Towards a free matter economy (Part 4)

Tools of the trade

Terry Hancock

> A good scientist is a person with original ideas. A good engineer is a person who makes a design that works with as few original ideas as possible. There are no prima donnas in engineering.—Freeman Dyson

Imagine where free software would be today if it weren't for the GNU C Compiler! Just as free software depends heavily on free compilers, so does free design rely on having free computer aided design and authoring tools.[1]

Before the gcc was created, when free software had to be written on proprietary compilers, the software development community was limited to the very small number of people who could afford to purchase such tools—either because they were professional programmers or very dedicated amateurs.

The idea of "bazaar style" development hadn't yet been conceived, but that was just as well, since such small bazaar sizes would lead to a breakdown of the bazaar development strategy[2]. No doubt some centrally controlled "cathedral" projects (such as the GNU project) would've continued, but the overall effect would be an extreme chill compared to the hotbed of innovation that free software currently represents. Lacking the kind of professional-quality design authoring tools that are found in the commercial engineering workplace, the free design community is in just that situation today.

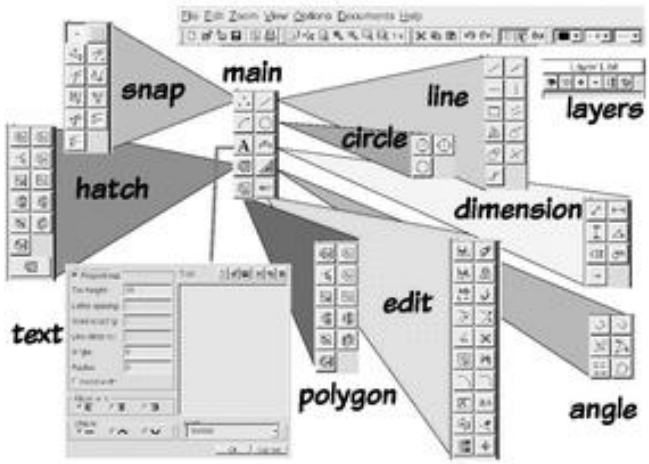
Of course, it can be (and has been) argued that engineering is a specialized discipline and that therefore only a small technocratic elite can participate in the process—hence bazaar size might always be too small to be effective. But twenty years ago, this was the conventional wisdom about software development, too!

By users for users

A compiler is a program to write programs, so the users of the tool are also those qualified to create it. With design tools, we're not so lucky: it's a fairly rare engineer who has the programming expertise to develop proper engineering design software. This is a problem because bazaar development works best when applications are developed by the people who need them. In order to get a CAD/CAM system started, it will probably be necessary to start with a centrally organized solution—a cathedral development model—working as quickly as possible towards a solution that relies heavily on *scripting* that *can* be done by the typical end user of engineering software.

This is the sort of approach that has proven itself functional in many projects in the multimedia and desktop application sphere: customization and scripting facilities have been the free software solution for desktop environments like KDE [3], vector graphics applications like Skencil [4], 3D animation and modelling applications like Blender [5], and particularly in game engines. All of these are situations where the same problem applies: the typical user is not a particularly skilled programmer, so a simplified programming environ-

Figure 1: QCAD GUI. QCAD has an excellent and intuitive palette-based GUI that allows quick access to drawing constraint features, and is easy to learn



ment is needed to make a more graded slope from user to developer.

This effectively increases the bazaar size for the parts of the program that can be scripted. User interfaces matter the most to the serious user and are the hardest for the programmer (who is not particularly skilled in the application domain) to predict. A user interface with a scripting engine gives a lot of leverage for the overall program, and the availability of easily-embeddable high-level interpreters such as Python make such a facility easy to provide. Thus, the "by users for users" philosophy of free software design can be stretched to suit real world applications, where skillsets vary between users and developers.

Computer aided design, simulation, and manufacturing

Although there are many other areas of design software which are useful and available, and many others for which no free solution yet exists, the most pressing and obvious need is for a general purpose 2D/3D mechanical Computer Aided Design and Computer Aided Manufacturing (CAD/CAM) system.

Specialized cases of CAD have already been covered by free software offerings—such as xcircuit [6] for drawing and capturing electronic schematics and pcb [7] for designing printed circuit boards, and general purpose 2D CAD drawing applications such as QCAD [8] are moderately well-developed (although they still fall short of proprietary competitors). The GNU EDA project[9] is making progress in the direction of integrated circuit design, and it should not be surprising that these "highly ephemeralized" technologies are among the first addressed.

Specialized cases of CAD have already been covered by free software offerings

Getting into the harder technologies, such as aerospace or mechanical engineering or robotics, requires much more sophisticated 3D CAD tools than those that are currently available. The modern manufacturing industry relies extensively on these tools to produce the kinds of complex technologies that we are accustomed to, and we won't have much chance of being competitive in the free design world, until we're able to use tools that are at least as good as these.

Friendlier interfaces

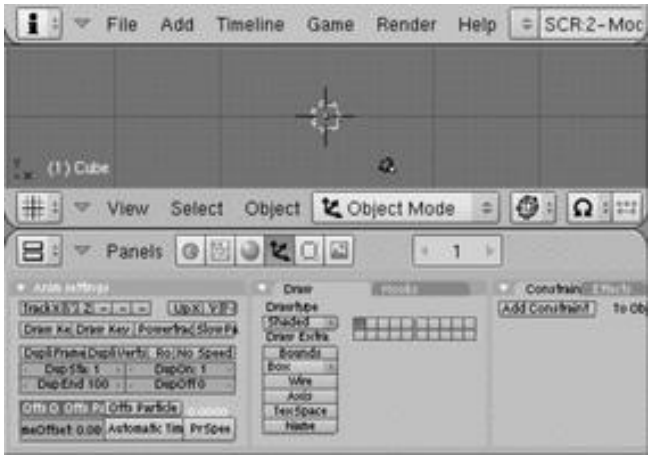
The interfaces in modern versions of proprietary CAD systems such as AutoCAD [10] have hardly changed at all in 20 years. They have traditionally relied on the high expressive power at low development cost of linguistically-oriented command line interfaces, leaving their visual-tactile graphical interfaces to stagnate. End user oriented free software artistic 3D tools like Blender, however, have seen much greater innovation, and they have demonstrated that a well-designed graphical interface for power users can greatly increase productivity, even for very complex tasks.

Collaborative design

In order to maintain a sufficient size for a free-design bazaar to establish itself and grow, CAD systems which are adequate to the task must be created. They must be free-licensed, so that all comers to the community can participate. They must use free, standardized drawing storage formats, and they must be friendly to version control and any other necessities of online collaboration.

Furthermore, they must permit a form of markup that allows communications about drawings to occur electronically. To my knowledge, this has never even been fully achieved even

Figure 2: Blender GUI. Blender's interface is cursed by new users, but greatly loved by power users. It does seem a bit daunting at first, but I quickly became attached to it. It compresses an enormous amount of options into a small space with an intuitive and highly internally-consistent design, using color-coded widgets, icons, and words as needed. If not actually used in a 3D CAD system, I think it should at least be emulated



with proprietary software, although I am aware of an abandoned attempt at the Jet Propulsion Laboratory called Supernova, based on "Lambda MOO" technology[11].

However, these network effects are precisely the sort of problems that free software is better positioned to solve! CVS, Subversion, and most forms of internet interpersonal communication are the types of software that started out free.

Integration with Narya Bazaar

In the Narya Bazaar system (see parts 1 (`http://www.freesoftwaremagazine.com/free_issues/issue_07/free_matter_economy`), 2 (`http://www.freesoftwaremagazine.com/free_issues/issue_08/free_matter_economy_2`) and 3 (`http://www.freesoftwaremagazine.com/free_issues/issue_09/free_matter_economy_3`) of this series), there's a vital role played by specification objects which must define work to be done adequately to form a contract between the donors, projects, and vendors. Whatever CAD system we create must be up to the task of defining these specification objects adequately for such agreements. That will require a commercial-quality CAD system, and it will need to support a range of meta-data from tolerances and materials to the relationship between components in large assemblies. Furthermore, as the bazaar community grows, parts from many different vendors will have to be integrated.

This has long been a problem in commercial design applications (Consider what happens when space station components from RK Energia in Russia need to be mated with components from Lockheed Martin in the USA), and good CAD systems constitute the solution. In the bazaar environment, virtually all components will be "outsourced", so it will be necessary to use good engineering communications processes to ensure that components interact properly.

Breaking the proprietary grip on the CAD market

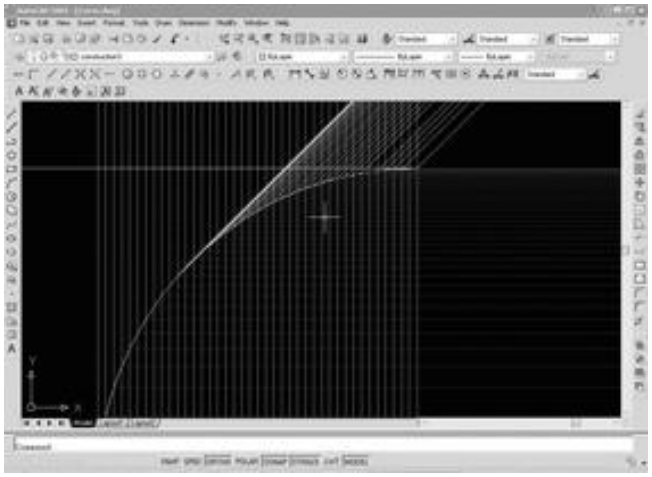
There is a kind of circular logic that keeps people locked into the systems they have—they are expensive and difficult to use because they have a very small user base. But the user base is small, precisely because they are expensive and difficult to use!

The providers of CAD software have effective monopoly power of a very limited market, defined by the restriction to the commercial industry who can pay the high licensing costs (typically $3000 per seat! [12]). Since the proprietary industry mostly develops proprietary technology, there hasn't been that much progress in making CAD systems fully collaborative, nor in making them integrate with communications channels to allow the virtualization of such standard engineering rituals as design-reviews and mark ups.

These systems should be "low hanging fruit", easy to pick off, but for one very unpleasant reality: 3D CAD systems are intrinsically very complex pieces of software that take a lot of work to write. Ultimately, however, they exhibit all of the criteria for a free software success story: they are complex, have rich network effects, and are just the type of high buy-in software that users expect to pay maintenance and support contracts for.

Unlike the desktop computing market, the manufacturing market is very much aware of the importance of standardization and the dangers of lock-in. Lock-in to single-source suppliers is a long-standing trap in the manufacturing industry, and managers already know the importance of avoiding it, which is why the value proposition of free-licensed

Figure 3: AutoCAD GUI. Although AutoCAD certainly has a GUI, many users find themselves relying on the command run line to accomplish most 3D modelling tasks, perhaps reflecting AutoCAD's origin as a 2D CAD system (Image credit Wikipedia)



software will be easier to present to them. On the other hand, there is still a credibility gap in proving that better (or even adequate) software can be maintained using free software development models—since conventional wisdom in the engineering community understandably favors engineered solutions (but there is a recent "rapid prototyping" trend, which could be comparable).

Pieces of the puzzle

We have to consider the individual components that are available to piece together a CAD system out of major elements that already exist in the free software community. To do this, I'm going to assume a "Model-View-Controller" (MVC) model of development [13], in which steps are taken to keep these major components separate from each other, although there are some reasons why we might want to relax this constraint.

Here I'm going to assume Python as an integration language, to simplify some of the choices, and also because Python is a language that is pretty popular and fairly easy to use for engineers as opposed to career programmers. For a project to succeed in the free software world, it's important that the implementation language be accessible to a sufficient quantity of the program's users to form a healthy development bazaar. [14]

Model (standard representation of CAD data and file formats)

CAD drawings are complex structured data, which is intrinsically object-oriented. Therefore it makes sense that some form of object database is used to hold drawing data, but the design of the schema is pretty complex.

There isn't one obvious way to represent a 3D object mathematically; there are many different systems, optimized for different applications. Fortunately only a relatively small number of the nearly infinite possibilities are actually in use, and so the representation of CAD data remains solvable. But it's a big enough problem to be recognized by the commercial manufacturing industry, and to call for some form of standardization.

Luckily for us, this has happened in the form of the Standard for the Exchange of Product model data (STEP) standard (also known as ISO 10303 and ANSI PDES) [15].

The most important contribution of STEP is not the data representation, but the classification and standardization of the object-oriented data models for representing CAD data. There are so many representations in use, segmented across so many industrial classifications, that collecting all of that into one standard is a monumental task, and the STEP standard schemas are indeed a monster as a result. However, this means we can regard the STEP standard as a fairly complete assay of what the manufacturing users need in a CAD system, which provides a blueprint not just for the file formats that we will use, but the internal data representations, and even the GUI controls that must be provided to manipulate that data. Even though STEP is intended as an exchange standard, it also provides a plan for developing the software itself.

STEP is not as open as we would like it to be. There don't appear to be any patent encumbrances, which is important, but the "source" documents from ISO are mostly copyrighted and available only for a fee [16], which you usually must pay to your national standards organization (ANSI in the USA, for example), and at least at present, the fees are too high for individual developers to pay. Since a free software development project relies on the cooperation of unaffiliated parties to operate, there's no easy way to spread the cost of these documents over a development team as there would be in a single company, so this practice is effectively protectionist to proprietary software vendors. This is not

Special domain design tools

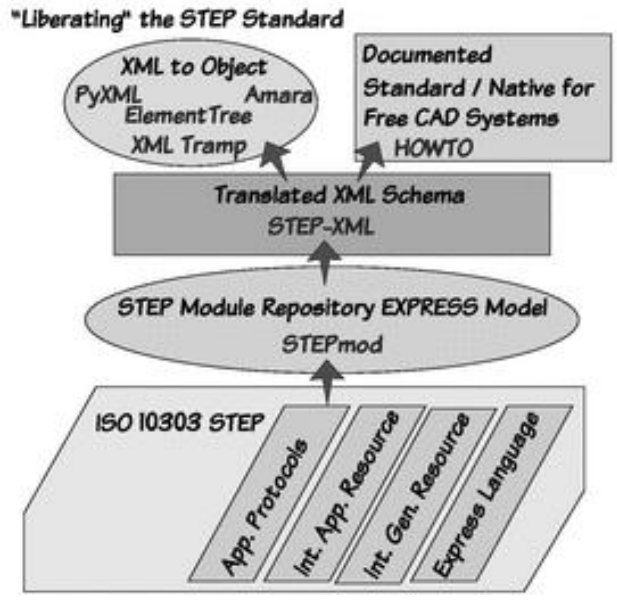
Although I believe 3D mechanical CAD is the most important design authoring challenge to be met, there are a lot of specialized domains for which there are existing free software design applications.
For software and systems design, there is GNU's dia program which makes several kinds of specialized diagramatic drawings easy to create. What can't be drawn with dia can be represented with more general vector graphic programs such as Inkscape or Skencil, and presented with KPresenter, or other free software slide presentation packages.
Electronic schematics can be drawn easily with more than one program, including for example, xcircuit and oregano.
The GNU EDA project primarily addresses integrated circuit design, and there is pcb for designing printed circuit boards. The biggest complaint I have about these applications is that they have very inconsistent interfaces. I can imagine a project to refactor these programs to a common GUI standard, but they are each well-established in a small design community of their own, so it isn't clear how popular such a project would be.
Shop drawings for mechanical projects can be created in 2D using programs like QCAD or PythonCAD, and indeed, this is how things were once done in the manufacturing industry, although it's easy to see that relying on this leaves us decades behind the commercial proprietary design world.

particularly popular with the manufacturing industry (who are the customers of those vendors of course, and would stand to benefit considerably from a free-licensed CAD solution), nor apparently with the ISO standards developers with whom I corresponded in preparing this article. [17]

So, if you look closely enough, you will find that a free-licensed open-source standard is there, in the form of the Express schema listings for STEP which are available freely (they are explicitly copyright disclaimed by the ISO, which means they are essentially subject to public domain rules [18]). If you regard the ISO "source" documents as documentation, which is really what they are, and the Express listings as the true source, then STEP can at least unofficially be used as a free format.

There are already two important sites dedicated to this mapping of the STEP standard into a free format, including the NIST's Step Modularization project [19], now hosted at Sourceforge, which provides an Express-to-XML translator, and the Engineering Exchange for Free (EXFF) site [20], which contains a number of useful resources for STEP. Combined with the free Express listings defining STEP, it's therefore possible to map the standard first from Express to XML, and then from XML to a variety of object-oriented programming languages, by using existing XML-to-object mapping libraries, available for popular free software programming languages, such as C, C++, and Python. Clearly, the wide range of alternatives in this mapping process means that there will be some issues with assuring compatibility, but it's a definite start at liberating the STEP standard.

Figure 4: Liberating STEP. The STEP standard could be "liberated" using the mapping tools from the Step Modularization project to convert Express schemas to an XML format. Once converted to XML, there's a wide range of object-mapping tools for different languages, including Python. It won't be a unique mapping, as it will also depend on the translation tools used, so it will probably be desireable to standardize on a toolchain after testing the alternatives, and provide free documentation for the XML-based derived standard



There's also a library called OpenCascade [21] based on the SDAI C++ implementation of STEP, but despite the name, this package is "non-free". Free software developers who want to base their work on an SDAI implementation would do better to go back to the original NIST C++ implementation [22], which is "copyright free" (produced by the US government).

Less expensive electronic forms of the complete ISO "source documents" may become available in the future, but there is no plan to make them freely distributable, as far as I have been able to ascertain. Working from the freely available documents to create a "liberated" CAD standard is probably the best strategy.

View (3D rendering)

At the most basic level, of course, are fast 3D rendering libraries and hardware, such as Open GL, but these are primarily for interfacing with (or emulating) hardware, and so are really more primitive than we want. Fortunately, many 3D rendering libraries have been built on top of these standards, primarily driven by game and computer animation applications.

Some of the more prominent free-licensed 3D rendering libraries that exist include: Crystalspace [23] (written in C++, mainly to support massively-multiplayer online games), Soya 3D [24] (written in Python and Pyrex, also mainly for games), and Blender [25] (written in C, mainly for commercial computer animation applications). All are under suitable free licenses (GPL).

Crystalspace and Blender are both primarily frameworks, which embed a Python interpreter; while Soya 3D is a python module library. The latter is better from the point of view of embedding the viewer as a major component, especially if the integration language will be Python. Blender also has embeddable builds: a game engine and a browser plugin, that might be investigated as options for a more pure "view" component.

In fact, though, the "impurity" of Blender as both "view" and "controller" may be an advantage, since there is already excellent control for viewing or browsing a 3D model at various detail levels. This is a large part of what one would hope to gain by using Blender, so it's worth considering abandoning the MVC model for this more complete system.

Controller (graphical user interfaces)

Given that the concept here is for a fully internet-connected collaborative CAD system appropriate for free-licensed design projects, a very attractive idea is to integrate it with cross-platform browser technology. This would mean using Mozilla's XPFE system, and therefore XUL for the GUI environment. The biggest drawback to this has hitherto been the limitation to Javascript as a GUI scripting language, which is less than ideal for a project of this type. By the time you read this, however, XUL will likely support Python bindings, so this problem is being resolved [26]. This would have the further advantage of easing integration with other out of channel communications such as email, web forums, and chat systems, in addition to the formal markup and review process that must be intrinsic to a collaborative CAD system.

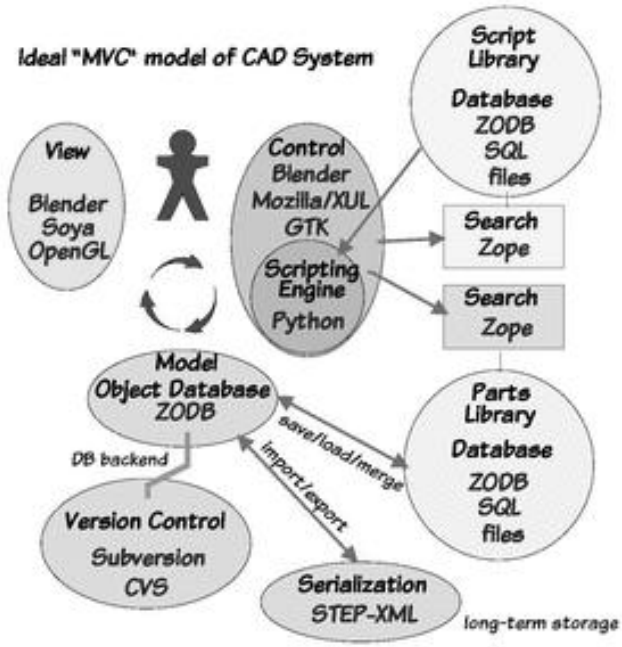
The design of the GUI, both as a skin style and as a guide to organization would do very well to follow Blender's lead. Although Blender's non-standard GUI is often criticized by newcomers, it is much loved by the people who use it daily. It is highly optimized for efficient use in 3D modelling applications by people who use it a lot. That makes it an excellent choice for a 3D CAD GUI as well. Although using the Blender interface itself (rather than emulating it with XUL, for example) is an attractive option, it has some technical obstacles: Blender isn't really factored into an MVC model. And, although there's some developer interest in this, it isn't a priority for Blender developers, who are focused on the core tasks of artistic 3D modelling and animation.

Other GUI options include the usual suspects, such as PyGTK, Qt, and wx.

Design concepts

There are obviously many, many different ways in which these components could be assembled to create a functioning, network collaborative 3D CAD/CAM system. Using Python as an integration language (as I have chosen to do for the server side on the Narya project), one could use ZODB [27] as an object model, deriving the schema from the express schemas for STEP via the Step Mod package. Then one could, for example create a Mozilla-based design using the new Python bindings for XUL and integrate with a cus-

Figure 5: Ideal MVC design. Ideal MVC designed CAD system elements. The purist approach to the CAD system would be to create separate model, view, and controller components (in a really ideal case, each would be pluggable, allowing for a variety of separate but compatible tools to coexist, but I'll be happy with just one of each). This approach is the soundest for development from scratch (or even nearly from scratch)



tom build of Blender based on the game engine or plugin versions.

Or one could abandon pure MVC design, and build the system entirely within the Blender Python API. Since Blender's internal model lacks features of the STEP model, and there is not likely to be a round-trip guaranteed set of transformations between the two, it is likely that it would be necessary to maintain two parallel models—a "representation model" within Blender, used for visualization, and a "design model" within the CAD database, which is the true "original". A transformation would be provided to render the design to representation in order to display it. Tweaks to the Blender source code (in C) would probably be needed to provide CAD objects with callbacks that could be handled by Python callbacks acting on the design model (otherwise, the representation model could drift away from the design model whenever non-CAD operations were performed on it, causing a synchronization problem and a lot of user confusion).

Sometimes you need a cathedral after all

Sophisticated 3D graphics and design modelling software is no picnic to write. This is one of those big, complex, highly-interdependent problems at which "cathedral" engineering seems to excel, and on which "bazaar" development frequently stumbles. The many half-finished projects in this area attests to this point. It will be necessary to create a successful prototype of the core of such a system before it can be built on and extended by a bazaar community. That prototype will require a lot of "up front design" contrary to the new conventional wisdom of "extreme programming".

Machines in general are complex, tightly-coupled systems, so it shouldn't come as a surprise that the mathematical models of mechanical systems are also complex and tightly-coupled, nor that the CAD/CAM software systems that manipulate those models need to be.

Fortunately, there is strong reason to believe a free software CAD system could be developed commercially. CAD systems are high-buy-in software, not unlike web servers, software compiler and library toolkits, or data reduction and analysis systems. They are primarily used within organizations which rely heavily on them working without a hitch. This means there is a substantial motivation for those or-

Too many cooks

In researching this article, I found nearly a dozen different initiatives to create free software CAD/CAM systems, started by different people, and ranging widely in success. None of them was on the scale of the need expressed in this article, but they make a clear case that such software is wanted, and there are people willing to do work on it. Also, few projects referenced the others, which suggests a lack of communication and organization.

I know that I don't really have the resources to pursue a CAD programming project myself (unless through some form of commercial collaboration), but what I certainly can do is provide a resource Wiki to keep track of other people's projects. Please have a look (`http://client.narya.net/Wiki/CadSystems`), and help me update the site with any additional projects that you know about. Maybe if the information is collected in one place, a solution will present itself.

ganizations to pay for maintenance and support contracts, and therefore a Cygnus Solutions [28] or Zope Corporation [29] business model (free-license the software, but charge for support services) would seem to be viable.

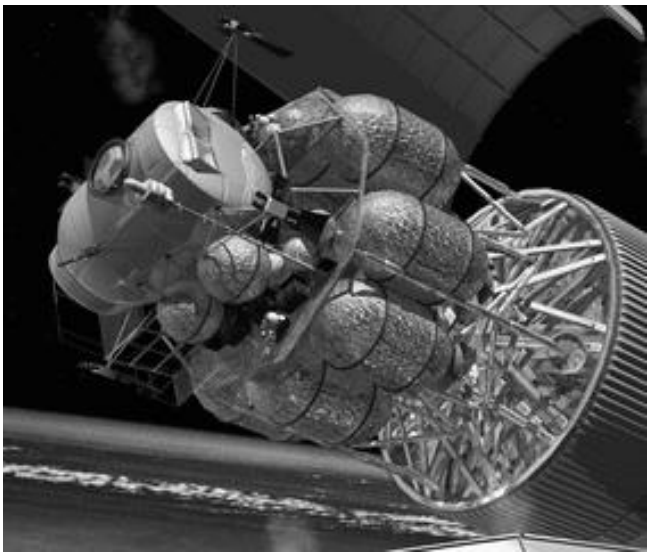
Free software tools and free data exchange formats are in the interest of the majority of the manufacturing industry, as demonstrated by initiatives such as EXFF. Thus we can expect to see cooperation and support from the end users of CAD once a serious effort is made to implement it, even though we'll also see competition from existing vendors of proprietary CAD systems, such as AutoDesk, and free software will still have to prove itself to a new set of users.

The real challenge is getting started. Cygnus started out supporting GNU software that already existed and was developed for quite different reasons, while Zope Corporation (then called Digital Creations) originally developed their product on a proprietary basis. It was only after these software products already existed and were relatively mature that these businesses made the decision to make support contracts their primary source of income. I'm not aware of an example where such a complex piece of software was developed by a company planning to release it for free and support it commercially.

Planning for the future

Although free design is already beginning to be done in limited domains now, the availability of better design authoring tools to a wider range of potential developers is needed to spur an explosion of interest in bazaar model development of hardware. Nowhere is this gap more severe than in the area of 3D mechanical CAD/CAM, which is extremely important for the kinds of aerospace and mechanical engineering that will be called for in the development of hardware that will be required by space pioneers as they attempt to settle and build using the materials they find in their new environment. If we want to follow a free development model with all the advantages it implies, we'll need this kind of software to be created, and there is reason to believe that it can be profitable as well as widely useful to develop it, although it will require a fair amount of entrepreneurial vision to make that happen.

Figure 6: CAD visualization. Model of hardware for the NASA "return to the moon" initiative. Visions of space frontier technology have relied extensively on CAD and 3D models, to the point that people expect to see such images before they take a technological idea seriously. It is far too much of a disadvantage for free designs to either reject 3D CAD or use proprietary 3D CAD systems. Blender can already create scenes like this, but without the technical data model to back it up (Image credit NASA/John Frassanito and Associates)



This is an area where a free software CAD system could not only match the proprietary competition, but leave it in the dust

This is an area where a free software CAD system could not only match the proprietary competition, but leave it in the dust—simply by leveraging the many years of experience in building collaborative, internet-based systems that are the legacy of free software developers worldwide. A true free software, general purpose, collaborative 3D mechanical CAD/CAM system won't be simply as good as the proprietary alternatives, it will likely be better than anything else on the market.

Bibliography

[1] Richard Stallman. The GNU Project (`http://www.gnu.org/gnu/thegnuproject.html`).

[2] Forrest J. Cavalier, III. Some Implications of Bazaar Size (http://www.mibsoftware.com/bazdev/), 1998.

[3] KDE Documentation (http://docs.kde.org/).

[4] Skencil Development Guide (http://www.nongnu.org/skencil/Doc/devguide.html).

[5] Blender Documentation (http://www.blender.org/modules/documentation/).

[6] xcircuit (http://opencircuitdesign.com/xcircuit/).

[7] pcb (http://pcb.sourceforge.net/).

[8] QCAD (http://www.ribbonsoft.com/qcad.html).

[9] GNU EDA (http://www.geda.seul.org/).

[10] AutoCAD (http://en.wikipedia.org/wiki/AutoCAD).

[11] Michael Brundage, Network Places (http://www.qbrundage.com/np/index.html).

[12] Price estimated from CDW Catalog (http://www.cdw.com), August 2005.

[13] Model View Controller (http://en.wikipedia.org/wiki/Model_view_controller) Definition.

[14] Terry Hancock. Praise for Python (http://blog.freesoftwaremagazine.com/users/t.hancock/2005/11/11/praise_for_python), 2005.

[15] STEP / ISO 10303 (http://www.tc184-sc4.org/SC4%5FOpen/SC4%5FWork%5FProducts%5FDocuments/STEP_(10303)).

[16] ISO Restricted Documents Notice (http://www.tc184-sc4.org/Notices/Authorized%5FUsers%2DPadlocked%5FDocuments/).

[17] Howard Mason, Chair, ISO TC 184/SC4. Private communication.

[18] Express Listings (http://www.tc184-sc4.org/SC4%5FOpen/SC4%5FWork%5FProducts%5FDocuments/STEP%5F%2810303%29/) (Search each block for “schema”).

[19] STEP Modularization Project (http://stepmod.sourceforge.net/).

[20] EXFF (http://exff.org/).

[21] OpenCascade (http://www.opencascade.org/) (non-free).

[22] NIST SDAI C++ Library (http://www.mel.nist.gov/msidstaff/sauder/SCL.htm) (free-licensed).

[23] Crystalspace (http://www.crystalspace3d.org/).

[24] Soya 3D (http://gna.org/projects/soya).

[25] Blender (http://www.blender.org).

[26] Brendan Eich Python for XUL scripting (http://weblogs.mozillazine.org/roadmap/archives/008865.html), 2005.

[27] Using ZODB Standalone. (http://www.zope.org/Wikis/ZODB/FrontPage).

[28] Cygnus Solutions (http://en.wikipedia.org/wiki/Cygnus_Solutions).

[29] Steve Litt. Zope Corporation/Digital Creations (http://www.troubleshooters.com/tpromag/199906/_digcreate.htm), 1999.

Copyright information

About the author

Terry Hancock is co-owner and technical officer of Anansi Spaceworks (http://www.anansispaceworks.com), dedicated to the application of free software methods to the development of space.