Defect Handling in Medium and Large Open Source Projects

A. Güneş Koru and Jeff Tian, *Southern Methodist University*

Open source projects have resulted in numerous high-quality, widely used products,[1] such as the Apache Web server, Mozilla Web browser, and Linux kernel, although they followed a nontraditional development method (see the "Software Quality in the Open Source Context" sidebar). Understanding the defect-handling strategies such projects employ can help us use the publicly accessible defect data from these projects to provide valuable quality-improvement feedback and to better understand the defect characteristics for a wider variety of software products.

We conducted a survey to understand defect handling in selected open source projects and compared the particular approaches taken in different projects. Our respondents consisted of 119 individuals who contributed to 52 medium and large open source projects. We focused on defect handling instead of the broader quality assurance activities other researchers have previously reported.[2] Our results provided quantitative evidence about the current practice of defect handling in an important subset of open source projects.

Understanding the defect-handling approaches that open source projects use can help us better understand the defect characteristics and improve quality for a wider variety of software products. A recent survey looks at the current practice of defect handling in open source projects.

Generic defect handling practices in open source projects

In the open source community, the term *bug* is commonly used to describe software problems and is closely related to the terms *failure*, *fault*, *error*, and *defect*. According to the standard IEEE definitions,[3] a failure occurs when program behavior deviates from user expectations, a fault is an underlying cause within an software program that leads to certain failures, and an error is a missing or incorrect human action that injects certain faults into the product. Failures, faults, and errors are often collectively referred to as defects, and defect *handling* deals with recording, tracking, and

0740-7459/04/$20.00 © 2004 IEEE

Software Quality in the Open Source Context

Open source is a software development method that makes source code available to a large community that participates in development by following flexible processes and communicating via the Internet. Numerous practitioners and researchers have studied open source development, concentrating on its socio-economic implications,[1] business scheme,[2,3] and legal issues.[4] They have also tried to identify the context for the successful open source projects.

Eric Raymond observed that most successful projects are initiated by individuals seeking a solution, or at least more efficient ones, to their own general and everyday problems.[5] Therefore, the problem domains are usually well known among technical communities. This makes it possible to define and specify less volatile, well-understood requirements, as Douglas Schmidt and Adam Porter stressed.[6]

To deal with such problems, usually a few talented developers capable of managing system complexity and overseeing a project and its goals suggest efficient, modular, and extensible architectures and designs. A core developers' group with certain access rights to software assist them. These efforts are complemented by many users and developers communicating over the Internet. Developers' motivations in this context vary from personal use to business, education, fun, relaxation, ideology, and so forth.

Paul Vixie evaluated open source development by considering the activities of traditional software engineering.[7] He expressed that implementation is the developer's favorite activity. However, the other activities such as requirements definition, system level and detailed design, unit and system testing, and support are not carried out in a manner similar to traditional software engineering. He also stressed that quality assurance activities are unorganized, but extensive field testing helps improve quality. He concluded that open source projects can deliver widely used products, but it simply takes longer for them to suffer from the lack of engineering principles.

Similar to other software products, the quality of open source products is related to the number of defects. Products with fewer defects are generally considered to possess higher quality than those with more. In the open source context, defects are located and fixed with the contribution of a large community. The common intuition is that open source product quality benefits from this "many eye balls" aspect accompanied with extensive field testing.[5,7] However, comparative and empirical studies are necessary to validate such arguments, as Robert Glass has expressed.[8]

Today, open source projects are becoming more organized efforts as companies initiate and lead open source projects to gain business advantages. Full-time company employees often participate in managing and developing these projects. The full office suite OpenOffice (www.openoffice.org) and the Java Integrated Development Environment Netbeans (www.netbeans.org) are such projects led by Sun Microsystems. Because of their business concerns, many companies are also interested in understanding the characteristics of open source development and defect handling under such a development environment.

References

1. G. Perkins, "Culture Clash and the Road to World Domination," *IEEE Software*, vol. 16, no. 1 1999, pp. 80–84.
2. E. Leibovitch, "The Business Case for Linux," *IEEE Software*, vol. 16, no. 1, 1999, pp. 40–44.
3. F. Hecker, "Setting Up Shop: The Business of Open-Source Software," *IEEE Software*, vol. 16, no. 1, 1999, pp. 45–51.
4. L. Graham, "Softlaw: Legal Implications of Operating Systems," *IEEE Software*, vol. 16, no. 1, 1999, pp. 20–22.
5. E.S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly and Associates, 1999.
6. D.C. Schmidt and A. Porter, "Leveraging Open-Source Communities to Improve the Quality Performance of Open-Source Software," *Proc. 1st Workshop on Open Source Software Engineering* (ICSE 2001), 2001; http://opensource.ucc.ie/icse2001/papers.htm.
7. P. Vixie, *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, 1999, pp. 91–100.
8. R. Glass, *Facts and Fallacies of Software Engineering*, Addison-Wesley, 2003, pp. 174–177.

resolving these defects.[4] However, consistency of defect recording, completeness of defect records, or even the concept of defect can vary from project to project.

In small-scale open source projects, developers usually make a note of defect fixes in change logs or fix defects without keeping any record. They often designate a mailing list to discuss discovered defects and how to fix them.

For medium- and large-sized projects, or those typically involving more than a few developers for more than a few weeks, more organized defect handling becomes a necessity. In most of the larger projects we are aware of, developers use Bugzilla (www.bugzilla.org) or a variant of it, such as IssueZilla (for example, see www.openoffice.org/project/www/issues/query.cgi), for defect handling. Bugzilla is an online, open source, and extensible defect tracking tool, providing many facilities. These include

- Creating, storing, and querying defect records using its online and graphical user interface
- Describing defects and posting attachments during follow-up actions

A defect's status typically changes several times during its lifetime and might become assigned, resolved, verified, closed, and so on.

- Classifying defects according to their priority, severity, and related components and versions
- Generating reports and graphics from the data stored in the defect database
- Automatically notifying project personnel of changes to a defect's status (for example, from new to assigned) via email to enhance collaboration

Both users and developers can create defect records. Developers possess additional access rights to manipulate previously created defect records. A defect usually gets an unconfirmed status when it's first recorded. A developer can change its status to new after seeing that it's not a false alarm. A defect's status typically changes several times during its lifetime and might become assigned, resolved, verified, closed, and so on. As a result of the follow-up actions, a defect's resolution attribute might become fixed, invalid, won't fix, and so on.

To fix a defect, a developer commits a patch, which includes a set of changed source files, to the concurrent versions system (www.cvshome.org). CVS is a version management system that's widely used in open source projects. In some projects, the developer also writes the corresponding defect number in the commit log and updates the defect record in Bugzilla so that it points to the changed source files.

Differences exist among projects within the generic practices we describe here. These differences stem from decisions about what constitutes a defect, the necessary format and content of defect records, particular processes, and so on. We concentrate on the main differences in defect definition and handling instead of considering minor process variations.

Our survey

We started by identifying the projects of interest for our survey. Our first criterion was that the project had to have resulted in a product that the open source community generally recognized as a success. Our second criterion was that the project used its own publicly accessible defect database and defect-tracking tool. Because some projects, such as Apache or Jakarta, include many subprojects, we decided to include such subprojects as well because each subproject might handle defects somewhat differently. We also consulted individual project Web sites and used our prior knowledge in selecting projects for our survey. This selection process yielded 75 open source projects or subprojects as our survey targets.

With this criteria, we tended to select successful projects of medium to large sizes, which cover an important subset of open source projects with defect-handling approaches and defect data that we can quantitatively evaluate. We can compare the results and findings to that of medium- and large-sized conventional software systems, which is the main object of studies for existing software engineering research.[4] Limiting our focus to projects that meet these criteria also made our survey manageable, since publicizing the survey and handling correspondences might have been infeasible otherwise.

After project selection, we defined our target population as the people involved in development, project management, documentation, testing, and defect fixing, due to their familiarity with our question topics. We did not target the users in our survey because their knowledge about defect handling is usually limited.

We prepared and conducted an extensive survey that included 10 questions directly related to defect handling (see the "Survey Questions" sidebar). The complete survey is available online at www.engr.smu.edu/~tian/data/DHsurvey.html, which also includes additional questions about the general product and process characteristics as well as information related to classifying defects.

We preferred to ask multiple choice questions wherever possible because they are more likely to be easily and consistently answered, and it's easy to statistically analyze the answers. The two common types of multiple choice questions were "choose all that apply" and "choose the best" types. To allow unexpected answers, we often included an Other choice with some extra space to enter text. We found this especially useful because some common patterns that we didn't anticipate became evident from these entries.

To prepare our survey forms and make them available on the Internet, we used an open source tool called Survey (http://opensource.isc.vt.edu/products/survey). After that, we subscribed to a few mailing lists designated for development, defect handling, or quality assurance activities for each chosen project. We sent emails and weekly reminders to the lists to inform the other subscribers about our survey.

Survey Questions

1. Which development project are you contributing to?

2. Which defect database do you use in your project?
   - GNOME (http://bugzilla.gnome.org)
   - KDE (http://bugs.kde.org/)
   - Mozilla (http://bugzilla.mozilla.org)
   - OpenOffice (www.openoffice.org/issues/query.cgi)
   - Apache (http://nagoya.apache.org/bugzilla)
   - LyX (http://bugzilla.lyx.org)
   - Mandrake (https://qa.mandrakesoft.com/cgi-bin/index.cgi)
   - Horde (http://bugs.horde.org)
   - If another, please enter the name of the project that hosts the defect database and its Web address.

3. Which one(s) below describe your participation in the project? (choose all that apply)
   - Owner of the defect database
   - Project manager
   - Developer
   - Tester
   - Defect fixer
   - Documentation
   - Other (please explain)

4. In your project, a defect might be recorded to the defect database because of... (choose all that apply)
   - A change required in the software for any reason
   - A problem/failure occurred
   - An incorrect piece of work that needs to be fixed
   - Any other meaning of defect, which also/better applies to your project (please explain)

5. The defects recorded in your project include the defects in the... (choose all that apply)
   - Source code
   - Design documents
   - Requirement documents
   - Other (please explain)

6. In your project, the defects recorded are... (choose all that apply)
   - Prerelease defects
   - Post-release defects
   - Both prerelease and post-release defects with no distinction
   - Both prerelease and post-release defects but marked as so
   - Any other strategy used about pre/post release issue (please explain)

7. When was the first time the defect database was employed in your project?
   - Inherited from another project or from a previous release
   - Right after the project started
   - Right after the design started
   - Right after coding started
   - Right after prerelease testing started
   - In the post-release stage
   - Other (please explain)

8. After put in operation, how consistent (or disciplined) was the use of the defect database in your project?
   - Very consistent, no defect gets fixed without reporting
   - Almost consistent
   - Not very consistent
   - Not consistent
   - Other consistency degree or approach (please explain)

9. A typical defect record includes several fields of information. In general, how often were the defect records in your project complete?
   - Always complete
   - Almost always complete
   - Sometimes complete
   - Rarely complete
   - Other completion degree or approach (please explain)

Many respondents indicated that they found the questions thought provoking and interesting, but a few complained about the duplicate mailings and weekly reminders they received.

We collected 119 individual responses between 27 September and 29 October 2002. We stopped the survey when the responses slowed down and when we decided that we had enough responses to make statistically valid inferences. We don't have the exact individual response rate because it's impossible to track the total number of people involved in all 75 targeted projects. On the other hand, we expected only a small subset of people directly dealing with defects to respond to our survey.

At the project level, we achieved a response rate of 49.3 percent; of the 75 targeted projects, we received at least one response each from 37 projects. We also received responses from 15 projects we didn't target initially be-

Table 1

The number and percentages of the respondents assuming each role

Respondent role	Number	Percentage
Defect database owner	16	13.45
Project manager	25	21.01
Developer	86	72.27
Tester	69	57.98
Defect fixer	66	55.46
Documentation	30	25.21
Other	4	3.36

cause people working on these projects learned about our survey and responded to it. We included these 15 projects in our survey results because they satisfy our selection criteria.

Consequently, we based our survey results on individual responses from people who worked on the 52 projects. (A complete list of these projects is available at www.engr.smu.edu/~tian/data/DHsurvey_proj.txt.)

Table 1 summarizes the respondent profile based on the answers to Question 3 of our survey. Most respondents identified themselves as developers, followed by testers and defect fixers. Further analyzing possible combinations of (multiple) roles that a respondent assumes gave us the top two choices as developer only, followed by the combined role of developer, tester, and defect fixer. The respondent profile we obtained met our prior expectations, considering the flexible characteristics of open source projects, where developer is a common role and individuals can assume more than one role when necessary. Therefore, this sampling from our target population is likely to be unbiased.

General results

Table 2 summarizes our general results, showing the numbers and percentages of responses that fall into each choice category for Questions 4 through 9. In Questions 4 and 5, the entries made in the Other answer option revealed several common patterns. We treated each detected pattern as a new choice category, which of course reduced the number of entries originally made for the Other answer option.

In response to Question 4, the reasons to record defects, survey respondents selected the

Table 2

General results for Questions 4 through 9

Choice order	Question 4 Reasons for defect reports (choose all that apply)	Question 5 Subjects of defect records (choose all that apply)	Question 6 Pre/post release (choose all that apply)	Question 7 Initial employment (choose one)	Question 8 Consistency (choose one)	Question 9 Completeness (choose one)
1	Any change 100 (84.03%)	Source code 116 (97.48%)	Prerelease 31 (26.05%)	Inherited 31 (26.05%)	Very 14 (11.77%)	Always 2 (1.68%)
2	Problem/failure 116 (97.48%)	Design documents 44 (36.97%)	Post-release 36 (30.25%)	Project start 37 (31.09%)	Almost 55 (46.22%)	Almost always 48 (40.34%)
3	Incorrect work 108 (90.76%)	Requirements documents 32 (26.89%)	Both & no distinction 55 (46.22%)	After design 2 (1.68%)	Not very 37 (31.09%)	Sometimes 59 (49.58%)
4	False positives 6 (5.04%)	User documents 16 (13.45%)	Both & distinguished 61 (51.26%)	After coding 4 (3.36%)	Not 8 (6.72%)	Rarely 6 (5.04%)
5	Feature enhancement 11 (9.24%)	Other 1 (0.84%)	Other —	Testing 4 (3.36%)	Other —	Other —
6	Tasks 2 (1.68%)	—	—	Post-release 15 (12.61%)	—	—
7	Other 1 (0.84%)	—	—	Other —	—	—
No answer	—	1 (0.84%)	—	26 (21.85%)	5 (4.20%)	4 (3.36%)

three choices we suggested (any change, problem or failure, incorrect work) with a high percentage. Using the Other option, respondents also noted they occasionally record false positives (not actually a defect), feature enhancements, and tasks as defects.

In Question 5, source code dominated as the subject of defect records, followed by design documents and requirement documents. Text entries for Other included defects in user documentation such as manuals, guides, and tutorials.

For Question 6, most respondents selected one of the last two choices—indicating they record both prerelease and post-release defects.

Responses to Question 7 about the initial employment time show that the respondents' project usually either employed a defect database and defect-tracking tool right after a project started or inherited them from other projects or previous releases.

For Question 8, more respondents indicated that they recorded defects almost consistently or very consistently than not very consistently or not consistently.

In Question 9, which asked whether every field in each defect record is complete, the vast majority of respondents selected the almost-always-complete and sometimes-complete choices.

We also investigated the relationship between the completeness of defect records and consistency of defect recording because they indicate the discipline exercised in defect handling. In the ideal case, developers and testers consistently record every defect they discover and fix, completing all required fields for each record. Because the answers to these completeness and consistency questions are in ranking order, we calculated the rank (or Spearman) correlation coefficient between the two to be 0.18, excluding the no-answer entries. Therefore, we argue that there's generally a positive correlation between consistency and completeness.

As we mentioned earlier, our survey also included questions about the general process and product characteristics as well as further details about the recorded defects. Answers indicated that open source projects often follow an iterative process or no specific process, the developers use both object-oriented and traditional languages, and various types of project data, such as the project schedule, plan, personnel's education and background, and so on are also available. We can use such information to characterize open source projects and customize various models for them. On the other hand, our survey revealed that there's little detailed information currently collected for analyzing and classifying defect data. Developers and testers usually classify defects according to impact severity because this helps them prioritize defect-fixing efforts. We might need to collect more defect-classification information for various quality-improvement initiatives.

Comparisons across individual projects

We also compared four open source projects with each other:

- KDE (Kool Desktop Environment), an open source desktop suite for Unix workstations (www.kde.org)
- PHP (PHP Hypertext Preprocessor), a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML (www.php.net)
- Mozilla, an open-source Web browser, designed for standards compliance, performance, and portability (www.mozilla.org)
- GNOME, a complete, free, and easy-to-use desktop environment for the user, as well as a powerful application framework for the software developer (www.gnome.org).

Each project had 16, 12, 11, and 10 responses, respectively. The rest of the 52 projects only had between 1 and 6 responses per project, so we didn't include them in this comparison.

Figure 1 includes percentage histograms for these four projects for Questions 4 through 9. The histograms we drew for all 52 projects are at the bottom. To provide a common baseline to compare individual projects, we use the same scale for the y axis for all histograms (0 to 100 percent) and the same scale for the x axis for each question. In determining the scales for the x axes in "choose all that apply" questions (Questions 4, 5, and 6), we used a consistent enumeration of the unique combinations of the selected choices. For Questions 7, 8, and 9, the scales for the x axes included the same answer categories in Table 2 but in reverse order.

Developers and testers usually classify defects according to impact severity because this helps them prioritize defect-fixing efforts.

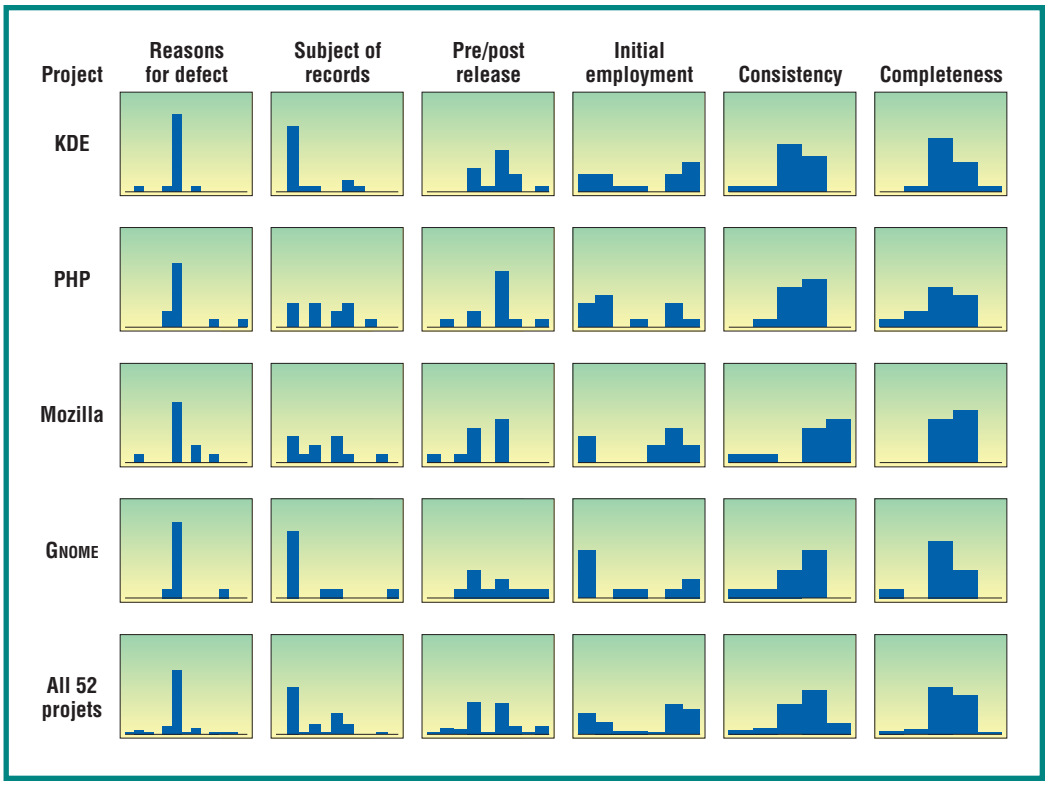

Figure 1. Comparison of projects using percentage histograms.

In our comparison, when we considered reasons for defects, the histograms showed similarities among all four projects in that most respondents selected all of the first three choices provided for Question 4. Regarding defect-record subjects, the histogram shapes formed two groups. In KDE and GNOME, source code stood out as the main artifact against which developers recorded defects. However, in PHP and Mozilla, other artifacts were also associated with substantial proportions of defect records. The histograms revealed a similar picture across the projects for prerelease and post-release defects, suggesting that developers and testers included both. The histograms also looked alike for initial employment time, indicating that initial employment was either after the project's start or via inheritance. Only in the PHP project did some respondents state that they used the tools in the post-release stage. Mozilla scored the highest for defect-recording consistency and completeness, confirming our assertion that the two are generally positively correlated.

Overall, our comparisons reveal many similarities across the compared projects. However, there are also some significant differences in the subjects of defect records and the amount of discipline applied.

Our survey results provide quantitative evidence for the following observations about defect handling in many open source software projects:

- The concept of defect is broad, including failures, faults, changes, new requirements, new functionalities, ideas, and tasks. This is because defect databases and defect-tracking tools are generally considered a communication and collaboration medium that helps developers and testers solve whatever they consider a problem.
- Code is the main subject of defect records, followed by design documents, requirement documents, and user documentation.
- Usually, developers and testers record both prerelease and post-release defects. Almost half the respondents stated that these two types are distinguished and marked so. However, in some projects no distinction exists because development is considered a continuous activity and releases are only perceived as snapshots of an ever-evolving product rather than a finished product.
- Developers and testers record defects fairly consistently and keep fairly complete de-

fect records, although considerable variability exists among different projects. Some projects, such as Mozilla, are more disciplined. On the other hand, in some projects, if developers can immediately correct a defect without further investigation, they might not feel a need to record this incident. Similarly, if developers and testers think a defect record conveys enough information to fix a defect, they might not complete all the fields.

Our findings provide a general characterization of defect handling for an important subset of open source projects and can be used as a starting point for people interested in open source development and related quality assurance and improvement activities. Practitioners and researchers can use the available data and collect additional details about the projects and their defects to build models, perform analyses, and provide valuable feedback to improve their products' quality.

However, due to various limitations for surveys of this type, readers should use caution when interpreting or applying findings from our survey to your project environment. The relatively small sample size and our selection criteria excluded unfinished or discontinued projects or small projects that rarely track their defects formally using defect-tracking tools and databases. Future work covering a larger and more representative sample of the complete spectrum of open source projects could provide more general findings to help us better manage the defect handling process and improve the overall quality for many open source projects.

Acknowledgments

We thank all 119 anonymous respondents of our survey. We also thank *Software*'s anonymous reviewers and editor for their invaluable comments and suggestions.

This work is supported in part by the National Science Foundation grants CCR-9733588 and CCR-0204345, and THECB/ATP grant 003613-0030-2001.

References

1. T. O'Reilly, "Lessons from Open-Source Software Development," *Comm. ACM*, vol. 42, Apr. 1999, pp. 33–37.
2. L. Zhao and S. Elbaum, "Quality Assurance under the Open Source Development Model," *J. Systems and Software*, vol. 66, no. 1, 2003, pp. 65–75.
3. *IEEE Standard Glossary of Software Engineering Terminology*, No. STD 610.12-1990, IEEE, 1990.
4. C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd ed., Prentice Hall, 2003.

About the Authors

A. Güneş Koru is a PhD candidate in the computer science program at Southern Methodist University. His research interests include software quality, measurement, testing, reliability, architecture, open source development, and Web engineering. He has an MS in computer engineering from Dokuz Eylul University, Izmir, Turkey and an MS in software engineering from Southern Methodist University. He is a member of the IEEE and ACM. Contact him at the Dept. of Computer Science and Eng., Southern Methodist Univ., Dallas, TX 75275-0122; gkoru@engr.smu.edu.

Jeff (Jianhui) Tian is an associate professor of computer science and engineering with Southern Methodist University, with joint appointment in the Department of Engineering Management, Information, and Systems. His research interests include software testing, measurement, reliability, safety, complexity, and applications to commercial, Web-based, telecommunication, and embedded software and systems. He has a PhD in computer science from the University of Maryland. He is a member of the IEEE and ACM. Contact him at the Dept. of Computer Science and Eng., Southern Methodist Univ., Dallas, TX 75275-0122; tian@engr.smu.edu.