# The Quail project: A Current Overview

R.W. Oldford
Department of Statistics & Actuarial Science
University of Waterloo

## Abstract

Every software project has reasons for its existence and continued development. Quail extends the ANSI standard language COMMON LISP to facilitate data analysis and statistical modelling. Important extensions include a rich object-oriented statistical graphics and general interface building system, multi-way array manipulation, in addition to a suite of mathematical, probabilistic, and statistical capabilities. The primary motivation for this extension is to provide an open-ended computing research environment where quantitative researchers can freely mix symbolic and numerical computation, and rapidly prototype ideas. Fundamental to Quail's design is the close modelling of statistical concepts in software.

An overview of the Quail project is given emphasizing the integration of its design concepts.

## 1 Why free software?

Given that this session is devoted to "free software", it might be worth asking the question as to why it exists in the first place? Or, more specifically, what motivates people to create and distribute free software?

I think that most developers (of non-commercial software at least) will admit that an important personal reason is simply that it is fun. Building software systems offers a technical challenge, a creative opportunity, and a sense of accomplishment when a piece is finished. It can be very much like doing mathematics, but with maintenance.

The reasons that the software is free can be quite varied. For some it is strictly a political viewpoint, to prevent the power concentrating in too few hands. Related is the notion that, as in science and other intellectual endeavours, all benefit from a free exchange of information. Free availability of software disseminates the ideas to a larger potential audience and if you are first, the impact can be enormous (e.g. X-windows for Unix systems). Having successfully tested the waters, free software has often been a stepping stone to commercialization.

However, the most important reason is that free software permits and even encourages unfettered innovation. The software is free to be experimental and innovative.

In statistics there seem to be two broad categories of computational innovation. The first is methodological. The purpose here is to exploit the computational resource to develop new methodological tools. One thinks of the various computationally intensive methods that have been developed such as nonparametric function fitting, automatic classifiers, neural networks, model search and averaging programs, and so on. The code exists to implement and disseminate the methodology. The new methodology is the interest, less so the code. These programs naturally figure importantly in repositories of statistical software.

The other broad category of innovation is essentially technological. Here innovation lies not so much in the exploitation of the technology as it does in its exploration. As technological opportunities arise, they are explored to see how they might be made useful. Familiar examples include the development of command-line programming languages for statistics, dynamic statistical graphics, small memory algorithms for early microcomputers, whole statistical systems for pcs, use of windows, icons, and mice for interface development, and new programming paradigms such as rule-based and object-oriented programming. Oftentimes, these innovations further develop the resource and indeed occasionally *become* the resource for others. Though strictly speaking not methodological in nature they can have a profound impact on the nature of the discipline (e.g. statistical analysis and programming systems).

Like any generalization this categorization is necessarily fairly coarse. Yet, I think it has some use. For example, all of the software described in this session appears to fall under the second category while most of the computational statistics research appearing in statistical journals seems to fall under the first category.

Why then, would one pursue free software whose raison d'être is technological innovation? Turning to another, this one famous, over-generalization might help shed some light.

On May 7 1959, C.P. Snow delivered the Rede lecture at Cambridge, entitled "The Two Cultures and the scientific rev-

olution" (see Snow 1964) and began what was to become a famous and controversial debate. In his lecture, Snow identified what he considered to be two distinct cultures, that of the 'literary intellectual' and that of the 'natural scientist'. Snow publicly worried that these two separate cultures rarely interacted or even understood one another. Moreover, neither much valued the 'intellectual' contribution of the other. Statistics, for example, is more naturally identified within the science culture than within the arts and traditional statistical education requires little or nothing from the arts. By 1964, Snow's hope was that a 'third culture' would develop to fill the gap between the two.

This year, in one of the AAAS's sesquicentennial celebratory essays in *Science*, Kelly (1998) of 'Wired' magazine identifies a now emergent third culture. But it is not the one Snow had hoped for. Rather than reconciling them, this 'third culture' could very well rival the two original cultures drawing off the best and the brightest young minds. Grown from a computer saturated popular culture, Kelly calls this third culture the 'technology culture', or the 'nerd culture'. He distinguishes it from Snow's two cultures by its purpose:

> The purpose of science is to pursue the truth of the universe. Likewise, the aim of the arts is to express the human condition. (Yes, there's plenty of overlap.) Nerd culture strays from both of these. While nerd culture deeply honors the rigor of the scientific method, its thrust is not pursuing truth, but pursuing novelty. "New," "improved," "different" are key attributes for this technological culture. At the same time, while nerd culture acknowledges the starting point of the human condition, its hope is not expression, but experience.
> ...Creation, rather than creativity, is the preferred mode of action.

Drawing from the now easy accessibility of novel computer technology, this culture is based "in technology, for technology". Novel tools are more easily created than theory and so we find that "[i]n the emerging nerd culture a question is framed so that the answer will usually be a new technology."

Hyperbole or not, there is truth in Kelly's conviction that some technological innovation is had for its own sake and that this innovation can quickly lead down research paths (good and bad) which might not otherwise have been taken.

In statistical software, it seems to me that the best work draws successfully and simultaneously from our 'two cultures' – scientific and technological. The first without the second can be good science, the second without the first good fun, but it is together that they offer the possibility of producing novel scientific tools.

# 2 Why Quail?

As do others, Quail draws from both cultures with the intention to develop new scientific tools for statistics. As Sheil (1983) promoted for programmers, the intent is to develop and to promote 'power tools' for statistical analysts.

Quail is intended to provide the applied statistician an integrated statistical environment. By 'environment' I mean that it is a self-contained system within which the analysis can be completely conducted. 'Statistical' means only that the environment is tailored towards the tools of data analysis and empirical modelling. And 'integrated' suggests that the components of the environment somehow fit well, both one with another and each with the analysis task. The research effort is to identify and to implement appropriate constituents of this environment.

Statistical analysis is tentative and exploratory by nature. An integrated environment must be flexible so as to allow the analyst to pursue new lines of attack whenever appropriate. A computational support environment needs to be built around abstractions which are suitable for the intended activity.

In Quail, then, part of the research is directed at determining the structures, functionality, activities, moves, and navigational tools appropriate for statistical analysis. That is, one needs to identify and implement these strategically important elements and herein lies an essential scientific aspect of the research.

The nature of statistical science suggests that these structures be incremental, the activities flexible, and any statistical strategy decentralized. All are to be naturally integrated as appropriate. If done right, such an environment should serve well as both a research and teaching device.

For research purposes, the environment should allow one to rapidly prototype new ideas in order to arrive more quickly at the good ones. These could then be transferred to production code if warranted. Going farther, the environment itself should be considered perennially experimental – its very design part of the ongoing research effort and so subject to change.

Training purposes would include encouraging technological innovation and training in statistical analysis. Indeed, it has for a few years formed the base of a statistical computing course at Waterloo (see Oldford, 1997).

# 3 The shoulders of giants

The technology key to Quail's design includes a large working memory (currently $\approx$ 20 MB), high resolution colour displays, and the multi-method, multi-parent class-instance object oriented programming style as available in *CLOS* (see Steele, 1991). The large memory assumption means that we feel free in Quail to maintain pointers to objects which might

reasonably revisited by the analyst. This is valuable for integrating software components.

Because design interest lies in determining the statistical components and features which would be important to the environment, Quail is designed to run on top of an existing fully supported Common Lisp environment (described in the Appendix). These are powerful modern programming environments that provide their users with programming tools unequalled by any existing statistical system. They should be part of any integrated statistical environment. It did not seem in the best interest of the statistical/scientific component of the research to re-develop these piecemeal for Quail.

This decision has two consequences. First, although Quail itself is free the base environment will typically not be. Second, Quail users will have full, and integrated, access to these powerful tools including the possibility to build their own specialized applications which employ Quail functionality.

# 4 Quail

In this section, I describe in rather broad strokes how the goals for Quail have influenced large scale issues of its design. An appendix at the end of the paper provides a brief sketch of the present base functionality of Quail.

1. *Software should model statistical concepts*. This is probably a distinguishing feature of Quail. The user is most comfortable manipulating statistically meaningful concepts. Consequently, the software environment should foster the illusion that the user is working directly with these familiar concepts. Because the environment is to be integrated to support this activity, we choose to represent these concepts, and their relations one to another, in the software (e.g. through object classes, class inheritance and generic functions) to match their statistical meaning. The identification and analysis of the appropriate concepts for integration can be an exciting scientific research problem in itself, the implementation a technological one.

2. *Interactive and Dynamic Visual Displays*. Key to statistical analysis, key to evoking the abovesaid illusion, and key to an integrated support environment, the importance of a good quantitative display model and implementation cannot be overstated. See the appendix for some of the features of the *Views* model used in Quail. Here we will note only that display components are typically modelled individually with highly localized behaviour and connected directly to an underlying 'viewed-object'. These components can be combined in nearly arbitrary ways.

3. *Robust design and low cost changes*. The possibility of making substantive conceptual design changes while minimizing painful consequences is highly desirable. The nature of object-oriented programming itself encourages this – multiple parents (or mixins) for the separation of conceptual 'orthogonal' structure, multiple generations of ancestors for incremental specialization, generic functions and methods for specialized behaviours.
   To this Quail adds automatic documentation constructed from stylized documentation strings defined at source, and topic generation from the hierarchical directory structure of source files.

4. *Focus by selection*. The context for analysis support is largely determined by direct mouse selection. What is the visual focus is taken to be the user's conceptual focus. This means, for example, that if the user has selected (highlighted) a collection of points in a scatterplot then any subsequent action (where sensible) is to be taken on that subset of the display, or that subset of the data as appropriate (recall that each view maintains a pointer to an underlying viewed-object). Programmatic means are also provided for the user to determine the selected focus.

5. *Managing variety in user action*. It may be simplest to confine user interaction to only one or two types – say command language interaction only, or only menu selection, or some combination of the two. In a support environment for statistical analysis, this seemed overly restrictive and arguably unnatural. Instead, a wide variety interactions are supported in Quail. To minimize confusion, both for the analyst and the system designer (ideally the same person), the interaction is stylized as follows:

   - Full programming language interface provided by the base Common Lisp system, extended by Quail functions and classes.
   - A global pull-down Quail menubar provides actions to be taken on the current focus, typically a window.
   - Pop-up menus appear in response to selecting components in any display. The menu is peculiar to the class of the selected view and the nature of the selection gesture.
     Middle-button pops a menu to change display style characteristics (e.g. colours, variates, ranges, as appropriate), right button pops one to make more fundamental changes to the display (e.g. cut, move, paste, link).

- Mouse selections with the CTRL key depressed access menus which act on the viewed object underlying the display component.

- Activity centres can be built up from any display components including a variety of control-buttons, sliders, etc. to build in any specially designed interaction.

- Display is a Quail generic function which when invoked on a Quail object, produces a display (possibly a full-blown activity centre) appropriate for that object.

- Special control buttons called signposts (of that shape) are reserved in Quail to capture an action from some object to a new visual display relevant to that object.

An example of the manageable but uncontrolled nature of interaction possible is the following. From the command line interface, the user has constructed a scatterplot. A subset of the points is selected and coloured green. From the global menubar the user chooses 'scatterplot by' from the plot menu. The focus being the selected window (assuming no display components within the window are selected), the dataset of the scatterplot is taken by Quail to be the data of interest. The user is prompted to select the 'by' variate(s) from a list of all known variates of the dataset plus the new variate 'colour' which refers to the categorical variate inferred from the different colours of points in the initial scatterplot. Suppose the user selects 'colour' as the single 'by' variate. Then the user is prompted for two more variates for the scatterplot. Having selected these the result is two scatterplots of the data, one for each coloured set of points. Both scatterplots appear in a single (new) window. Via a right mouse menu selection over the scatterplot of green points, the user might choose to paste a fitted least-squares line on this scatterplot. Then selecting the displayed line via a CTRL mouse selection, the user could choose to display the line's viewed-object, in this case a 'fit-object', possibly with signposts. The display might be of coefficients and t-statistics and the signposts would appear as 'signpost' shaped buttons to the side of the display. When pressed with the mouse, a 'diagnostics' sign might lead to a whole activity centre for regression diagnostics on a least-squares fitted linear model. Selected results from this diagnostic centre could be used in a programmatic fashion via user defined functions at the command language level.

# 5 Growing from here

Quail is already a considerable extension of Common Lisp for quantitative analysis. With no further research, the Quail environment is now well suited for exploratory data analysis with a complete, integrated, and easily extended interactive statistical graphics system.

Besides data analysts, researchers on statistical graphics would find Quail an ideal environment for developing new graphical methodology. Quail's graphical design coupled to the rapid prototyping capabilities of commercial Common Lisp environments provides a unique opportunity for such research.

Quail now provides a programming environment for developing technological support tools for statistical analysis. Following the broad strokes outlined above, there are a number of exciting research possibilities.

A promising approach in this regard is that of developing a technological 'workbench' for an applied area of statistics with which one is familiar. Narrowing further to a particular subject matter area might provide an even better focus.

Once within a comfortable domain, a variety of projects can be undertaken:

- Basic conceptual structures need to be abstracted and implemented.

- Useful displays for objects associated with these concepts need careful and specialized attention.

- Given the user is displaying one of the new statistical objects, and one has programmatic access to the instance, what would be useful activity centres to associate with signposts for each new class of objects? These need design and implementation. Each signpost leads to a new display but many signposts (from different objects) can point to the same display.

- What other activity centres would be useful? How would the user move from one another?

The approach taken is uniform, incremental. In this way, the expertise can be decentralized across many researchers, yet ultimately connected together in a simple uniform way.

Research of this kind requires individual persons, or teams, whose cultures are both scientific and technological to really pay off. It seems to me to be well worth undertaking and growing the results by making them freely available to others.

# Acknowledgements

Quail has been developed, and continues to be developed by many individuals including R.W. Oldford, C.B. Hurley,

D.G. Anglin, M.E. Lewis, and G.W. Bennett. Quail runs on Macintosh and on Windows operating systems via different commercial vendors (see references). Research has been supported by the Natural Science and Engineering Research Council of Canada.

# Appendix: Sketch of present functionality

- the **Common Lisp** programming language including its powerful object system allowing multiple class inheritance and generic functions which can type on any number of arguments.

- a base **programming environment** supplied by the Macintosh Common Lisp vendor (Digitool, 1997) which is quite modern providing lisp file editors, incremental compilation, program steppers, process backtrace, structure inspectors, and some program analysis tools (who calls, etc.).
  Similar features are available for the PC from Franz Lisp for their Allegro Common Lisp (1997).

- **Quail's multidimensional arrays** including
  — array mapping operators, too numerous to mention.
  — usual matrix operators including solution of linear systems. As an example of the implementation strategy, matrix inverse is by default postponed until needed for example in matrix multiplication at which point solve is called original matrix and the matrix it is being multiplied by.
  — matrix decomposition objects (e.g. QR, LU, SVD, . . .)
  — complete LINPACK collection of subroutines implemented for Quail matrices.

- **Quail's statistical functionality** including
  *Summary statistics* — mean, median, percentiles, sd, . . .
  *Data objects* — array objects containing some meta-data information.
  *Model objects* — Extended Wilkinson-Rogers specification of generalized additive models. Includes link-objects, etc.
  *Fit objects* — contains pointers to the model, the data, and results of fitting one to the other.
  *Probability objects* — classes representing a standard suite of univariate discrete and continuous distributions, including classes for a general continuous distribution, a general discrete distribution and an arbitrary finite mixture of either. All instances respond to density, distribution, and quantile calculations as well as requests for random values. Datasets are treated as empirical distributions for these purposes, making bootstrap calculations transparent as any other simulation.
  *Random number generators* — collection of linear congruential generators.

- **Quail's graphic objects.**
  *Views and viewed-objects* — A graphic in Quail is a data structure called a *view* which can be displayed simultaneously in any number of viewports. The metaphor is that each graphic is a "view" of some other object, its *viewed-object*. Hence every *view* data structure retains a pointer to the viewed object. See Hurley & Oldford (1991) for further detail.
  *Compound views* — views which contain subviews. Compound views position their subviews in a display. The compound view and every subview may have its own viewed-object; subviews can themselves be compound views.
  *Stock statistical graphics* — dotplots, boxplots, histograms, stem and leaf, 2 & 3D scatterplots, 2 & 3D line-segment plots, 2 & 3D function plots, scatterplot matrices, brushing, linking, . . .
  *Controls* — needle-sliders, bar-sliders, push-buttons, editable text-input, dialogs, pop-up menus. These could operate on anything.
  *View layouts* — compound views which layout subviews in row, column, or grid fashion, or at arbitrary user specified positions.
  *Interactive display* — every view responds to three mouse buttons (left, middle, right) alone or in combination with two modifier keys (shift and ctrl). Unmodified mouse buttons typically produce menus which refer to the physical display of the selected view; ctrl-mouse buttons refer to the *viewed-object* of the selected view.
  *Postscript output of any possible Quail display* — wysiwyg, colour encapsulated postscript.

- Miscellaneous mathematical functionality:
  symbolic and numerical differentiation, numerical integration methods, root-finding procedures, continued fraction expansion approximation, beta, gamma, and log-gamma functions, basic combinatorial functions.

- Automatic Documentation:
  constructed from specially structured documentation strings
  constructed from file organization

- Interactive help system.
  every newly defined function, class, method, etc. can have help displayed and or written out as a latex or postscript file.

- Two **strategic functions** having methods for any object (*Signposts* object . . .) — returns a list of "signposts"

particular to the given object; each signpost is a kind of control button view which if displayed and mouse selected would lead to some other relevant display peculiar to that signpost from that object.
(*Display* object . . .) — returns a view, which if drawn would produce a reasonable display of the given object. Display always accepts a boolean argument :signposts? which if true will return a view augmented by signposts.

Ctrl-middle-mouse on any view pops a menu offering the user the opportunity to call display on the viewed-object, with or without signposts. This means from a display, any viewed-object could be interacted with directly.

## References


Allegro Common Lisp (1997), PC and Unix based Common Lisp from Franz Lisp Inc, Berkeley California.

Hurley, C.B. and R.W. Oldford (1991). "A software model for statistical graphics" pp 77-94 of *Computing and Graphics in Statistics* edited by Andreas Buja and Paul A. Tukey IMA Series on Mathematics and its Applications, Volume 36.

Kelly, K. (1998). "The Third Culture", *Science* Vol. 279, # 5353, Issue of 13 Feb 1998, pp. 992 - 993.

Oldford, R.W. (1997). " Computational thinking for statisticians: Training by implementing statistical strategy", *Proc. Comp. Sci. & Stat.: Interface*, Houston, TX

Oldford, R.W., C.B. Hurley, D.G. Anglin, M.E. Lewis, and G.W. Bennett (1988-1997) *Quail: Quantitative Analysis in Lisp*. A statistical programming environment available free of charge from R.W. Oldford at the University of Waterloo.

Macintosh Common Lisp (1997), Digitool Inc., Cambridge Massachusetts.

Sheil, B.A. (1983). "Power Tools for Programmers", *Datamation*, Feb., pp. 131-144.

Snow, C.P. (1964). *The Two Cultures and the Scientific Revolution (2nd Edition)*, Cambridge Univ. Press, Cambridge U.K.

Steele, G. (1991), *Common Lisp: The Language, 2nd Edition* Digital Press.