue to be more successful in the marketplace. In several studies of programming language popularity, Java, C, C++, and PHP all rank ahead of Python in marketplace success, with other languages such as Visual Basic, Javascript, and C# ranking closely.[14] There is no indication that Python has achieved greater success in the specialized ABS security analysis market,[15] but even if Python were the de-facto standard today,[16] it would still be ill-advised to mandate its use in waterfall computer programs.

Because markets are fluid and difficult to predict, often an entity will mandate a

---

[14] *See TIOBE Programming Community Index for August 2010*, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html; Programming Language Popularity, http://langpop.com/ (last updated June 17, 2010).

[15] *See Taking the Plunge—Can ABS Survive the Waterfall Computer Program?*, at 4, Kirkland Alert (July 16, 2010) ("Essentially no one uses Python as the language for developing a waterfall program. . . .").

[16] Python may not, in fact, be the best current choice for standard waterfall computer programs as it does not contain a large collection of financial functions. *See* Richard Careaga, *Demonstration of Waterfall Computer Program*, Python for Lawyers, *available at* http://www.pylaw.org/demonstration.html.

technology standard in an attempt to isolate itself and its constituents from the detrimental effects of technology's dynamic nature. An unfortunate side-effect is that the entity is deprived of precisely what we seek from technology: its dynamic ability to find new ways to solve problems. And when governments are urged to adopt technology standards, it is often an attempt to use the purchasing power and influence of a large government entity to create change in technology markets, rather than merely a misguided effort to ensure "best practices." Either motivation is doomed to failure, especially in the area of software. Even when you choose the best option available, that option will not be the best for long. For example, if the SEC does mandate use of the Python programming language for waterfall computer programs, there remains the choice of version 2 or version 3, the currently "intended future of the language." [17] ACT member and python programmer Darrell Hawley says:

> *I'm a Python programmer and I think this is a bad idea. What version of Python? 2.7, 3.1? Keep in mind that Python is not shy about introducing breaking changes into new versions of the language. Of course, what makes Python so much better than every other language? Why not Ruby? Or C#? Or Java? Or Haskell?*
>
> *…In short, the government should not be in the business of determining **how** we accomplish something. They are in the business of determining **what** needs to be done.*

Top-down technology directives are almost always a bad idea, but there are special problems that arise when governments adopt them. Governments are not as nimble as individuals or private organizations at change, nor would we want them to be.

[17] http://wiki.python.org/moin/Python2orPython3.

We want our governments to have fair and stable procedures for making decisions. Indeed, when governments do not follow established procedures, we consider their decisions to be arbitrary and capricious. But mandating particular technology standards goes too far in the opposite direction by casting a fluid innovative area in cement.

The SEC may be able to learn from past mistakes in this area. The US Department of Defense ("DoD") hired the best and brightest to create a genuinely superior programming language in the 1980s called Ada.[18] DoD wanted all software systems to be built in Ada to reduce the cost of development and maintenance, and established Ada as the standard language throughout the four military services.[19] Congress even passed legislation to mandate its use; any exceptions required a "special exemption."[20]

However, at the same time, the "C" programming language was taking off like wildfire, in spite of (or perhaps because of) the fact that it was not nearly as "good" a language. In contrast to Ada, the apparent simplicity of C made it very easy for less-skilled programmers to create systems that "crashed" and were hard to maintain. C became the de facto standard for system programming, and many defense program directors sought waivers to procure systems built in C and other languages rather than Ada. Furthermore, there was very little commercial adoption of Ada, despite the expert consensus of Ada's elegance and superiority as a programming language and a huge built-in market in the DoD. When the DoD finally dropped the Ada mandate in 1997, it

---

[18] http://www.adahome.com/.

[19] DoD DIRECTIVE 3405.1, Computer Programming Language Policy (April 2, 1987), http://sw-eng.falls-church.va.us/3405-1.html#ajpo, *rescinded by* Memorandum of Emmet Paige Jr. of April 29, 1997, http://sw-eng.falls-church.va.us/oasd497.html.

[20] http://archive.adaic.com/pol-hist/policy/mandate.txt.

finally saw the cost savings and reliability it hoped for in creating Ada, "not only because [commercial sector] tools are cheaper and better understood, but because [after 15 years of commercial sector innovation] the specialized military tool (Ada) does not do the job significantly better," as one leading computer science professor put it.[21]

In hindsight, the best thing the government did in its efforts to adopt a standard systems language was to provide a mechanism for waivers and exemptions from the requirement. If the government had not provided for these exceptions, project managers that had different needs would have risked litigation and censure for considering better solutions. The lesson from the Ada experience is that governments should commit to learning and decision-making processes, not technology.

### B. The SEC Should Not Mandate Any Particular Software Licensing Model.

On page 23380, the SEC asks:

*"Should we restrict ourselves to only open source programming languages or allow fully commercial or partly-commercial languages (such as C-Sharp or Java) to be used? If so, what factors should be considered? Are there other requirements we should impose on the possible computer programming languages that are used to satisfy this requirement, other than that such languages be open source and interpreted?*

ACT believes the question itself reveals some of the underlying problems with this direction. By defining available products as "open source" and "commercial", the SEC misses the fact that many open source products are fully commercial, and many commercially built products make code available for free. Open source and proprietary

[21] http://cse.stanford.edu/class/cs201/projects-99-00/critical-systems/military.htm.

products differ in only the way in which intellectual property rights are licensed. The key component of any software license is the set of restrictions on how you copy, modify or redistribute the software.[22]

Both the proprietary and OSS business models provide tremendous benefit to developers and customers. There are open-source and proprietary implementations of almost all the popular programming languages that could be used for a waterfall computer program. While there are certainly those who are philosophically opposed to anything other than free or OSS software, the reality is that we live in a mixed-source world where most users and producers of OSS also use proprietary software for business and personal use. For example, an IBM consultant and long-time Linux advocate, Jason Perlow, explains that in both his professional and personal computer use, there are some things for which there are no OSS functional equivalents.[23] There is nothing to be gained by pretending that there is a war between licensing models, as professionals increasingly recognize that "[t]here is no one-size-fits-all for either software approach."[24] Continued competition and innovation in business and licensing models results in better solutions for those customers and a healthier software industry. The imposition of a single license model such as open source may actually increase costs to customers who don't currently have the training or skills to build in Python. Ironically, the lowest cost option could very well be Excel, a program which uses an open standard document format (XML) that has wide acceptance and compatibility

---

[22] Please see the attached ACT document "A Crash Course on Open Source".

[23] Jason Perlow, *Why I Can Never Be Exclusive to Linux and Open Source on the Desktop*, ZDNet, Sept. 20, 2009, available at http://blogs.zdnet.com/perlow/?p=11153.

[24] Matt Asay, *Free software is dead. Long live open source*, The Open Road, Sept. 29, 2009, available at http://news.cnet.com/8301-13505_3-10361785-16.html.

within the industry. More importantly, the SEC seems to be laboring under the misconception that mandating the use of an open-source implementation of a specific iteration of a programming language provides investor protection. The availability of the interpreter code has no bearing whatsoever on the formulas used to give effect to the waterfall. Whether the particular waterfall-formula interpreting engine is open-source, proprietary, or mixed-source, what matters is that the formulas used for the waterfall be accessible, not the source code of the interpretation engine.

## V. CONCLUSION

For the above reasons, the SEC should not mandate any particular technology solution in its proposed rules to increase disclosure in the Asset-Backed Securities market.

Respectfully Submitted,

Morgan Reed
Executive Director
ASSOCIATION FOR COMPETITIVE TECHNOLOGY
1401 K St. NW, Suite 502
Washington, DC 20005
Tel: (202) 331-2130

Protecting Small Business Innovation

# A Crash Course on Open Source

The concept of "open source" has been both heralded as *the future* and *the destroyer* of the software industry. The reality is far less dramatic. In fact, the term "open source software" refers to software released under a collection of software licenses, ranging from the very basic "public domain" to legally technical "GPL".

**What is the "Source" in "Open Source"**

Source code is the step above the ones and zeros that make up digital information. It is human readable text programmers use to produce programs that can run on a computer. To modify a program, a programmer changes the source code text and then generates a new version of the program from it. Without the source code one generally cannot modify a program beyond options explicitly provided by the original programmer.

**What is "Open Source?"**

People's conceptions of "open source" often differ wildly from what it actually means. What makes source code "open" is the way in which intellectual property rights are licensed. The key component of any software license is the set of restrictions on how you copy, modify or redistribute the software. Officially, open source software (OSS) is software that is made available under the terms of a license approved by the Open Source Initiative (OSI), a corporation dedicated to the promotion of OSS. Currently there are more than 50 approved licenses with a wide range of licensing terms, but all adhere to a general framework:

1. The most important aspect of any piece of software is your "freedom" to change it.
2. There is no distinction between users of an application and programmers who wish to modify the code.
3. Programmers may modify the software at will and redistribute – even sell under some licenses – without having to obtain permission of the original programmer.

In the most popular open source license (the GPL) this freedom goes beyond just a matter of relinquishing control, but goes on to say what must be done with code that is based on the original work. Any new work you do (and distribute) that is tied to the original GPL code is now automatically open source as well: it's a reciprocal license. By requiring you to make public your changes —your creativity and intellectual property —for no charge, the structure of compensation is irrevocably changed.

**How Does Open Source Software Get Developed?**

While the mythology of open source often focuses on an international team of volunteer programmers, the reality is that there are a wide range of development and business models around the open source software. Richard Stallman, the leader of the Free Software Foundation says this freedom is *"… a matter of the program's license; it has nothing to do with how the code was written, or by whom, or by how many people. A program can be free software if it was written by a collaborative group, and it can be free software if it was written by a single person."*

Though some would like to believe otherwise, open source software has not suspended basic economic realities. Since it is impossible to charge a fee based on exclusive rights in the software itself, companies must earn revenue through other means: custom software development, the sale of ancillary products or services such as hardware that runs OSS, the sale of proprietary software that uses or works with OSS, and technical support.

For customers, OSS operating systems like Linux are attractive because they enable Intel-based servers to replace older, more expensive machines. OSS is also attractive because customers can avoid paying license fees for off-the-shelf software, while those doing in-house programming are attracted to the growing body of OSS software as a way to jump-start their development projects.

For developers, OSS is attractive to those who want to build upon existing OSS programs to cut development time and costs; also, developers can create service based businesses to replace revenue they can't earn through software license fees.

Although the quality of OSS is a function of the programmers and the development methods used, there are some noticeable trends in the types of OSS created. Because software companies cannot make a profit from the sale of new, innovative software products through the OSS model, OSS tends to be more imitative of other products on the market. Many have drawn the analogy to the generic drug industry and the role of commoditizing old technology.

**What types of software are NOT Open Source?**

Most software available today is not available under an open source license. However that does not mean that all non-open source software eschews the principles of Open Source. Just as there is a wide spectrum of open source licenses available, there is an even wider set in the world of non-open source software ranging from strict proprietary software to licenses that are open, but haven't been approved by the OSI.

**Proprietary Software**

What is traditionally called 'proprietary software' is software that is held as a trade secret, copyrighted, and/or patented method by its authors. Just like with OSS, the key component of the license is the set of restrictions on how you copy, modify or redistribute the software. For most proprietary programs, your usage is defined with regards to the number of computers on which you may install the program, how many copies you can make, and what modifications programmers may make to it. The ability to view the source code, modify the program or redistribute the program is usually restricted under these licenses. However, there are many proprietary licenses that also enable varying degrees of source code access, modification and redistribution rights.

This licensing model enables the developer to take on the risk of investing heavily in research and development of innovative products that create new markets, because he is able to protect his invention from being usurped by competitors. While companies distributing OSS also bear certain market risks, the trend is to create products where the market is established and the risk associated with an entirely new idea has already been borne by someone else.

**Non-Proprietary, Non-Open Source Software**

There is also a lot of software that is neither strictly proprietary nor open source. Software that is developed under contract for a client often gives the client the rights to view the source code, modify it, and even redistribute it under some circumstances. Sometimes a vendor will share of portions of code to aid with collaboration, but keep the rest proprietary. There are also many open licenses that enable all or virtually all of OSI's requirements but have not been submitted to or approved by the body at this time.

**What License is Best?**

The answer is that there is no “best” license.

From the developer side, it is a question of market and business strategy being pursued:

1. If the developer is spending millions to develop and a market a groundbreaking product, they will likely choose a proprietary license because it is most efficient way to capture the value of what he or she has created. The income from this becomes the fuel for future development.
2. Where there's a small or specialized market, like access for the disabled, proprietary software may be the only way for a developer to recoup the cost of development - the needs are specialized, and don't translate well to a larger user community.
3. If the developer is entering a maturing market where price and customization are prioritized, the open source model becomes more attractive. The market may have plenty of products, and a service contract is the only opportunity.

For the customer, having access to products that may come from a range of licenses insures access to the best technology for the need. Restricting access to technology based on license only serves to prevent the best ideas from reaching the party that matters most - the people using the software.

Ultimately, open source or proprietary software is about choice – first, in the hands of the creators as to how they want their works to be licensed and second, the consumers’ choice as to what they want to buy and how they want to buy it.