A virtual laboratory

Matthias Schwab[1]

keywords: *Reproducibility, software, editor*

The future masters of technology will have to be lighthearted and intelligent.
The machine easily masters the grim and the dumb.
– *Marshall McLuhan, 1969*

Note: This is very preliminary and it may stay this way.

A computer offers an incredible creative space. Anything (and I virtually mean *anything*) we can precisely describe, we can somehow represent, simulate, or manipulate it in a computers bits and bytes. (John von Neumann when asked about intelligent computers is supposed to have answered that if anyone could tell how the brain works he could build a computer to simulate it.) Today, the computer is the premier research tool in many scientific and engineering research. The computer accomplish any computation accurately, fast, and cheaply. (I somewhere read that Gauss estimated the number of computations he did in his life were about 4 million. This equals about 300 computations every day for 40 years.) We can store huge amounts of data, process it, and visualize it.

Over the past decades, most researcher have developed a research environment. It probably comprises an editor, a text word processing or type setting program, a web page, a few compilers, a collection of research programs, and a collection of shared software such as a graphics or numerical library. But as the computer technology improves new opportunities arise and it is probably appropriate to continuously wonder, how to improve our little virtual lab. I do not suggest to implement all or any of these items: Often the diminishing return by replacing an okay-solution with an ideal-solution does not justify the effort. I advocate to stick to a mastered software as long as alternative only improve. I advocate a switch if the new software either accomplishes things impossible for the old, or if the new software promises a paradigm shift in my research. For Example, Java is the first language that incorporates the Internet. An object-oriented language such as Java is a new programming paradigm.

GOOD SOFTWARE CRITERIA

So what do I want from my software?

[1]**email:** matt@sep.stanford.edu

- **Ease-of-use, productivity and efficiency:** I want to prototype new ideas quickly. If successful on a small example, I want to apply the newly developed approach to large data sets.
- **Mobility and portability:** I want to organize my research for future access by me and others. I want to have access to my virtual laboratory wherever I am. I want to run it on any machine I might come across.

Furthermore, Easy things should be easy, hard things can be hard. (Mathematica and its book by Wolfram are a great example on making simple things simple, and difficult things possible.)

I like to use any tool that helps me to get my work done. But I shy away from tools that seem to make it difficult to switch to a superior tool if it becomes available. Similarly, I shy away from tools that do not collaborate well with other tools I use. For example, a LaTeX editor that does not allow me access to the standard underlying LaTeX ASCII text violates both criteria.

Furthermore, I like software tools that are widely used since they tend to be reliable (so called pseudo-standards) and ease the exchange of information and data. (The masses are great at avoiding bad software. But don't expect silver bullets in mass software.) Furthermore, new competing software will probably attempt to smoothen the transition for users of standard alternatives.

Example for good standard software solutions to common tasks are LaTeX for text typesetting, GNU make makefile implementations, or RCS for version control. Only if the software directly relates to my research area might I forego standard software for potential silver bullets.

THE LABORATORY NOTEBOOK

To be useful, research has to be reproducible and accessible. Fortunately, experiments on a computer are very reproducible and accessible by nature. The computer can easily do the same computation over and over again. The reproducibility and accessibility of a computational result can be preserved, if the research project is organized as a what SEP calls a reproducible research document (Schwab et al., 1996). Reproducibility allows the frequent and automatic recomputation of all results by a script: the essential function in the maintenance of research results. A simple paper publication of research does not suffice.

How about a laboratory book (don't be fooled by the name: this is of course a software application) that organizes reproducible documents? Currently, a reproducible document consists of LaTeX, programs, data files that are all linked by the rules in a makefile. The envisioned lab-book implements the same functionality in an object-oriented system.

The lab-book distinguishes the the contents level, the logic and functional structure, and the page layout description.

- the **contents elements** such as paragraphs, programs, figures, buttons, etc. These elements are references to *sources* such as files, URLs, applets (anything that can implement the yet unspecified documentComponent interface[2]. The sources are *loaded* only when needed.
- the **logical relationships** between the various contents elements (information that is usually stored in a traditional makefile). If the button is pressed
- the **page layout** that describes how the various elements ought to be presented (if possible). Applications that can handle page layout (e.g. postscript printers) will honor these instructions, while others (e.g. an HTML browser) will ignore them.

Is that new? Knuth literate programming (Web) connects programs and documentation. The Openbook (I think by SUN) does something similar, I think. There must be other efforts of that type for Internet browsers.

The lab-book can be automatically translated into several representations and formats, such as a chapter in a book in postscript format, a stand-alone HTML document, a standard makefile in ASCII, or a set of slides in DVI format. Depending on the environment that renders the lab-book, the various parts adopt. Each part knows of its appropriate adoption.

The lab-book is not supposed to be a huge stand-alone program, but a simple organizational tool that combines many independent software units to a functional document. I expect the various document components to be objects with some base functionality (such as a rendering in various environments). Alternatively, the translation could be handled by the lab-book using plug-ins similar to a browser. Each object has methods that declares if the object is up-to-date, that up-dates the object (build rule), and that frees any resources used when rebuilding the object (clean rule). For example, the paragraphs could still be written in LaTeX. To translate a document to HTML the LaTeXdocument is informed about the new format. It then invokes its latex2html translation. If the script is not available, the LaTeXfile *detex*s itself and returns an ASCII version.

Such a lab-book would of course be reproducible: any reproducibility command translates directly to the reproducibility command of its components. I discuss object-oriented reproducibility in greater detail in (Schwab, 1998).

I want to distribute my lab-book documents on the Internet. If the lab-book were written in Java, the lab-book itself could become the document that is transmitted in a net connection (rather than a derivative such as an HTML or postscript version). By the way, PDF fails to distinguish the logical level and the page layout level. Consequently, the document cannot be adopted to various environments. CD-ROMs may still be needed for reliable data storage.

Object-oriented make dependencies

Objects could implement a *Target* interface with three simple methods:

[2]This replaces the file as the building block of make

- `boolean isUpToDate()`
- `void upDate()`
- `void clean()`

The first two are self-explanatory. The third frees any resources the object may have occupied while up-dating. Maybe this method should be dropped and the upDate method should simply free these resources when done. A target might include a set of targets itself depends up-on. In that case the methods above would simply invoke the corresponding methods of the constituent targets. Such a object-oriented dependency mechanism would naturally support the reproducibility commands. To ease the transition, I would love to have Jake, a GNU make implementation in Java.

Time synchronization

All computers could receive a single time signal: these time signals exist. The distribution could be via the Internet. This would allow makefile-like dependencies over the Internet. I own a wrist watch that receives the railways time signal while I am in Germany. The US military sends a time signal world-wide. Why don't computers use that to synchronize their time? We actually had quite some problems with this kind of stuff in the previous years at SEP.

Object-oriented filing

I want all the stuff - tools, projects, documents - pertaining to a single project to be *close* together: in the same directory or neighboring directories. But I do not want to duplicate stuff. When things are shared among projects, I move the two projects together and create a shared file. I call this object-oriented filing system and it is not at all easy.

The filing by project allows me to find quickly everything pertaining to a project, to combine various projects into larger ones, and to collaborate with others on specific projects without burdening them with unnecessary software.

I have a collection of directories that contain my various research projects in from of a reproducible document. A nice portfolio. The object-oriented filing allows me to combine projects into books (possibly with overlapping chapters).

Interactivity

I like interactivity for computer games, but I don't find them useful in research. By their nature they imply individual choice and consequently violate the universality of scientific results. Interactivity is useful in teaching situations (even often overstated) and in interpretative tasks.

On the other hand, I would like a parameter explorer: a script that invokes a given program over and over again with slightly varying parameters. An interactive visualization complement lets me explore the solution dependent on the possibly high-dimensional parameter space.

CONCRETE SOFTWARE TOOLS

Java I discuss my reasons for using Java in `http://sepwww.stanford.edu/sep/matt/jest`.

Jest: An inversion library Jest (Schwab and Schroeder, 1997) shows that Graphics, regularly sampled data, irregularly sampled data, solvers, operators are important geophysical software units with well-definable interfaces. Especially, the separation of solvers and operators permit mathematicians and geophysicist to easily collaborate without inhibiting each other. I believe Jest will become even more attractive in the future when we consider more complex inversion problems integrating various vastly different data sets (well logs, seismic, pump tests, etc) into a single Earth model.

Editor The Emacs editor and its commands have become a powerful pseudo-standard. My shell (Tcsh), my debugger (GDBM), my version control tool (RCS) all use the same syntax and I can invoke them within my editor. I wish I had a performance tool that executes within emacs the way my debugger does. Furthermore, I wish my debugger could display arrays graphically. Until recently I used Emacs as mail tool and for FTP. And I have not even mentioned the various editing modes, the command execution, the parenthesis checking while typing, and the endless possibility to extend Emacs myself. The editor is the queen of all software tools.

I see a future editor as a smart extension of the command line. I hope that editors and browsers will merge. Then all resources available are accessible through the same tool. The form of access may vary (some is slow and read-only).

A functionality I miss most in my operating system and editor-command line, is the consistent change of file names. I wish a change in file name would propagate through all references to that file name (hyper links, system links, etc). But I doubt that can be done efficiently nor safely.

Configure When distributing software, a configure script should figure out what software components the user needs to successfully install the desired software. Maybe RPM by Redhat is even better.

Modules I would like to use Modules to support various software packages without complicating the user account setup. Furthermore, Modules help distributing a software's environment variables, etc.

Seismic data class Have Geophysicists defined an SEGY mime type? I want an object-oriented wrapper for SEGY data files (Java). If you want to read it, you simply invoke its methods. For example a geophysical data set could be accessible by a simple query interface that give access to the data records and the data header values. The internal representation would be irrelevant (for example the data could be compressed).

DISCUSSION

Java allows me a simple object oriented programming language that includes powerful network and graphics capabilities. RCS works wonderfully, especially when used with emacs. TEX is just the best mathematical type setting language there is. Emacs is a wonderful, extensible editor. I wish it were a better web-browser.

In the future, we need to use the web. The web allows us to increase our effectiveness by sharing our research effectively. I would like to develop my research as object-oriented reproducible Internet documents. that integrate a diverse family of applications. The object-oriented structure would probably replace GNU make by objects that up-date themselves.

In the geophysical community someone needs to define a simple seismic data object with a simple interface for standard input and output operations. Ideally the interface would be independent of a specific implementation. The data application would be accompanied by a mime type.

REFERENCES

Schwab, M., and Schroeder, J. M., 1997, Jest, an optimization library: `http://sepwww.stanford.edu/sep/matt/jest`.

Schwab, M., Karrenbach, M., and Claerbout, J., 1996, Making scientific computations reproducible: SEP–**92**, 327–342.

Schwab, M., 1998, Reproducible documents and interactive graphics on the world wide web: `http://sepwww.stanford.edu/sep/matt/join/ijDoc`.