

# **The 2008 Ruby GUI Survey**

## **Background, Findings and Commentary**

**Alex Fenton ([alex@pressure.to](mailto:alex@pressure.to))**

**10 February 2009**

# 1. Summary of Findings

- The survey received a total of 399 responses; 80% of these completed the whole survey. Respondents were evenly split into those who'd never done GUI programming in Ruby, those who had in the past but weren't doing so now, and those who were currently doing so.
- Most of those doing GUI development were working alone, either on "fun" projects or open-source software. One in three was using GUI libraries to develop in-house company tools; just under 10% were working on commercial GUI software.
- The Ruby GUI "scene" remains fragmented: the survey found at least a dozen separate GUI libraries in current use. The most used toolkits were Shoes (21%), Ruby-GNOME2 (19%) and wxRuby (16%).
- Of users naming a single preferred toolkit, Ruby-GNOME2 and Shoes were chosen by 26%, wxRuby by 17% and RubyCocoa 11%; no other toolkit received more than 10%.
- There are striking differences between Japanese and Euro-American Ruby users. Among Japanese Ruby developers, Ruby-GNOME2 is the preferred toolkit of a majority (56%), whereas among Euro-Americans, it lies third behind Shoes and wxRuby in popularity.
- Preference for one or other of the two leading comprehensive toolkits (GNOME2 and Wx) is not strongly predicted by the general importance attached to features of GUI libraries. This suggests their capabilities and range of potential applications largely overlap.
- The emergence of new Ruby implementations and their associated GUI options has already had an effect on usage. MacRuby/Cocoa and, to a lesser degree, JRuby/Swing are well used and well regarded. MacRuby/Cocoa was the highest rated among all options for how well it met users' GUI development requirements.
- Ruby-Tk received the worst rating for how well it meets users' GUI requirements, with a modal rating of 'poor'. It was the only library for which fewer respondents said they planned to use it in the future than are currently using it. Its continued inclusion in the standard library is unjustified.
- Among those with an opinion, there's a 60/40 split against including any GUI library in the Ruby standard distribution.
- The high degree of fragmentation has not served potential GUI developers well. Almost all see Ruby as a viable GUI programming language, but the immaturity of the toolkits is the commonest reason for not using Ruby for GUI work. The means of redistributing ruby GUI apps to end users is another obstacle.
- The release of Ruby 1.9 addresses some perceived impediments to GUI development in Ruby, such as improved speed, and, more importantly, the availability of system-level threading. There is scope for the reference Ruby implementation to further improve Ruby as a platform for desktop applications, for example, by offering bytecode loading.

## 2. Overview

Section 3 provides some background to the survey, with general information about Ruby as a language for GUI development. Section 4 describes the practical and general objectives of the research. Section 5 describes how the survey was developed and administered. Section 6 gives a profile of the sample of Ruby developers who responded to the survey, and Section 7 describes where and why they are doing GUI development. Section 8 presents findings on GUI developers' criteria for selecting a toolkit. Section 9 reports current patterns of usage and preference of the different toolkits available; Section 10 tries to link stated general requirements to these selections of particular libraries. Section 11 describes respondents' hopes for the future development of Ruby as a language for GUI programming. The last section 12 offers some general conclusions and commentary on the results.

## 3. Background: Ruby and GUI programming

Ruby is an object-oriented programming language. Until recently, its only usable implementation was an open-source interpreter. As with interpreters for comparable languages such as Perl and Python, *ruby* is a command-line tool in the Unix tradition. By default Ruby programs are not interactive; those that are accept text inputs in a terminal, and are limited to using text to provide feedback to users. Ruby programs can be developed in any editor; the standard interpreter *ruby* is not closely integrated with any particular program development environment.

A number of libraries exist to enable Ruby to provide graphical user interaction (GUI). These libraries enable a program to accept input and provide feedback using desktop computing interface elements and conventions, such as buttons, text boxes and windows. The standard distribution of Ruby includes the 'Tk' library to do this; a number of other libraries are provided by third parties. Most of these are wrappers around toolkits written in C or C++, such as FxRuby, wxRuby, ruby-GNOME2 and RubyQt; Shoes is a graphical library which includes some interactive elements, implemented for Ruby alone. More recent alternate implementations of Ruby have their own GUI facilities provided by toolkits associated with the environment, such as Cocoa for MacRuby and Swing for JRuby.

There is a substantial overlap between the libraries in terms of their capabilities; they all enable display of interface elements, on-screen drawing, and handling of user interaction. There are also, however, considerable differences between the libraries in their aims, supported platforms, size, API style, range of widgets, aesthetics, licensing terms and supporting tools and documentation.

On discussion lists, Ruby users often seek advice on selecting a GUI toolkit. Many responses to these requests amount to no more than a statement that "I use XXX toolkit and it works well for me". Such statements, it can be assumed, are both objectively true and well meant. They are little help however in selecting a toolkit, because they say nothing about what specific requirements were met, or what other options were considered. There is no "best" toolkit in abstract terms.

There have also been some attempts at systematic comparison of GUI libraries for Ruby, but these face the difficulties of summarising a large, heterogeneous and evolving field; what follows probably also applies to web framework comparisons. These efforts are limited to describing the superficial

and general characteristics of different libraries in abstract, whereas in fact the usefulness of a library is only really tested by trying to employ it to a specific end. Shortcomings may only become apparent after a substantial amount of purposeful use of a library. Comparative summaries of GUI libraries' features also become invalidated by the release of new library versions.

## 4. Objectives of the Study

The primary purpose of the research was to describe usage, preferences and trends in Ruby GUI programming. A web-based quantitative survey was carried out in order to:

- provide current and potential Ruby GUI programmers with a picture of how well used and well regarded the available options are;
- provide GUI toolkit developers with information on the priorities and requirements of Ruby GUI programmers;
- provide ruby core developers with information on how they might best support the advancement of ruby as an effective and popular language in this field.

The survey was intended firstly to guide potential new GUI programmers, although not to adjudicate which is the "best" toolkit. The survey observed the outcomes of a reasonably large set of individual programmers' work and play with Ruby as a GUI development language. Each had different requirements, prior knowledge and expectations. By looking at the aggregate results in terms of current usage of and preference for GUI libraries, we can say something about how well the toolkits have each been able to meet user requirements overall. Potential new users might then start by evaluating some of the more popular options, on the reasonable assumption that they're popular for a reason.

On the second and third points, developers of toolkits and Ruby interpreters still mostly work on a voluntary basis. For all, the limiting factor for improving the product is generally time. The choice to work on one thing (for example, improving documentation) generally implies delaying another (for example, adding more widgets or features). For them, having a broad summary of the relative priorities of potential users may be a useful guide in choosing what to work on in their respective projects.

### **Ruby GUI Libraries as a case study in open source development**

The survey was also motivated by a broader interest in the limits of the open-source model of development. A broader question is whether the Ruby GUI scene is a 'pathological' case in open-source. Competition and diversity between products offering similar functions are generally held to be an important positive element of open-source development. The effort wasted in developing competing, overlapping products is offset over time by exchange of ideas and gradual consolidation around technically better options.

Whilst by various metrics, Ruby's popularity has greatly increased in recent years, it remains a small language relative to C, C++ and Java. It is an open question whether having a large number of libraries seemingly offering similar features is an optimal outcome.

For several reasons, developing a GUI library for Ruby is particularly time-consuming: the large number of classes and methods often involved; the need to employ lower-level compiled languages; the salience of cross-platform variation; the difficulty of automated testing; highly variable paths through code; and the complexity of reconciling Ruby's GC-based memory management with that of the base language (often C or C++) in long-running applications. Most or all of the GUI toolkits are projects with only a small number of active developers, in some cases, only one. As projects, therefore, they're of a scale where sustainability and development progress are at continued risk, rather than where forking and reconciliation bring long-term improvements.

This report seeks to assess whether the current state is a desirable outcome in two ways. Firstly, it asks users directly about their views of the toolkits and of GUI development in Ruby overall. It hopes to assess whether end users value the diversity, or whether they find the existing options relatively immature – implying that more directed efforts on a smaller number of libraries would have been more in their interests.

Secondly, it considers whether the range of libraries available offers a rational set of choices. It could be that there are important differences in features between the libraries, and the diversity arises from the different toolkits occupying different niches. Users are making rational selections of tools which best meet their requirements. The report attempts to assess this by looking at the links between programmers' stated requirements and their preferences for particular libraries. If these are strong, this would suggest important differentiation among the libraries; if weak, it would suggest that the options are not that different, or that the costs to users of properly evaluating options are too high.

## 5. The Survey Method

Given the intent to provide a coherent picture of overall usage and trends, an online survey was set up to solicit the responses of Ruby programmers. A draft survey was designed using topics and options drawing on previous online discussions on newsgroups and mailing lists. This draft survey was circulated by email to individuals identified as being involved in development on various toolkits; Ruby-GNOME2, Shoes, FxRuby, RubyQt, JRuby/Swing; the author is the lead developer of wxRuby. Several responses were received and the survey modified accordingly.

A record of the survey instrument is available separately. It consisted of 28 items, mostly multiple choice. Respondents could supply their own alternatives where appropriate, and overarching free text comments were invited at the beginning and end of the survey. Questions that invited respondents to rate each of a series of alternatives (such as rating how well different toolkits met their requirements) were presented with the rateable items in random order. The survey was hosted on SurveyMonkey, and responses were submitted between 19 November and 3 December 2008.

An open invitation to complete the survey was posted on the *comp.lang.ruby* newsgroup, which is mirrored to the main English-language Ruby mailing list, *ruby-talk*, and to web-based forums. It was also posted to *Ruby Flow*, a user-driven news syndicator. An email was sent to the same list of toolkit developers, who were invited to forward the survey invitation to mailing lists dedicated to their toolkit. A translated copy of the invitation was posted independently to the main Japanese-language

mailing list. The invitation specifically invited responses from those who were not using Ruby for GUI development, and a short form of the survey could be completed by those who had no experience of GUI development.

## 6. Characteristics of the Sample

An inherent difficulty of online research of this sort is that the size and characteristics of the population from which the sample is drawn is unknown. Therefore, it's impossible to assess exactly how representative the sample is of, say, "all Ruby programmers" – even if that category could be defined clearly enough to be usable. So we must rely on looking at the characteristics of the sample, and seeing if it suggests any likely source of bias in the results. The survey appears to have succeeded in getting responses from programmers with a range of Ruby and GUI experience, natural language background and working environment. Over the two weeks, a total of 399 people undertook the survey<sup>1</sup>. Of these, 319 (80%) completed all the questions.

### Spoken Language

Users were invited to state their first language. English was unsurprisingly the largest category, but nearly as many Japanese Ruby programmers completed the survey:


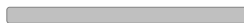











	<i>n</i>	%	
English	108	35.0	
Japanese	98	31.7	
German	20	6.5	
French	19	6.1	
Polish	8	2.6	
Spanish	8	2.6	
Dutch	6	1.9	
Italian	5	1.6	
Russian	5	1.6	
Chinese	4	1.3	
Portuguese	4	1.3	
Danish	4	1.3	
All others	20	6.5	
N/A	90		
All respondents	399	100.0	

Table 1: "Which of the following languages is your first language?"

The survey attracted both veterans with substantial experience with Ruby, and those newer to the language. The number with less than three years experience reflects Ruby's recent growth in popularity.

<sup>1</sup> SurveyMonkey reports 400 responses, but the data file contained only 399 rows. Most tables in the report are based on the cleaned data file, so there are small discrepancies with the totals calculated by SurveyMonkey.







	<i>n</i>	%	
less than 6 months	27	6.8	
6 months - 1 year	51	12.8	
1 year - 2 years	90	22.6	
2 years - 3 years	87	21.9	
3 years - 5 years	65	16.3	
more than 5 years	78	19.6	
N/A	1		
All respondents	399	100	

Table 2 : "How long have you been programming in Ruby?"

## GUI usage

A third of respondents were currently using Ruby for GUI development; a slightly smaller number were not, but had some experience in the past. The remainder had never used Ruby for GUI programming, and so completed the short form of the survey only.




	<i>n</i>	%	
Yes, currently	135	33.8	
Yes, but not now	115	28.8	
No, never	149	37.3	
All respondents	399	100.0	

Table 3 : "Have you ever used Ruby for GUI programming? Do you currently use Ruby for GUI programming?"

As might be expected, those with longer experience in the language generally were more likely to have some experience of GUI programming in Ruby. Those with GUI experience were in a minority among those with less than year's experience; more than 4 in 5 of those with over five years knowledge of Ruby had at least tried GUI development.

The range of uses to which Ruby was being put by respondents ranged widely. 70% of all respondents were using Ruby for web development, 66% for 'systems tools'; write-in answers included "UDP server", "games", "testing", "exploratory programming", "accounting", "data munging", "security tools", "windows automation", "music", "middleware" and "everyday chores".

## 7. Context of Ruby GUI Programming

Before turning to which GUI libraries developers are using, it's worth looking at where and for what purposes they're using them. The survey requested several pieces of information about development environments, tools as well as the social and economic context of GUI development. The table below shows the different situations in which GUI programmers were using Ruby.







	<i>n</i>	<i>%</i>	
Alone, for fun, interest or curiosity	185	78.1	
Alone, on free/OS software	89	37.6	
In a company, on in-house tools	71	30.0	
In a company, on free/OS software	27	11.4	
In a company, on commercial software	22	9.3	
Alone, on commercial software	21	8.9	
All Ruby GUI developers	237	100.0	

Table 4 : "In what situations have you used Ruby GUI toolkits?" (more than one response permitted)

One conclusion that can be drawn from the above is that Ruby is not yet being widely used to develop commercial desktop applications – but it probably didn't require a survey to conclude that. It would be wrong to be pessimistic about the large numbers using Ruby for "fun". Firstly, these experimental uses may well eventually contribute to "real", usable ideas; secondly, it's likely a reflection of Ruby's inherent appeal as a language that it's something to be played with – it might be harder to conceive of similar numbers using C++ to do GUI programming for fun.

As noted in the introduction, Ruby originates in the \*nix environment; it has been argued, quite persuasively, that ruby's support for Microsoft Windows has been less robust. Nonetheless, Windows is the operating system upon which the largest number of Ruby GUI developers expect their applications to run. Linux is a target platform for just over half the respondents; just over a third are writing GUI apps for Apple's OS X.






	<i>n</i>	<i>%</i>	
Windows 2000/XP/Vista	165	71.4	
Linux	129	55.8	
Mac OS X	84	36.4	
Other (please specify)	10	4.3	
Mobile / embedded	6	2.6	
All GUI developers specifying platforms	231	100.0	

Table 5 : "Which platforms do you develop Ruby GUI applications FOR?" (more than one response permitted)

Respondents were also asked to name which operating systems they themselves used for development. Here, Linux was the most commonly named (60%), with Windows in second place (52%). A similar proportion are developing on OS X (35%) as are using it. The total reaches more than 100% because many users are working on more than one platform.

The reference C-based ruby implementation of Ruby 1.8 remains the dominant platform for development, being used by 97% of respondents. The use of other implementations is shown below, with the development version of ruby, Ruby 1.9, being the most popular.








	<i>n</i>	%	
Ruby 1.9	54	22.9	
JRuby	47	19.9	
MacRuby	20	8.5	
Rubinius	6	2.5	
IronRuby	1	0.4	
All responses	231	100.0	
(Ruby 1.8)	228	96.6	

Table 6 : "Which Ruby versions / types do you work with?" (more than one response permitted; bar not shown for Ruby 1.8)

The survey did not ask in detail about editors and tools, as these have separately been the subject of a recent survey<sup>2</sup>. Of specific interest, however, is whether GUI programmers make use of Rapid Application Development (RAD) tools, which enable visual design of user interfaces, as opposed to describing an interface solely in code. Pertinent examples mentioned by respondents include QT Designer, XCode (for MacRuby/Cocoa), NetBeans (Swing), Glade (GTK) and DialogBlocks (for WxWidgets). Overall 25% of Ruby GUI programmers used such tools. Others may not do so because they find the tools inadequately integrated with Ruby, or because in general they do not find such tools useful; this wasn't asked in the survey.

A noteworthy finding was the proportion of respondents who had experience of GUI programming in other languages; 283 (71% of the whole sample) had used one or more other languages for GUI development. Among them, the most familiar languages were C/C++ (known to 57% of those with experience in other languages); Java (55%); VB (36%); C#/.net (34%) and Delphi (22%). Users were less likely to report GUI experience with other scripting languages more similar in design to Ruby, such as Perl, Python, Tcl or Lua; the most common among these was Python (16%).

## 8. Users' Requirements of a GUI Library

Respondents were asked to rate a set of seventeen general features and characteristics of libraries for their importance in selecting a GUI toolkit. These questions about features in the abstract were deliberately posed before any specific toolkits or libraries had been mentioned, partly to avoid any distortion as a result of *post hoc* justification of a particular choice.

The characteristics which respondents rated covered both core technical features of the libraries themselves, such as platforms and internationalisation, and characteristics more related to their organisation as projects, such as documentation and support. The table below shows the ratings given to the requirements. It shows both the proportion of respondents who rated the item as either "very important" or "important", and an average rating given to each option by scoring "very important" as 5, "important" as 4, and so on, down to "not at all important" as 1.

<sup>2</sup> <http://www.tbray.org/ongoing/When/200x/2007/11/26/Ruby-Tool-Survey>

	avg	% "important" / "very important"	
Ease of distributing applications	4.2	83.7	
Web-based documentation	4.2	83.3	
Availability for relevant platforms	4.3	79.8	
Maturity / stability	3.9	71.3	
Appearance / aesthetics	4.0	70.8	
Licence compatible with open source use	4.0	70.5	
API programming style	3.9	68.6	
Ease of installation	3.8	66.5	
Range of features / widgets	3.8	66.2	
Community support	3.7	58.8	
Speed / performance	3.5	54.1	
Internationalisation support	3.5	52.5	
Licence compatible with commercial use	3.2	41.2	
Accessibility features	2.8	25.9	
Availability of extra tools	2.7	23.2	
Familiarity of toolkit other languages	2.5	17.2	
Paper-based documentation	2.4	16.7	

Table 7: "Different Ruby GUI toolkits are sometimes said to have particular strengths and weaknesses. Please rate the aspects below in terms of their importance to you in choosing which GUI toolkits to use in Ruby." (scale: "Very important" to "Not at all important")

There are perhaps few surprises here, although the topmost item points to a particular problem that Ruby, in common with other scripting languages, confronts: the need for an interpreter to be present to run code. This presents problems if the application is to be distributed to ordinary, non-technical end-users, for whom the process of installing an interpreter, the GUI libraries, other dependencies and the application code itself, and then running it from the command-line may well be prohibitively complex. Whilst tools exist to ameliorate this, other comments and answers suggest this has not been overcome, particularly for commercial applications where the application code may need to be obscured.

It's worth noting also that some of these requirements are more or less absolute – that is, if needed, they either are or are not satisfied. Examples include licensing compatible with commercial development and internationalisation support. If needed, they are very important, if not, they are of little importance. In contrast, aspects such as maturity, API style and aesthetics are typically a matter of degree and subjective judgement, and the distribution of responses reflected this.

## 9. Prevalence of Different Libraries

A central aim of the survey was to establish the degree to which the different libraries are actually being used, and whether a clear favourite has emerged. Whilst this won't establish the "best" toolkit, the degree to which longstanding libraries have been adopted reflects the degree to which developers have found them fitted to a range of GUI development scenarios.

For the most part, the options offered by the survey covered all the libraries in use by more than one or two people. The main omission was VisualuRuby, which uses the Windows API to provide a Windows-only GUI library. This project is little known or discussed in the English-speaking Ruby world, but is actively maintained and was cited by around twenty developers, all but one of them Japanese.

	<i>using now</i>		<i>in future</i>		
	<i>n</i>	<i>%</i>	<i>n</i>	<i>%</i>	
Shoes	45	23.8	70	37.0	
Ruby-GNOME2 / GTK	41	21.7	51	27.0	
wxRuby	34	18.0	48	25.4	
Ruby-Tk	26	13.8	19	10.1	
Ruby Cocoa / MacRuby	23	12.2	51	27.0	
QtRuby	15	7.9	30	15.9	
JRuby + Swing	14	7.4	41	21.7	
FxRuby	14	7.4	21	11.1	
JRuby + SWT	1	0.5	25	13.2	
Respondents using any toolkit	189				

Table 8 : "Which of the GUI toolkits do you currently use, and which do you think it's likely you'll use in the future?" (more than one response permitted)

The numbers above clearly demonstrate that GUI library usage in Ruby remains highly fragmented. The most widely used library, Shoes, is currently being used by less than one in four GUI developers. They also suggest that this fragmentation is likely to persist. However, the strong showing of relatively new options, such as JRuby + Swing, MacRuby and Shoes, and the relatively low usage of long-established libraries such as FxRuby and Qt, are an indication that the situation is labile. It's a poor showing for the "standard" Ruby library, Tk, it being the only one where fewer users expect to use it in the future than are using it now.

	<i>n</i>	<i>%</i>	
Ruby-GNOME2 / GTK	42	26.3	
Shoes	41	25.6	
wxRuby	27	16.9	
Ruby Cocoa / MacRuby	17	10.6	
JRuby + SWING	11	6.9	
FxRuby	10	6.3	
QtRuby	8	5.0	
Ruby-Tk	4	2.5	
JRuby + SWT	0	0.0	
All specifying a listed preference	169	100.0	
All naming any other preference	13		

Table 9 : "Which, if any, would you describe as your PREFERRED toolkit?"

Respondents were also asked which of the toolkits, if any, they preferred to use for GUI development. The reason for asking this question was that current usage may be constrained by availability, history or the need to maintain legacy code, but a single preference may better distinguish views on the overall relative merits of the options.

As might be expected, patterns of preference broadly follow patterns of usage, and most of the same comments apply. There was a very strong relationship between mother tongue and preferred toolkit. Among Japanese Ruby developers, Ruby-GNOME2 was the preferred toolkit of an absolute majority; among speakers of European languages, the same toolkit was less popular than Shoes and wxRuby. This is almost certainly a reflection of the language of the lead developers of each of those toolkits: Japanese for GNOME2, English for Shoes and wxRuby. If the lead developers speak the developer's language, this is likely to furnish more documentation and timely community support in that language. This is a nice example of the significance of non-technical factors in toolkit selection.

Users were also asked to rate every toolkit they had experience of as to how well it met their GUI development requirements. Overall, users were not especially impressed with the options to hand; MacRuby/Cocoa was the only one to receive an average rating above "Good". Most other libraries were rated, on average, between "Fair" and "Good", in a similar ranking to overall preference. Below them, FxRuby was rated just below "Fair", and the lowest score was given to Tk, which was most often rated "Poor". This high rating of MacRuby along with anticipated future use suggests that it has a promising outlook in its platform-specific niche.

## 10. Linking Requirements and Toolkit Choice

The survey did not ask respondents to rate each toolkit on how well they met all the different requirements described in section 8.. For a start, this would have been tedious. More importantly, given the time needed to explore and understand a GUI library enough to provide a meaningful rating, it seems that very few programmers would be in a position to answer comparative questions on specific features objectively.

Instead, to test whether the available GUI libraries are differentiated by meeting some requirements better and others worse, statistical models were constructed. These models test how well preferences for particular toolkits can be predicted from the stated abstract requirements of Ruby GUI programmers. For example, does the fact that a respondent attaches a high importance to API style, or a low importance to the range of widgets offered reliably predict that they are more likely to prefer Shoes? If they can – in other words, valuing particular features leads users to select a certain option – this would suggest that the fragmentation found results from the GUI toolkits fitting certain niches better. It should also suggest which requirements lead users to prefer which library.

### Modelling method

To understand the following, first note that they are models of binary choices – for example, "Prefer wxRuby / don't prefer wxRuby". A logistic regression model is constructed in each case; these examine a set of factors and see how much, and in what direction they affect the probability of that

binary choice being true. In this case, we're interested in whether a high or low importance being attached to a particular requirement affects choosing a particular toolkit.

For the statistically minded, the model yields a coefficient – the **strength** of the relationship, and a **significance**. The former shows **how** the factor affects the choice; the latter shows **how likely** it is that the relationship observed is merely down to chance, or is in fact statistically significant. Four models were constructed, for preferring Shoes, GNOME2, wxRuby and MacRuby/Cocoa – the tests are most reliable if a reasonably large proportion of the sample makes each side of the binary choice. The same set of independent variables was used for each model, consisting of all the general requirements, plus variables for whether the user was or was not developing for Windows, Linux and OS X.

### Predicting a preference for Shoes




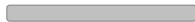

	<i>val</i>	<i>sig</i>	
Extra tools	-3.0	.002	
Community support	+3.7	.005	
Maturity / stability	-2.8	.016	
Aesthetics	+2.5	.055	
Paper documentation	+1.7	.067	
Proportion of variance explained by model	25%		

Table 10 : Factors significant at 90% level in model of Shoes preference

These salient features of the model for Shoes shows that those who prefer it place a high value on the aesthetics, availability of community support and paper documentation relative to other GUI users, and a low value on the maturity and extra tools of GUI toolkits. Note that it doesn't mean that Shoes **has** better community support, or **is** immature and unstable – just that those requirements are valued greatly and little respectively by Shoes users. Falling just outside the 90% significance criterion, but also important, was a negative correlation between Shoes preference and a need for a wide range of widgets, and a positive correlation with valuing the API programming style of a library. This seems consistent with how Shoes is presented: a small library that's fun to use. This makes Shoes a nice example of an open-source library fitting a niche within a broader field.

The other thing to note is the low "proportion of variance explained". This is the amount to which the decision to choose Shoes could be predicted from the factors in the model, against the amount explained by other factors and random variation. This suggests that the reasons that some prefer Shoes was not captured by the abstract technical requirements users were asked to rate. There are various plausible explanations for this. One might be the excellent presentation and promotion of the library, coupled with the fact that developers, having tried it, do not find it necessary to try other options. Another might be that it is not intended to be a "GUI toolkit" in the traditional sense, and so the concerns that dictate its usage are not the same as those that concern other GUI developers.

## Predicting a preference for Ruby-GNOME2





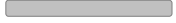
	<i>val</i>	<i>sig</i>	
Developing for Linux	+2.5	.000	
Developing for OS X	-2.2	.001	
Commercial licence	-1.8	.041	
Platform availability	-2.6	.055	
Ease of installation	+2.2	.060	
Proportion of variance explained by model	35%		

Table 11 : Factors significant at 90% level in model of Ruby-GNOME2 preference

Although Ruby-GNOME2 is notionally a cross-platform library, it does not appear that valuing cross-platform development highly inclines people to prefer it. As might be expected, it has a strong base among those developing for Linux. It's most interesting to compare this with wxRuby.

## Predicting a preference for wxRuby








	<i>val</i>	<i>sig</i>	
Developing for Linux	+2.5	.000	
Developing for OS X	-2.2	.001	
Commercial licence	-2.2	.021	
Ease of installation	+2.0	.064	
Platform availability	-2.4	.067	
Extra tools	+1.7	.084	
Accessibility	-2.0	.088	
Proportion of variance explained by model	34%		

Table 12 : Factors significant at 90% level in model of wxRuby preference

What is striking here is the very high degree of similarity in the requirements that predict preferring GNOME2 and those that predict wxRuby. These are two "big" toolkits, in the sense that they offer a comprehensive set of GUI features, and are in common use in Ruby. The model results suggest that they are both meeting similar requirements; rather than holding the niche occupied by Shoes and Cocoa, they are largely overlapping products. Given that wxWidgets is well established for Windows development, it's perhaps surprising this doesn't come us a significant predictor for wxRuby.

## Predicting a preference for MacRuby / Cocoa





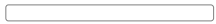
	<i>val</i>	<i>sig</i>	
Developing for OS X	+6.1	.002	
Developing for Windows	-3.3	.015	
Accessibility	+9.3	.043	
Commercial licence	+4.5	.058	
Ease of installation	-5.5	.060	
Proportion of variance explained by model	67%		

Table 13 : Factors significant at 90% level in model of MacRuby / Cocoa preference

The predictors of preference for the fourth most popular toolkit go some way to explaining the preferences for the others. Those who use MacRuby / Cocoa are of course, interested in developing for the OS X platform; the negative correlations between developing for OS X and preference for GNOME2 and wxRuby suggests that Cocoa is already becoming a preferred option for those targeting Apple's platform. Worth noting here is that an interest in developing commercial apps predicts a preference for Cocoa – it may be that, with MacRuby, Apple has begun to provide an attractive platform for commercial desktop application development in Ruby where the other GUI toolkits have not.

## 11. Views of the Future

The last sections of the survey asked respondents about the future of Ruby as a GUI development language: what are the biggest obstacles, and whether things are moving in the right direction. Before looking at those answers, it's useful to look at the reasons that some Ruby programmers are **not** doing GUI work.

### Reasons for not doing GUI development in Ruby

As noted above, around a third of the sample had never used Ruby for GUI development. As these users presumably enjoy using Ruby to some degree, the reasons that they choose not to use it for this type of programming are of particular interest. For some, the reasons are benign – simply lack of time to explore this application of the language: *"am interested, but never took the time to actually do it"*. Other responses are important in understanding Ruby's shortcomings in this area.

	<i>n</i>	%	
No GUI toolkit meets my requirements	59	49.2	
Prefer to develop UIs through web or rich media	36	30.0	
Development tools are better for other languages	29	24.2	
Not interested in GUI programming at all	28	23.3	
No way to protect an application's source code	15	12.5	
Already happy with another language	9	7.5	
Ruby's performance is too slow	7	5.8	
Ruby is the wrong kind of language	2	1.7	
All GUI non-users specifying reasons	120		
Number making additional comment	23		

Table 14 : "Which of the following reasons explain why you don't use Ruby for GUI programming? Please choose as many as are relevant."

The most common reason is not an insurmountable objection to using Ruby for GUI development, but dissatisfaction with the currently available libraries. As one respondent succinctly put it in the comments, *"every GUI library I've tried has sucked"*. Several users suggested that libraries which are ports from other languages are poorly integrated with Ruby's features: *"Too difficult to write the code (writes like java, runs like ruby... where's the upside?)"*.

Whether, when and where web-based client/server applications are equal or preferable to desktop applications is an open debate. Ruby's popularity is in no small part due to its web frameworks such

as Rails, and so it's to be expected that a fair number of non-GUI developers prefer this medium. More important, from the point of view of the GUI library developers, is the number who are put off GUI development by the lack or shortcomings of development tools.

### Obstacles ...

Those with experience of GUI programming in Ruby were similarly asked what they thought most impeded this use of the language. Their answers tell of the real obstacles encountered by those who've tried one or more of the available options:


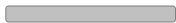




	<i>n</i>	%	
Maturity of toolkits	76	25.8	
Difficulty of distributing applications	66	22.4	
Quality of documentation and learning materials	64	21.7	
Availability of RAD / design tools	43	14.6	
Ruby interpreters' speed / performance	26	8.8	
Prejudice against certain platforms or vendors	20	6.8	
All GUI users specifying an obstacle	295		
Number making additional comment	21		

Table 15 : "Which of the following do you think most impedes the use of Ruby for GUI programming?" (one answer only permitted)

The commonest reason, already suggested by the low average ratings given to most toolkits, is that the toolkits are just not yet sufficiently developed to be comfortable to use. More work will be needed on the libraries themselves, and also their documentation to advance this domain of Ruby development.

A problem specific to Ruby picked up here is the difficulty of distributing applications to potential users. In compiled languages like C and C++, a compiled binary provides a natural way of providing applications to end users. They are immediately usable, fast, reliable, well integrated with the system, and secure against at least casual attempts to discern the source code. There are ways for different ruby implementations to turn interpreted code into a standalone runnable binary that can be distributed – perhaps the best known for the reference interpreter is RubyScript2Exe. However it's clear that the problem is not yet fully addressed.

Numerous write-in comments suggested that some current and potential GUI developers are deterred by the lack of a standardised library for this purpose: *"It would be nice to have a modern, simple but complete GUI toolkit as part of Ruby. It's the one thing all the modern scripting languages lack, except TCL/TK"*. The survey specifically asked whether Ruby ought to include a GUI library as part of the standard distribution. 23% said that it should, 35% said that it shouldn't, but the largest number, 42%, didn't know.

### ... and optimism

To end the results on a cheery note, respondents were asked their opinion of how Ruby was improving as a platform for GUI work. A large majority think that things are improving, even if most of those think that it's not doing so at a blazing pace:







	<i>n</i>	<i>%</i>	
Improving quickly	54	18.0	
Improving slowly	177	59.0	
Staying about the same	65	21.7	
Getting worse	4	1.3	
All GUI users responding	300		

Table 16 : "What direction do you think that the quality of Ruby for GUI development is going in? Please judge this relative to other options that you're aware of."

## 12. Commentary

The preceding has attempted to provide a fairly neutral description of the survey and its context. This last section is more opinionated; it tries to distil what the survey shows about Ruby, GUI development and open source more generally.

Ruby is potentially a great language for GUI programming. Its sophisticated and well-integrated treatment of anonymous functions and closures is well suited to event-driven programming: handlers for UI events can be defined tersely and with access to whatever contextual information is needed. Ruby offers a thorough and flexible object model; GUI libraries tend to utilise deep and complex class relationships to represent the behaviour of UI objects. UI development, with a multitude of paths through code driven by unpredictable user actions tends to mean a lot of iterative development; Ruby, as an interpreted language, means that changes can be tested nearly instantaneously. As discussed further below, Ruby 1.9 offers important improvements in the core library that are relevant to GUI development.

Given all that, it's in a way disappointing that the survey results suggest that Ruby is not yet an ideal platform for GUI development. Many respondents are not using GUI libraries for Ruby because they are not convinced of their merits. Whilst some of those who are doing GUI programming like their tools, others have reservations, some of them serious.

### The costs of fragmentation

*"I think there are enough GUI libraries already. (Just not for my purpose)."*

The survey findings show that the pattern of usage of different libraries is still highly fragmented. The most widely used toolkits are employed by around a quarter of all GUI developers. A lot of them, including the reference Ruby interpreter's standard toolkit, Tk, are used by fewer. Competition and differentiation among are often regarded as characteristics of healthy "markets": potential users are thus able to make rational choices amongst available products, selecting whichever best meets their particular needs. The reality falls rather short of this ideal, for several reasons. Firstly, there's a high cost in time needed to properly assess the available options as to how well they meet requirements. Secondly, it seems that the more comprehensive "big" toolkits are not much differentiated by their features. And, thirdly, the libraries available are falling somewhat short of users' expectations.

Overall, it's probably fair to say that Ruby's GUI toolkits are not blazing ahead. The ports of the "big" cross-platform C/C++ toolkits (GNOME2, Qt, Wx) are at the moment more or less keeping up with

the base libraries, but are not yet really exploring how a really “Rubyish” toolkit might work. The smaller, more lightweight toolkits (Tk, Fox) are stable and comfortable to use, but their inherent aesthetic limitations mean they're not really viable for general-purpose end-user applications. Newer implementations of Ruby (JRuby, MacRuby) may offer new and interesting possibilities, but are limited by the acceptability and availability of their base platforms. And whilst Shoes is probably the most well fitted to Ruby's paradigm, it is not, and has never set out to be, a general purpose desktop application development toolkit.

From one perspective it's a missed opportunity that Shoes was implemented from scratch in C, rather than as a layer atop an existing cross-platform library like QtRuby or wxRuby. These already offered cross-platform widgets and event handling, and lower-level cross-platform drawing primitives capable of doing everything that Shoes does. The attractions of Shoes are clear and confirmed by the survey: an attractive, fun, Ruby-ish API for graphical applications. However providing this on top of an existing library would have had benefits all round: to Shoes (an easier and more extensible, pure-Ruby implementation), to the base library (refinement and testing), and most importantly to end users (being able to drop down to a more comprehensive library when the limits of the Shoes API is reached, rather than being caught in a gilded cage).

From a more authentic perspective – its own – Shoes is successful, and its popularity is confirmed by the survey. The plaintive preceding paragraph in fact stems from the way that the purported benefits of open source have only rarely been realised in the field of Ruby GUI development. To be specific, the exchange of code and ideas between the different Ruby libraries has so far been quite limited. This is largely because, with there being limited resources scattered over many projects, each project's effort has been primarily consumed by dealing with the technicalities of the interface between Ruby and the lower-level functionality being ported. This effort is relatively non-portable, and has been at the expense of more exchangeable work on topics of common concern: the development of API models for Ruby GUI programming, usability, and innovation in interaction design. In this regard, Shoes is a singularly valuable and important experiment.

To return to the quotation above: developers do not want many options, they want good ones. Various reasons have been suggested why Ruby, according to this research, has not yet wholly delivered this for GUI development. The high degree of fragmentation in Ruby GUI options has not, it seems, particularly served the needs of potential GUI developers, but it's not possible to speculate whether that fragmentation will continue, reduce or increase.

### **The importance of Ruby 1.9**

*“Native Threads. That's all I have to say about that.”*

The recent release of a new major version of Ruby, Ruby 1.9, merits comment. Several of the most salient features of this release are likely to be of particular benefit to GUI development in the language. The most remarked change is improved speed relative to Ruby 1.8. As interpreter performance is directly observable by users of a desktop application, increased speed is definitely welcome. The speed improvements are however evolutionary rather than revolutionary, and are less interesting than other developments. For example, Ruby 1.9 offers more sophisticated handling of

string encoding, valuable for applications which work with multilingual text or which need to be localised.

From the point of view of GUI development, the most important single change is the availability of system-level threaded programming in Ruby 1.9. Desktop applications frequently need to execute some long-running task (for example, downloading a file) whilst still allowing user interaction with the GUI. With Ruby 1.8's threads, implemented at the interpreter level, achieving this was frequently difficult; from early experience, Ruby 1.9 is a great improvement.

Ruby 1.9 also steps along the path to a (byte-)compilable system, which potentially can support easier and more secured redistribution of Ruby code as runnable applications to end users. As the survey found, the difficulty of redistributing applications is an obstacle for more than a few GUI developers: "*For proprietary application, wrapping scripts in .exe file (as exerb/rubyscript2exe) is not enough*". At present the default Ruby 1.9 doesn't allow running from compiled instruction sequences; hopefully this survey demonstrates to core Ruby developers the importance of this feature to GUI development and look to implement it fully and enable it upcoming 1.9 releases.

### **Tk, and the Ruby standard library**

Having discussed what might be added to Ruby 1.9, let's consider what ought to be removed. The survey shows that it's well past time that the standard distribution of Ruby dropped Tk. Despite being bundled with the most widespread implementation of Ruby, the survey shows it's not widely used, and is held in very poor regard by GUI programmers. There's no reason that Tk can't continue to exist, like all the other GUI libraries, as an independent project, installable as a gem or by other means. There's also absolutely no reason that the limited resources of the core Ruby development team should be committed to maintaining it in the standard library.

Unlike some recondite but benign parts of Ruby 1.8's standard library (for example *abbrev.rb*) Tk being included has very real negative consequences. It makes porting (for example, to 64 bit systems), and installation more complex, and may lead end users who have no intention of ever using Tk to configure their installation in a way that is otherwise sub-optimal; for example, configuring with *-enable-pthreads* is recommended for Tk, but may otherwise significantly reduce interpreter performance.

Of those with an opinion, a majority do not want any GUI library bundled with the standard distribution of the C-based interpreter. Among those that do want that, there is no consensus as to which library should be included, and most options are anyway technically unattractive for integration into the standard library. In the free-text comments, some respondents indicated that they do want a clear-cut preferred GUI library choice, but it seems better that this should happen through competition between independent projects, and research like this, rather than a choice by the core team. Similar debates have taken place with Python 3.0, and bundling a GUI library has been rejected for similar reasons.

### **Things change quickly**

Lastly, the widespread adoption and preference found for novel options such as Shoes, and, to a lesser degree, MacRuby and JRuby + Swing should be taken as encouraging signs about Ruby and about open source more generally. The Ruby GUI field, as with others, is being constantly refreshed by people starting anew, as well as by seasoned developers experimenting and playing. The open source model is fundamental to enabling this exploration and play, and this, in tandem with Ruby's inherent merits, bodes well for the future of desktop development with Ruby.

## **13. Etc.**

### **Acknowledgements**

Thanks are due to all the Ruby programmers who completed the survey, especially those who did so in a second language. I'm grateful also to those who commented on earlier drafts of the survey, and to those who publicised it in various channels.

### **About the author**

I'm a sociologist, anthropologist, and occasional Ruby programmer. My day job is as Research Associate in the Department of Land Economy at the University of Cambridge, UK, where I do research and teaching on housing, neighbourhood deprivation and urban policy. I designed and developed Weft QDA<sup>3</sup>, a desktop tool for qualitative data analysis in the social sciences, written in Ruby. I'm also the lead developer and maintainer of wxRuby.

---

3 <http://www.pressure.to/qda/>