Conference on Human Factors in Computing Systems - Work-in-Progress Posters (CHI 2008), Florence, Italy, April 2008, pp. 3519-3524.

Toped: Enabling End-User Programmers to Validate Data

Christopher Scaffidi
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA
cscaffid@cs.cmu.edu

Brad Myers
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA
bam@cs.cmu.edu

Mary Shaw
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA
mary.shaw@cs.cmu.edu

CHI 2008, April 5 – April 10, 2008, Florence, Italy
ACM 1-xxxxxxxxxxxxxxxxx.

Abstract
Existing tools require end-user programmers (EUPs) to write regular expressions ("regexps") or even scripts to validate textual inputs, which is slow and error-prone. We present a new technique enabling EUPs to describe data as a series of constrained parts. We incorporate this technique into a prototype tool called Toped, which generates validation code for Excel and web forms. This technique enables EUPs to validate data more quickly and accurately than with existing techniques, finding 90% of invalid inputs in a lab study.

Keywords
End-user programming, spreadsheets, web applications

ACM Classification Keywords
D.2.6 [Programming Environments]: Interactive environments

Introduction
End-user programmers (EUPs) write programs, but not as their primary job function. Rather, they write programs to support accounting, teaching, or another goal unrelated to programming [6]. For example, over 45 million American workers create spreadsheets using Microsoft Excel, while hobbyists, students and other non-expert users create web applications with Microsoft Visual Studio.NET Express or other tools [9].

Often, EUPs' programs will operate on semi-structured strings, such as part numbers and mailing addresses. Unfortunately, though they enable EUPs to create applications, these tools require EUPs to specify regular expressions (regexps) or even scripts to validate data. However, most EUPs do not know how to write regexps [1], nor do they have the time to invest in learning regexp syntax, since programming is not their primary job. So when EUPs cannot create a regexp, they must wait for a more expert person to create a regexp that they can reuse. This dependence on experts is exactly the problem that EUP tools are meant to avoid.

We present and evaluate a programming tool called Toped enabling EUPs to implement validation code. This tool describes strings in the same way that we found EUPs describe data, as a series of constrained parts.

Pilot study

To learn how EUPs describe strings, we asked four administrative assistants to imagine describing two types of data (American mailing addresses and university project numbers) so a foreign undergraduate could find them on a hard drive. (We used this syntax-neutral phrasing in order to avoid biasing participants toward or away from regexps or any other particular notation.)

Participants almost always described data as a hierarchy of named parts, such as describing a mailing address as a street address, city, state, and zip code, with a street address defined as a number, street name, and street type. This structurally resembled a context-free grammar (CFG), down to where sub-parts became so small that participants lacked names for them.

At that point, participants used soft constraints to define sub-parts, such as specifying that the street type usually is "Blvd", "St", or "Circle". By "soft", we mean that participants often used adverbs of frequency such as "usually" or "sometimes", indicating that valid data occasionally violate these constraints. Participants' use of not-always-true constraints stands in stark contrast to validation based on regexps, which classify inputs as valid or invalid, offering no shades of gray.

Finally, participants mentioned the punctuation separating named parts, such as the periods that punctuate parts of our university's project numbers. Such punctuation is common in data encountered on a daily basis by end-user programmers, such as hyphens and slashes in dates, and parentheses around area codes in American phone numbers.

Toped user interface

Based on the pilot study results, Toped describes strings as a sequence of named parts with constraints that can be always or often true (Figure 1). Toped supports a variety of constraints (Table 1), which we initially identified based on the pilot study and later expanded as we used Toped to validate a variety of data. Toped supports an advanced mode, not shown in Figure 1, enabling EUPs to specify conditional constraints that span multiple parts, such as the intricate rules for validating dates. To prevent certain mistakes, the editor disallows some illegal combinations of constraints.

Toped includes a window where EUPs can enter a list of examples, from which Toped infers a boilerplate format that EUPs can review and customize. We have described and evaluated this algorithm in detail elsewhere [8].

To support iterative refinement of formats, a window allows EUPs to enter example strings to test with the format. For each invalid test, Toped shows a list of vio-

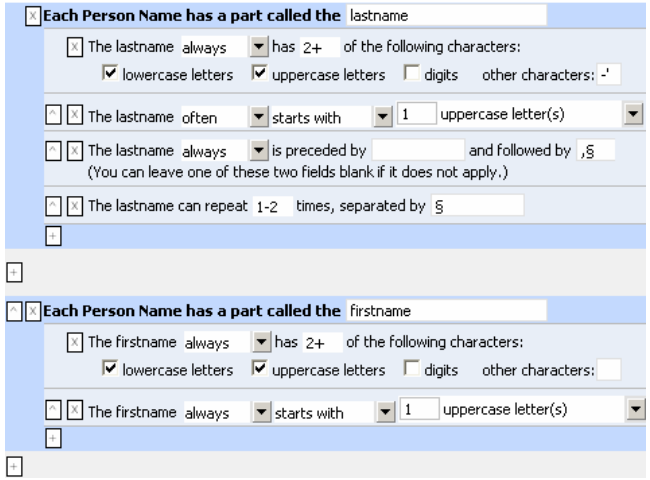

Figure 1: Using Toped to describe person names in "Lastname, Firstname" format, the + buttons add parts/constraints, the x buttons remove parts/ constraints, and the ▲ buttons change the order of parts/ constraints. Each § symbol indicates a space.

Table 1: Constraints that EUPs can apply to parts.

Constraint	Description
Pattern	Specifies the characters in a part and how many of them may appear
Literal	Specifies that the part must equal one of a certain set of options
Numeric	Specifies a numeric inequality or equality
Substring	Specifies that the part starts or ends with a certain literal string, or a number of certain characters
Wrapped	Specifies that the part is preceded and/or followed by a certain string
Optionality	Specifies that the part may be missing
Repeat	Specifies that the part may repeat, with possible separators between repetitions
Reference	Specifies that the part must match another format

lated constraints. To perform the tests, Toped converts the format to a grammar and checks the strings with the grammar. This grammar is a CFG with constraints attached to the productions. Strings are parsed according to the CFG, and nodes of the parse tree are checked against constraints.

When generating the grammar, Toped uses a hierarchy of productions to represent *Reference* and *Optionality* constraints and uses *Wrapped* and *Repeat* constraints to determine placement of separator punctuation and repetition of parts. Toped generates leaf productions based on *Numeric, Pattern* and *Literal* constraints. When a constraint is soft, Toped generates productions that permit the constraint to be violated, but it also attaches the constraint to relevant productions so it can be checked at runtime using parse tree nodes. For algorithm details, refer to [10].

Integration with other tools

Toped's integration with Microsoft Excel and Visual Studio enable EUPs to reuse formats (without modification) for validating spreadsheet and web form data.

To validate inputs in a web form, existing tools require EUPs to specify a regexp in a programming tool such as Microsoft Visual Studio, which automatically generates code for checking inputs against the regexp at runtime. Our Visual Studio plug-in allows EUPs to type in examples of the strings to be validated, rather than a regexp. The plug-in infers a format from the examples and presents it for review and customization in our user interface (Figure 1). Toped stores the resulting format, as well as the generated grammar, on the web server.

The plug-in generates code that checks inputs at runtime against the grammar. This code rejects inputs that violate the CFG or non-soft (mandatory) constraints and displays a message summarizing violated constraints (Figure 2). If an input matches the CFG but violates *soft constraints*, the code shows the message so the end user can correct the input, but the user can *override* the warning. (EUPs can also specify alternate settings, such as always rejecting any input that violates soft constraints.)

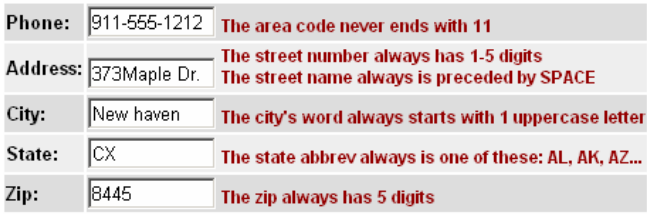

Figure 2. Targeted human-readable error messages are displayed beside textfields or in popups for invalid web form data.

To validate spreadsheets, EUPs highlight some cells, click a button in our toolbar, and select a format file, which can be customized if desired. Alternatively, clicking another button causes Toped to infer a new format from highlighted cells and present it for editing. Based on the

grammar generated from the format, the plug-in validates each cell and flags invalid cells with a red mark and a comment. EUPs can browse through comments using Excel's Reviewing feature.

Related Work

The SWYN tool infers a regexp from example strings and presents it using a visual language for review and editing [1]. Lapis infers a pattern in a specialized textual language that EUPs can edit and use to find outliers that do not match the pattern [5]. Whereas SWYN and regexps require EUPs to describe a string as a character sequence, Lapis allows EUPs to describe a string as a sequence of *parts*, each of which matches a regexp, literal string, or a certain primitive such as Number or Word. Like Lapis, Toped treats data as a sequence of parts but supports various soft constraints on parts, thus better matching the way that EUPs describe data.

Grammex [4] and Apple data detectors [7] describe strings as character sequences with CFGs rather than regexps. No usability studies have been done on these tools, but we expect that the relative complexity of CFGs versus regexps make them less usable than SWYN.

Evaluation

We conducted a between-subjects experiment to assess Toped's usability. As a baseline, we compared Toped to Lapis because Lapis patterns are more expressive than regexps or CFGs.

Using emails and posters, we recruited 7 administrative assistants and 10 graduate students, who were predominantly master's students in information technology and engineering. None had prior experience with Toped or Lapis, but many had some experience with programming or grammars. We paid $10 to each.

We randomly assigned each to a Toped or Lapis group. Each condition had four stages: a background questionnaire, a tutorial for the assigned tool, three validation tasks, and a satisfaction questionnaire.

The tutorial introduced the assigned tool, coaching subjects through a practice task and showing all tool features necessary for later tasks. Subjects could ask questions and work up to 30 minutes on the tutorial.

The validation tasks instructed subjects to use the assigned tool to validate three types of data. Subjects could spend a total of up to 20 minutes on these tasks and could not ask questions. Subjects could refer to the written tutorial as well as an extra reference packet extensively describing features of the assigned tool.

Task details

In Lapis, text appears on the screen's left side, while the pattern editor appears on the right. EUPs highlight example strings, and Lapis infers an editable pattern. Lapis highlights each string in purple if it matches the pattern or yellow if it does not. For comparability, we embedded Toped in a text viewer with the same screen layout and highlighting. Each example string on the left was highlighted in yellow if it violated any constraints or purple otherwise.

Each task presented 25 strings drawn from one spreadsheet column in the EUSES corpus, an existing collection of spreadsheets from the web [3]. Each column also contained at least 25 additional strings that we did not show but instead reserved for testing. For each task, all 50 strings were randomly selected.

The first task used American phone numbers, the second used street addresses (just the street address, not a city

or state or zip), and the third used company names. We selected these types to exercise Toped on data ranging from highly structured to relatively unstructured. The data contained a mixture of valid and invalid strings. For example, most mailing addresses were of the form "1000 EVANS AVE.", but a few were not addresses, such as "12 MILES NORTH OF INDEPENDENCE".

We told subjects that the primary goal was to "find typos" by creating formats that properly highlighted valid strings in purple and invalid strings in yellow. To avoid biasing subjects, we did not use Toped or Lapis keywords in the description of validity. To further clarify the conditions for validity, the task instructions called out six strings for each data type as valid or invalid.

Results

We asked subjects to think aloud when something particularly good or bad occurred in the tool. One Toped subject interpreted these instructions differently than the other subjects, as she spoke aloud about virtually every mouse click. We discarded her data from analysis, leaving 8 subjects assigned to each group.

Tasks completed: In the allotted time, Toped subjects completed an average of 2.79 tasks, while Lapis subjects averaged 1.75, a significant difference (Mann-Whitney, $p<0.01$). Toped subjects were more successful at their primary goal, finding typos. Of the 18 actual invalid strings in the 75 visible strings, Toped subjects found an average of 16.5 invalid strings, compared to 5.75 for Lapis subjects, which was a significant difference (Mann-Whitney, $p<0.01$). In addition, of the 35 typos in the total set of 150 test strings, the completed Toped formats found an average of 31.25 invalid strings, whereas completed Lapis patterns found only 9.5, a significant difference (Mann-Whitney, $p<0.01$).

Accuracy: Finding invalid data is not sufficient alone. Validation should also classify valid data as valid. We evaluated accuracy using F_1, a standard statistic commonly used to evaluate classifiers, with typical F_1 scores in the range 0.7-1.0 [2]. The 23 completed Toped formats had an F_1 of 0.74 on the 75 visible strings and 0.68 on all 150 strings, whereas the 14 completed Lapis patterns had respective scores of 0.51 and 0.46, though these inter-tool differences were not statistically significant (Mann-Whitney, $p<0.05$). (Compared to simply counting classification errors, F_1 more effectively discourages classifiers from simply classifying all items as valid [2], but we also compared simple inter-tool classification error rates and found no significant differences.) Thus, Toped subjects completed more tasks without sacrificing accuracy.

User satisfaction: Subjects generally commented that Toped was easy to use, "interesting" and "a great idea". Most suggested other types of data to validate, such as email addresses, license plate numbers, bank record identifiers, and other application-specific data.

The satisfaction questionnaire asked subjects to rate on 5-point Likert scale how hard the tool was to use, how much they trusted it, how pleasant it was to use, and if they would use it if it was integrated with spreadsheets or word processors. We found that we could combine answers into a moderately robust scale (Cronbach's alpha=0.74). On this scale, subjects reported an average satisfaction of 3.78 with Toped and 3.06 with Lapis, a significant difference (Mann-Whitney, $p=0.02$).

No confounds with background: Subjects had different job categories and varying levels of experience with grammars and programming in general. However, for each tool, we found no statistically significant effects

(Mann-Whitney, $p<0.05$) on task completion, format accuracy, or user satisfaction based on prior programming or grammar experience, or job category.

Comparison to regexps: Though not perfectly comparable, it appears that subjects completed our tasks with Toped more quickly and accurately than subjects completed tasks with regexps in an earlier study conducted during the development of SWYN [1]. For each of 12 data types, that study presented 39 graduate students with 5 strings and a regexp written in one of four notations. Data types were presented in random order, and different subjects received regexps in different notations. Subjects were asked to determine which of the 5 strings matched the data type's regexp.

Average speeds on the last six regexp tasks (after subjects grew accustomed to notations) ranged from 14 to 21 seconds per string, and error rates ranged from 27% to 47%. (No F_1 was reported.) In contrast, Toped subjects not only *checked* strings, but also *constructed a format* at an average of 15 seconds per string (373.8 per task) with a simple classification error rate of 19%.

Conclusion

In this paper, we presented a technique enabling a new class of humans, EUPs, to implement accurate validation code for computer programs. The technique is supported by Toped, a tool that treats strings as a series of constrained parts, resembling the way that EUPs describe data.

Our evaluation only involved three data types, and EUPs might struggle to implement formats for some data. We will develop a repository where EUPs can publish and share formats, enabling us to collect a wider variety of formats and to get feedback from people using formats in real applications.

Acknowledgements

This work was funded in part by the EUSES Consortium via NSF (ITR-0325273) and by NSF under Grants CCF-0438929 and CCF-0613823.

References

[1] Blackwell, A. SWYN: A Visual Representation for Regular Expressions. *Your Wish Is My Command: Programming by Example*, 2001, 245-270.

[2] Chakrabarti, S. *Mining the Web: Discovering Knowledge from Hypertext Data*, 2002.

[3] Fisher II, M., and Rothermel, G. *The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms*. Tech. Rpt. 04-12-03, Univ. of Nebraska, 2004.

[4] Lieberman, H., Nardi, B., and Wright, D. Training Agents to Recognize Text by Example, *Auton. Agents and Multi-Agent Systems*, *4*, 1 (Mar. 2001), 79-92.

[5] Miller, R., and Myers, B. Outlier Finding: Focusing Human Attention on Possible Errors. *Proc. 14th Symp. on User Interface Software and Technology*, 2001, 81-90.

[6] Myers, B., Ko, A., and Burnett, M. Invited Research Overview: End-User Programming. *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 2006, pp. 75-80.

[7] Nardi, B., Miller, J., and Wright, D. Collaborative, Programmable Intelligent Agents. *Comm. ACM*, *41*, 3 (Mar. 1998), 96-104

[8] Scaffidi, C. Unsupervised Inference of Data Formats in Human-Readable Notation. *Proc. 9th Intl. Conf. Enterprise Integration Systems – HCI Volume*, 2007, 236-241.

[9] Scaffidi, C., Shaw, M., and Myers, B. Estimating the Numbers of End Users and End User Programmers. *Proc. 2005 Symp. Visual Lang. and Human-Centric Computing*, 2005, 207-214.

[10] Scaffidi, C., Myers, B., and Shaw, M. *The Topes Format Editor and Parser*. Tech. Rpt. CMU-ISRI-07-104, Carnegie Mellon Univ., 2007.