Appeared in the Proceedings of International Conference on Software Engineering (ICSE2001), Toronto, CA., IEEE Computer Society, Los Alamos, CA., pp.524-533, May, 2001.

A Case Study of the Evolution of Jun: an Object-Oriented Open-Source 3D Multimedia Library

Atsushi Aoki[1], Kaoru Hayashi[2], Kouichi Kishida[1], Kumiyo Nakakoji[1,3],
Yoshiyuki Nishinaka[1], Brent Reeves[4], Akio Takashima[3], Yasuhiro Yamamoto[3]

[1]SRA Key Technology Laboratory Inc.
[2]Open Source Business Dept, SRA Inc.
3-12 Yotsuya, Shinjuku
Tokyo, 160-0004, Japan
+81-3-3357-9361

[3]Graduate School of Info. Sci
Nara Institute of Sci. & Tech.
8916-5 Takayama, Ikoma
Nara, 630-0101, Japan
+81-743-72-5381

[4]TwinBear Research
6138 Gale Dr.
Boulder, CO. 80303
USA
+1-303-499-2666

aoki@sra.co.jp, rin@sra.co.jp, k2@sra.co.jp, kumiyo@is.aist-nara.ac.jp,
nisinaka@sra.co.jp, brent@twinbear.com, akio-ta@is.aist-nara.ac.jp, yasuhi-y@is.aist-nara.ac.jp

Abstract

Jun is a large open-source graphics and multimedia library. It is object-oriented and supports 3D geometry, topography and multimedia. This paper reviews the development of the Jun library from five perspectives: open-source, software evolution processes, development styles, technological support, and development data. We conclude the paper with lessons learned from the perspective of a for-profit company providing open-source object-oriented software to the community.

1. Introduction

Jun is a graphics library that supports 3D geometry, topology, and multimedia. Current commercially available 3D graphic libraries have almost exclusively focused on geometry in order to increase rendering performance, and do not handle topology well. A few 3D graphic libraries that can handle both geometry and topology do exist but are very expensive and thus limited to professional usage. The goal of Jun is to allow *regular* programmers to create, modify, and combine 3D objects without having to learn the complexities of mathematics and rendering. In addition, Jun supports multimedia data such as movies and sound. A user can write a program that plays multiple movies concurrently and places movie windows in a 3D space. Figure 1 (a)-(c) shows example programs to illustrate how simple it is to (a) create a 3D object, (b) take the image from a scene in the middle of a movie and (c) use it to texture the surface of the object.

Jun has several unique characteristics in terms of how the software has been developed in comparison with conventional software development. First, Jun is open source software developed within a for-profit company. The software is free and downloadable from our Web site. Second, Jun is developed on Smalltalk and adheres to a pure object-oriented development style. Third, Jun has been evolved over the last five years producing more than 360 versions and currently consists of more than six hundred classes. Fourth, data is available regarding the entire development process of Jun. We have collected data on each version of Jun as well as journals and mail messages exchanged via the Jun mailing list.

By using this data as a resource and by conducting intensive interviews with the Jun development team members, this paper reviews the evolution of the Jun library from five perspectives: (1) open-source, (2) software evolution processes, (3) development styles, (4) technological support, and (5) development data. Having source code and development data openly available among the community, we have been able to find what works and not works in object-oriented open-source evolutionary software development.

The next section gives a brief description of the Jun library. The following five sections discuss the above five perspectives respectively. We conclude the paper with lessons learned from our experience developing object-oriented open-source free library at a for-profit company.

2. A brief overview of the Jun library

Jun is a library for VisualWorks Smalltalk versions 2.5, 3.0 and 5i. It runs on VisualWorks Smalltalk platforms: Windows, Macintosh, and Linux.

Jun adheres to a pure object-oriented MVC (Model-View-Controller) architecture [10]. We have applied an object-oriented methodology in all phases of analysis, design, and programming. Because it is written in Smalltalk, Jun users benefit from features of this pure object-oriented language environment, including incremental garbage collection, multi-platform compatibility, virtual machine acceleration techniques including dynamic and just-in-time compilation, information hiding using objects, and abstraction through inheritance [7].

Basic functions provided by Jun include:

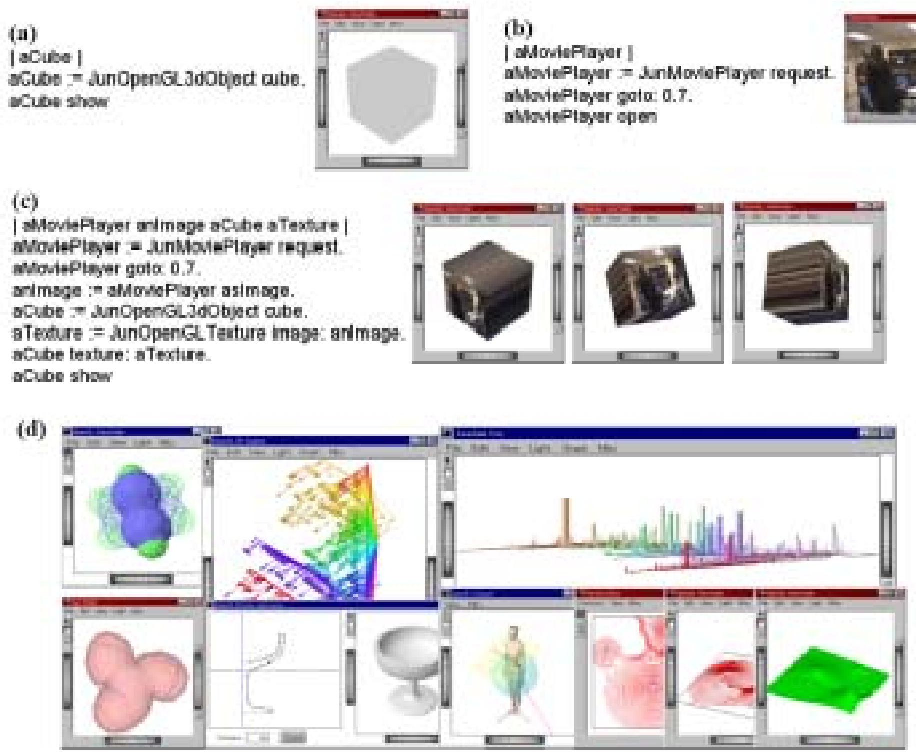
- geometric elements, such as point, line, plane, NURBS curved line, curved surface;
- topological elements, such as vertex, edge, loop, surface, shell, solid, and Euler operations, geometric operations, and set operations;
- rapid rendering using OpenGL;
- conversion between VRML1.0/2.0;
- operations on movies in AVI and MVI formats, sound in AIFF format, and images in JPEG and GIF formats; and
- importing Autocad^{tm} DXF files.

Jun goes beyond a typical utility library by also providing more application-like features including:

- *Viewfinder* (see Figure 1 (d)), an interface for 3D object display and selection, coordinate transformation, point of view transformation, calculation of visible volume, illumination, shading, wire frame, solid modeling, and add projection;
- 3D graphs using nodes and arcs, 3D animation, 3D plotter, and 3D charts;
- texture mapping of an image onto a 3D surface;
- the creation of a 3D shape through rotation of a 2D object;
- parameterized shapes, which are 3D objects whose shape can be altered using parameters;
- high-level image processing such as outline tracing and line thinning; and
- movie and sound players and editors based on Apple QuickTime^{tm}.

Each major class provides several examples similar to the ones shown in Figure 1. Our experience has shown that these examples play a crucial role in helping people learn about the library and put it to use.

As Open Source software, Jun is free, and has hundreds of users all over the world. Our ftp (ftp://ftp.sra.co.jp/pub/lang/smalltalk/jun/) and Web



(a)

```
| aCube |
aCube := JunOpenGL3dObject cube.
aCube show
```

(b)

```
| aMoviePlayer |
aMoviePlayer := JunMoviePlayer request.
aMoviePlayer goto: 0.7.
aMoviePlayer open
```

(c)

```
| aMoviePlayer anImage aCube aTexture |
aMoviePlayer := JunMoviePlayer request.
aMoviePlayer goto: 0.7.
anImage := aMoviePlayer asImage.
aCube := JunOpenGL3dObject cube.
aTexture := JunOpenGLTexture image: anImage.
aCube texture: aTexture.
aCube show
```

Figure 1. The use of Jun: (a) creating a solid cube shape (b) random access to movie frames (c) texture mapping from a movie scene (d) collage of Jun features

(http://www.sra.co.jp/people/aoki/Jun/Main_e.htm) sites received more than 200,000 hits in March 2000 (see Figure 2). Jun has now gone through more than 350 version-updates and the latest version consists of more than 650 classes. As described later, Jun has been ported to Java and C++ by both internal and external collaborators.

Jun began as a refactoring [4] of a 2-D graphic library that one of the authors started in 1991. In June 1995, when OpenGL was made available as DLL by the Windows95 operating system, we started constructing 3-D graphic libraries on top of the 2-D graphic library. In September 1995, the first version of Jun, Jun004, was made public as free software.

Since then, several small- and large-scale projects have used Jun as a substrate. Each new project caused additions and changes within Jun as further discussed in Section 4.

3. Open source software development

Jun is an open-source software library freely available at our ftp site. It is published under the terms of the Free Software Foundation's (FSF) *GNU General Public License* [5], which protects the intellectual property rights of our company and the developers of the software.

Open source software adheres to a development process that promotes rapid creation and deployment of incremental features and bug fixes [13]. Open source proponents claim that having all source code open to inspection by the public results in higher quality code and quicker bug fixes.

However, Jun's evolution differs from other well-known open-source systems such as Linux [18], PERL [19], or Apache [3]. Instead of a wide community of programmers each contributing a small part, almost all of Jun was developed by a small group of three to five programmers at a time. Though the community did not provide much source code, it did provide feedback, feature requests and bug notices.

This section discusses characteristics of Jun development as open-source software development. We explain two different types of observed effects of Jun's being open-source, describe how our company views the development of open-source freeware within a business model, and show how open-source developers produce high-quality software.

3.1. Two types of effects of open-source products

We noticed two major effects of Jun being open-source (Figure 3). The first is well known in the open source community as described in Raymond's Cathedral and Bazaar paper [15]. We call this aspect *incremental growth*. Here, the community of Jun users finds and fixes bugs and contributes enhancements. The community of users exists inside and outside of the company.

The second effect can be described as *reference model development*. Developers use the source code as a model for porting Jun to other languages, for example C++. In this case, the Jun source code itself is not directly affected, but its underlying object-model may be. In other words, users do not simply re-use source code; they use our object model for the domains of geometry, topology, and multimedia data handling.

There is no single *correct* answer to design object models in any given domain. We have made certain design decisions to handle topology, geometry and multimedia data. If the object-model architecture reflected in our design decisions becomes widely used in the community, we can play a leadership role by continuing to improve and refactor the architecture.

3.2. Open source development at a commercial company

When people learn that Jun is free, the question people ask is: "*How can your business survive giving away such complex software?*" There are several ways a company

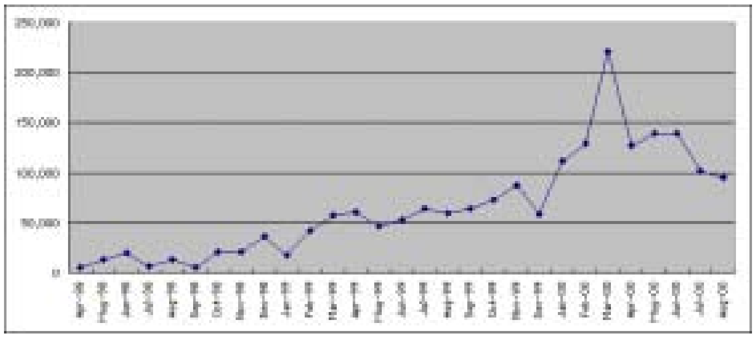


Figure 2. Jun Web page accesses

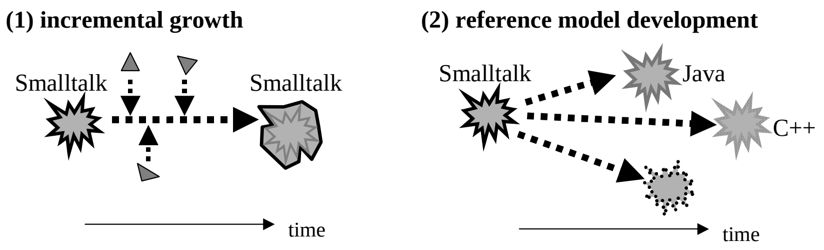


Figure 3. Two types of open-source effects

can profit from open-source products. One can produce learning materials such as textbooks and training, distribute packaged versions of the software, build development tools for the products, and make proprietary extensions [14].

Our business case for open source software is twofold: long-term leadership and consulting services.

First, as discussed above, Jun is beginning to serve as a reference model in 3D graphics and multimedia data handling. Although this does not directly generate income, it serves to advertise our competence in the long run.

Second, many customers need to modify the Jun library for their specific needs. In theory, open-source allows anybody to change the software as they see fit. But in practice, regular computer users and end-users cannot actually do too much with source code [12]. They need expert programmers to adapt the software to their needs.

It is possible for other companies to develop products with Jun, or to change or enhance Jun. However, the library itself has now become so large and complex that it is not easy for outsiders to completely understand it. Consequently, people ask us to help them use or modify Jun. Though the source code is free, the learning curve is steep and clients can save much time and resources by using us to help them customize Jun.

3.3. Open source as quality assurance

One of the major contributors to increased software reliability and quality has been *walkthroughs* [2]. One reason that walkthroughs work is that simply knowing others will be reviewing their code in detail causes programmers to work hard at writing clear and high quality software. Open source software takes code inspection and walkthroughs to the next level, because now programmers know that *everybody* will be able to see and review their work.

It is well known in the software engineering community that much software is written as a quick hack to meet a deadline. Such code is difficult to understand and maintain. In contrast, the Jun project has never experienced this kind of problem. Because the development team members are all aware that programmers all over the world will see source code, they take the time necessary to write clear and high quality code and add examples to show how it should be run.

Making the source code open has motivated the whole development team to observe disciplines such as: keeping design simple, continuously testing each portion of a program and integrating as often as possible, agreeing to coding standards, including documentation, and continuous improvement of the source code through refactoring [4]. Interestingly, these disciplines correspond with most of guidelines suggested by *eXtreme Programming* [1]. As we discuss more in Section 5, the development style of Jun in fact has been found very similar to the one advocated by the eXtreme Programming approach.

4. A model of evolutionary process

Jun has evolved through 360 versions released over the last five years. Usually open-source software evolves continuously from feedback from the community of users.

In our experience of Jun, however, the evolution is not simply driven by feedback from the community. As we briefly mentioned in Section 2, several large-scale projects using Jun identified new needs for Jun, which also guided the evolution of Jun.

This process is similar to the biological evolutionary process [11]. According to Maturana and Varela, changes are determined by the structure of an organism and a perturbation. A perturbation itself does not determine how the organism evolves, but it triggers the organism to change its structure. The evolved organism with its new structure affects the outer environment and produces another perturbation. This iterative process of the interaction between the organism's structure and the environment through a perturbation is a driving force of evolution.

In the case of Jun, customers might need something that Jun does not fully support. New requirements emerge from the project and they serve as a perturbation to evolve Jun. Refactoring and other evolutionary changes also take place. Each new version of Jun then attracts new customers with other needs and interests.

There are two types of evolution in biology: *phylogeny* and *ontogeny* [11]. The former refers to the evolution as species while the latter refers to the evolution of individual living beings. We have also found two types of evolutions in the history of Jun corresponding to these two types.

Incorporating bug-fixes and minor change requests causes Jun's version updated; however, this usually does not affect the underlying class structure and change the fundamental nature of Jun. This is similar to an *ontogenic* process where an individual living being grows.

In contrast, there have been several major version updates in the history of Jun. They were caused by large-scale projects using Jun. The Jun team members identified new needs for Jun, which required major modifications in its underlying object models. Two versions, before and after such a major change, are different in nature belonging to two different *stages*; Jun has experienced *phylogenic* evolution.

Figure 4 illustrates how both ontogenic and phylogenic evolution took place in the history of Jun. Contributions by Jun user communities such as bug reports and change requests as well as refactoring by Jun team members continuously evolve Jun within a single *stage*. This has been reflected in *version updates*. In the mean time, large-scale projects caused the phylogenic evolution of Jun producing *stage updates*.

Figure 4 shows how version updates (ontogenic evolution) and stage updates (phylogenic evolution) took place in the period between September 1995 and the end of 1999. When the first version of Jun was released in September 1995, it got some attention from the Smalltalk community. Several bugs were reported to us as well as refinement requests. This *Stage1* of Jun was simply a "3D geometry modeler."

Then, the first large project, HQL started in August 1996. The project was about the development of human sensory indices, particularly measuring environmental adaptability of human bodies, and product adaptability. The goal was to produce a 3D human body model that a user can directly manipulate and simulate its movement within a 3D environment. This motivated us to have strong integration of Jun with Open-GL, to be able to handle 3D topology, and to achieve fast rendering. This has evolved Jun into *Stage2*. This stage of Jun was a "3D geometry and topology modeler."

Another large-scale project that served as a perturbation for Jun was a project that supports empirical software engineering, called NSN. The NSN project started in July 1999. In order to build a user interface for empirical studies, Jun needed to handle multimedia data such as video and audio data within the 3D space. This motivated us to add multimedia functionality to Jun. *Stage3* of Jun was then called a "3D geometry and topology modeler with multimedia handler."

Other large-scale projects that phylogenically evolved Jun include educational contents authoring, and computer integrated manufacturing for large ships and tankers.

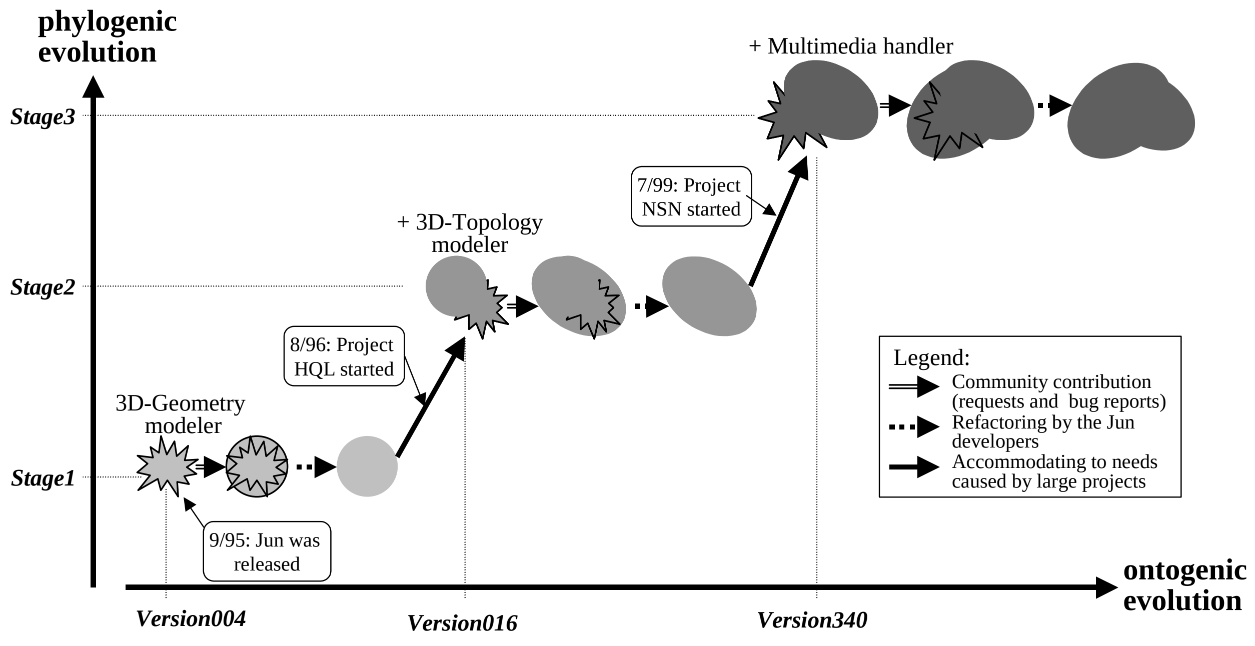

phylogenic evolution
Stage3
Stage2
Stage1
+ Multimedia handler
7/99: Project NSN started
+ 3D-Topology modeler
8/96: Project HQL started
3D-Geometry modeler
9/95: Jun was released
Legend:
Community contribution (requests and bug reports)
Refactoring by the Jun developers
Accommodating to needs caused by large projects
ontogenic evolution
Version004
Version016
Version340

Figure 4. Evolution of Jun

5. Jun development style

Jun was written by a small group of programmers with a leader who served as a manager for the entire Jun project. We observed that Jun's evolution is highly dependent on this chief programmer's management style. We found that strong leadership and selection are the two keys that have made Jun a successful object-oriented open-source library.

The project leader receives all email messages concerning Jun. Though the library is large and the amount of mail communication has increased, the leader still sees every message. His team members develop Jun classes and methods, and the leader integrates the newly added portion to an officially released Jun version upgrade. Users from the community also send in their own contributions, but the leader frequently chooses to revise the contribution to maintain the integrity and coherence of the library.

eXtreme Programming (XP) has recently gained attention in the software engineering community [1]. The XP style is characterized by: early, concrete, and continuing feedback from short cycles, incremental planning approach, flexible needs and functionality, automated tests, evolutionary design process, and communication. As we discussed in Section 3, the Jun development style is very similar to that of XP. XP values communication, simplicity, feedback and courage.

Communication. Jun development team members work closely with each other either physically or logically linked by frequent mail exchanges. Jun mailing lists have seen extensive use to keep project members aware of the current state of Jun. They strongly encourage good documentation to increase communicability of source code. The naming convention embraced by the Smalltalk culture also helps the communicability of the source code. For instance, they use names such as `JunMoviePlayer` and `openAndPlayAndCloseWithoutTracker` as a class name and an instance method name. Good names have been shown to increase subsequent programmers' ability to understand and reuse programs.

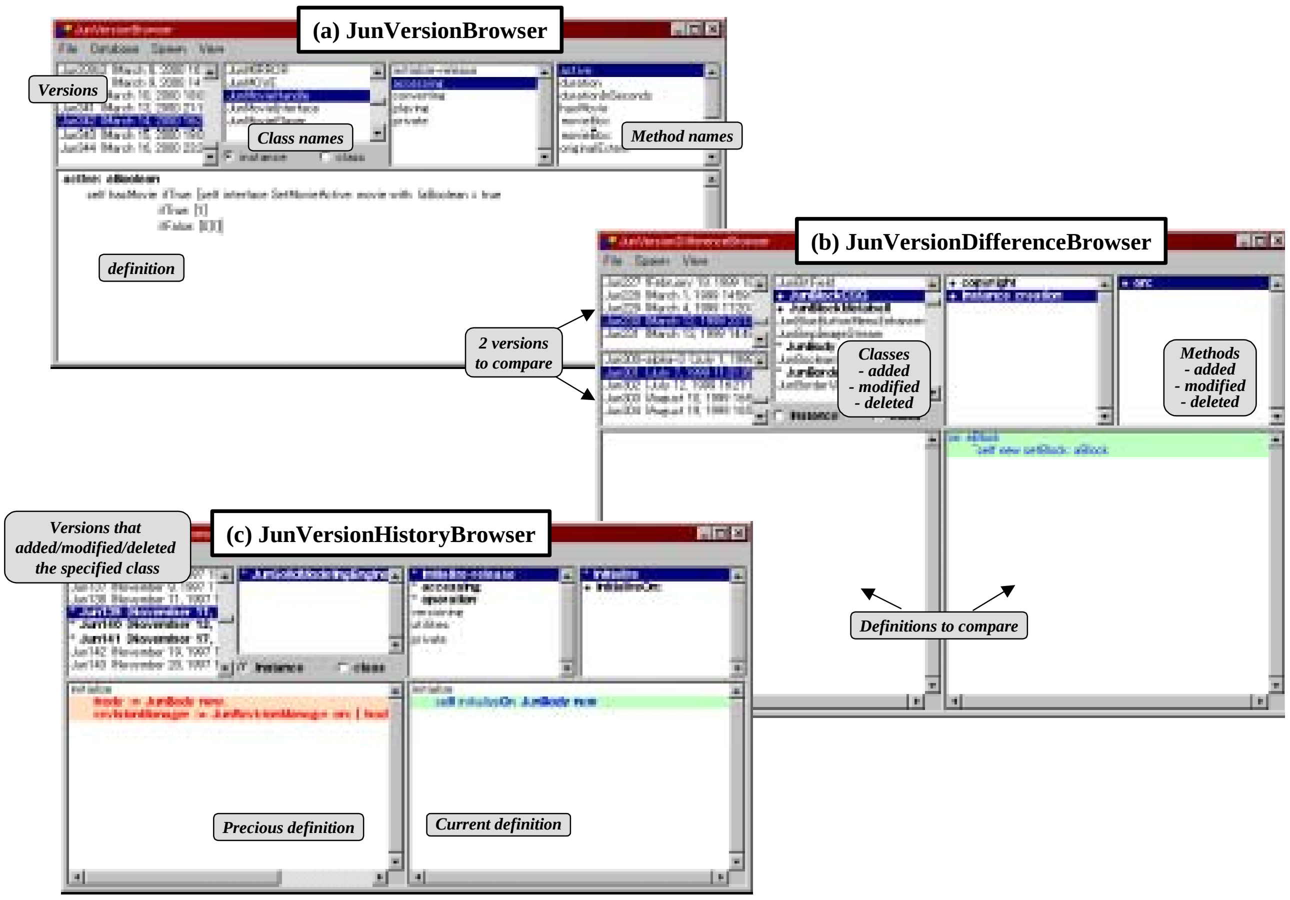

(a) JunVersionBrowser
Versions
Class names
Method names
definition
2 versions to compare
(b) JunVersionDifferenceBrowser
Classes - added - modified - deleted
Methods - added - modified - deleted
Definitions to compare
Versions that added/modified/deleted the specified class
(c) JunVersionHistoryBrowser
Precious definition
Current definition

Figure 5. Tools used to support the evolution of Jun

Simplicity. Simplicity has also been a key driving the Jun project. Jun is a library, and not an application itself. Library components need to be simple and general because they cannot be widely used otherwise. As we discussed in the previous section, although Jun is a library, it was shaped with applications as a driving force. When constructing new classes and methods for Jun, Jun development team members carefully design them so that they would maximize generality and extensibility.

Feedback. Jun has mainly grown in small, incremental steps. 360 versions in five years development time means six releases per month on average. This means more than once per week.

Courage. Finally, courage has also been necessary. Jun has been refactored many times [4]. Refactoring always involves a certain amount of risk. The usual rule is, "*If it's not broken, don't fix it*." Refactoring goes directly against this. As we show in Section 7, sometimes a new version will remove a large number of classes and methods for the sake of refinement. There is always the risk that something important will get deleted and will be difficult to put back in. This attitude is motivated and supported both by the fact that code is open for public, as well as by the project culture, that they are working for "Jun."

6. Technological support for evolution

Releasing 360 versions of a large library requires technological support, which in this case was written into the library itself.

`JunVersionDatabase` is a class that stores versions of Jun. A `JunVersionDatabase` consists of a sequence of `JunVersionChunks`, each of which contains one version. A `JunVersionDatabase` produces a "`VDB` (Version DataBase)" file that contains information about multiple versions.

Jun also provides interfaces that help programmers browse the content of the `VDB` file. Figure 5 shows three types of version-database browsers. Figure 5(a) shows `JunVersionBrowser`, which is an extension of system browser that is originally provided by the VisualWorks Smalltalk environment. It allows a user to select a version stored in a version database (the top-left window), then the system shows class and method definitions (in the bottom window) for that version. The top-right window shows a list of method names and selecting one of them will display the definition of the method.

`JunVersionDifferenceBrowser` (Figure 5 (b)) allows a user to compare two versions. The system shows two definitions of a selected class name or a method name from two specified versions one in each window at the bottom. Color is used to identify changes, deletions, and additions.

Finally, `JunVersionHistoryBrowser` (Figure 5 (c)) allows a user to examine how specific classes have evolved over a series of version updates. When a user opens `JunVersionHistoryBrowser` with a set of class names, the browser displays versions in bold font in the top-left corner showing that those versions added, modified, or deleted one of the specified class or its methods definitions. The bottom-left window shows the previous definition, and the bottom-right window shows the current definition.

Along with frequent update of the software using these version management tools, the Web pages have been kept up-to-date. As the WWW and ftp logs attest, access via web is now of critical importance and as the library has grown in size and complexity, online documentation available as html files has played a larger and larger role in helping new users download and install the software.

7. Measurement data analysis of Jun

Figure 6 shows how Jun has evolved during the last four years. The figure illustrates how the total numbers of classes, class methods and instance methods have evolved over the 360 versions.

Godfrey and Tu has reported that they have found Linux, a widely known large-scale open-source software, has been growing at a super-linear rate [6] and that their finding does not conform to Lehman's law, which states that large-scale software grows more slowly as it gets bigger and more complex [9].

We have found that Jun's evolution does not follow Lehman's law either. If we use the number of classes as an index to one aspect of the Jun evolution, we can see from Figure 6 cycles of steep increase and flattened out modest increase. The steep increase indicates a period of time when major modifications to Jun were being made. This corresponds to the phylogenic evolution of stage updates as we discussed in Section 4. The modest increase indicates a period of time when minor modifications were being made. This corresponds to the ontogenic evolution.

Let us now take several examples of Jun development activities to describe how such phylogenic evolutions took place. First, there is a little steep around August of 1997 (see Figure 6). It was the time when we added support for VRML to Jun.

April and May of 1999 shows another steep increase, when we added major changes to Jun to make the library compatible with Linux and Mac operating systems on VisualWorks3.0. Before then, Jun was compatible with multiple platforms on VisualWorks 2.5, but only with Windows on VisualWorks 3.0. The changes did introduce not only many new classes but also large changes to instance methods; 2814 instance methods were added and 1199 instance methods were deleted to accommodate this

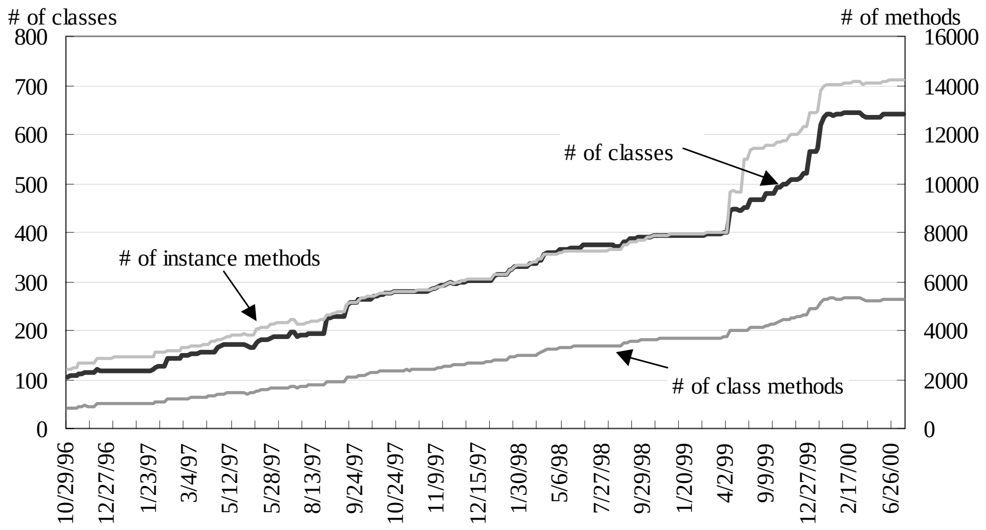


Figure 6. The numbers of classes, class methods and instance methods of Jun

change. As we discussed in Section 5, this is a good example of how the Jun team members demonstrated *courage* in refactoring the software.

In December 1999, we added two application-like features. One was called Teddy, a free-hand drawing interface for 3D objects based on the work by Igarashi [8]. The second was a terrain-modeling program that generates a 3D terrain model based on a topographic map data. Both made heavy use of triangulation, and many classes were added. Later, the Jun team refactored these classes as indicated by a slight decrease in the number of classes around February, 2000.

In March 2000, another major evolution happened. We ported Jun to VisualWorks5.0i and refactored support for Windows, Mac and Linux. Much new code was written while the number of classes actually not increased. In fact, if we examine how classes were added, deleted, and changed in each version, we have found a large number of classes were deleted and then added.

We have used this kind of development data as a means for reflection among the Jun development team members. Although graphs such as Figure 6 were useful, they were not helpful enough to examine some aspects of Jun evolution, such as how the growth of the number of classes and that of instance methods are correlated to each other. We needed more finely crafted, customized visualization tools that would allow us to interactively explore specific aspects of Jun evolution.

Based on this recognition, we have developed an information animation tool by using Jun to animate the evolution of Jun itself (Figure 7) [17]. Although this project has just got started, we have already found characteristics of class and method evolution by using the tool; while the number of classes increases prior to the increase of instance methods in carefully designed projects, the numbers of classes and instance methods simultaneously increase in poorly designed projects. This type of finding will be useful in identifying the quality of an object-oriented project by measuring artifacts produced by the project.

8. Lessons learned

We currently have about 100,000 monthly hits on our web site, and a growing list of customers requesting help with customizing the library to fit their needs. In terms of acceptance by users and worldwide distribution, Jun has been a great success story in the open-source field. In terms of how constantly the system has been evolved, Jun has been a success case of evolutionary object-oriented software development.

As we reflect on this story, the success of Jun seems to be due to the following factors:

Community leadership. As discussed in Section 2, open-source development does not necessarily only mean the cooperative development empowered by *many eyes* [15]. Object models that the source code is based on can be copied and ported to other languages if source code is open. The topology, geometry and multimedia-data handling architecture in Jun has been ported to Java by our internal collaborators and to C++ by people outside of our company. Open source software does not only mean that

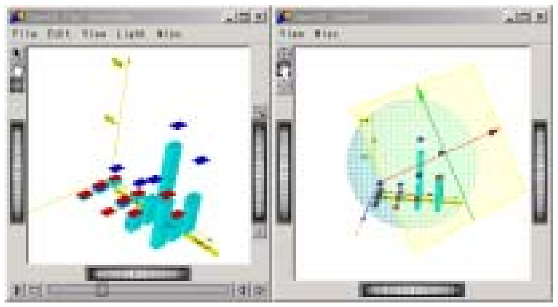


Figure 7. Information animation on Jun development data using Jun

the source code is open; it also means that the underlying object models are open. When the models become widely used in the community, we are able to play a leadership role in the community by continuing to improve and refactor the models.

High quality. Our interviews with Jun team members showed that much more time and effort is required of them for the Jun development than for other conventional projects because they know their code is going to be open to inspection by many other people. This pressure has driven the entire Jun development team to keep producing high quality programs. This coincides with our findings during interviews of university students; when asked why they would not make their programs open-source, they answered that their source code is "a shame" and they did not want to be publicly critiqued. Our experience has demonstrated that *open* source can now mean better quality than "*closed* source" software. We believe this aspect of open source will become more and more important in the software industry.

Evolutionary triggers. As discussed in Section 4, we have seen that Jun has gone thorough *version updates* (ontogenic evolution) as well as *stage updates* (phylogenic evolution). Especially for phylogenic evolution to take place, we need periodic perturbations caused by other projects. In one sense, which projects to take determines which direction the library goes, and selecting the "*right*" project in the course of the development of an open-source library is critical in keeping the evolution on the "*right*" track. Not all projects are suitable for inclusion – some would detract from the overall architecture and goals. The chief programmer plays a critical role in judging the suitability of new projects.

The role of a project leader. One fear in open-source development is that *too many cooks ruin the soup.* Trust and dependability are important aspects in the growth of an open-source project. Potential contributors must believe in the long-term direction and quality of leadership. The project leader of the Jun development team has been responsible for deciding which submissions to incorporate and which projects to accept. People trust his decisions. The promise of open-source software is not just access to the source code, but trust in the human leadership.

Tool support. As we discussed in Section 6, we made use of the version database tools, most notably when porting Jun from Smalltalk to Java, begun in 1998. The tools helped make one's contribution *tangible*. The project is large and an individual's contribution can get lost. These tools affirmed the role that each individual played by making the changes explicit.

Journals and mailing lists. Not only program versions, but also recorded mail communications and individual journals have helped Jun team members remember and understand why certain design decisions had been made. Because such documentation activities are not directly related to the actual product development, one tends to forget, avoid, or postpone tasks related to documentation. Journals were notes taken by Jun developers on individual basis and made public through internal Web pages as necessary. Discipline was necessary to maintain accurate logs of changes. The project leader has been playing an important role also in this respect.

Visual feedback on software evolution. The Jun team members have been accumulating data on the Jun development as presented in Section 7. Visual representations of such development data have been found very useful to help the team members reflect in what and how they have been doing, and to encourage them for further development. This coincides with the finding that the visual feedback of development data was useful to motivate the software engineering process group (SEPG) to push their process improvement activities forward [16]. A variety of interactive visualization techniques, including information animations [20], must be explored to more effectively give such feedback.

Nonlinear steps in class- and method-growths. In many important ways, Jun is typical of large-scale object-oriented development. As discussed in Section 7, Jun has evolved neither as a simple linear nor non-linear increase in objects. The number of classes increased or decreased depending on the type of the evolution. Doubling the number of classes may just be the result of copying a package to test a new class structure, and does not necessarily mean either evolution or improvement. Decreasing the number of classes, on the other hand, may be a result of refinement, or refactoring, rather than a decrease of functionality. Current measuring schemes for object-oriented programming, such as counting the number of objects, therefore, are not necessarily a very useful way of capturing the evolution of an object-oriented program. Analyses of data on Jun have demonstrated this point. We need to work on metrics and analyses, which can aid developers of large-scale object-oriented systems.

The role of examples. Not all interested users are programmers. Only a small fraction of users have the technical skills to download the library, install it in their environment, and then use it for their tasks. Typical users, even experienced programmers, ask as for help to make full use of the library. Semi-expert programmers sometimes need a seed created by us, to get them started in the right direction. One may argue that having many good examples could decrease the likelihood of customers asking us to do consulting work for pay. To the contrary, however, we have found that having many good examples makes the library accessible to more people and thus in the long run brings us more business.

9. Conclusion

This paper reported our experience with Jun, an open-source 3D graphics and multimedia library for Smalltalk. We examined our experience from several aspects and discussed lessons learned. We have found that much of the success of this project can be ascribed to its being open-source, object-oriented software. None of us have foreseen these success factors at the beginning of this project. The disciplines, XP-like development styles, self-producing development support tools, and evolutionary development patterns have all emerged in the course of this project.

Now that we have articulated what have been key factors for the successful Jun development, we hope to disseminate the findings and keep applying them to other software development projects.

10. Acknowledgements

We would like to thank the entire crew of the Jun development team, whose contributions made the current Jun possible. They are: Tomohiro Oda, Takanori Hoshi, Ryousuke Yamada, Katsuhiro Watanabe, Satoshi Nishihara, Ankur J. Chavda, Ryouishi Matsuda, and Minoru Matsuo. We would also like to thank Yunwen Ye and Keiko Kondo for their valuable comments and feedback on the paper.

11. References

[1] Beck, K. eXtreme Programming eXplained: Embrace Change, Addison-Wesley, Boston, MA. 2000.

[2] Fairley, R.E. Software Engineering Concepts. McGraw-Hill Inc., New York:, NY. 1985.

[3] Fielding, R.T. Shared Leadership in the Apache Project, Communications of the ACM, Vol.42, No.4, ACM, New York, NY, pp. 42-43, April, 1999.

[4] Fowler, M. et al. Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA., 1999.

[5] GNU General Public License, Free Software Foundation, http://www.fsf.org/copyleft/gpl.html.

[6] Godfrey, M.W., Tu, Q., Evolution in Open Source Software: A Case Study, Proceedings of the 2000 International Conference on Software Maintenance, San Jose, California, October 2000.

[7] Hopkins, T. Horan, B. Smalltalk: An Introduction to Application Development Using Visual Works, Prentice Hall, 1995.

[8] Igarashi, T., Matsuoka, S., Tanaka, H. Teddy: A Sketching Interface for 3D Freeform Design, SIGGRAPH 99 Conference Proceedings, pp. 409-416, 1999.

[9] Lehman, M.M., Perry, D.E., Ramil, J.F., Implications of Evolution Metrics on Software Maintenance, Proceedings of International Conference on Software Maintenance (ICSM'98), Bethesda, MD., November, 1998.

[10] Lewis, S. The Art and Science of Smalltalk, Prentice Hall, 1995.

[11] Maturana, H. R., Varela, F.J. The Tree of Knowledge: The Biological Roots of Human Understanding, Shambhala Publiccations,Inc., Boston, MA. 1998.

[12] Nardi, B.A. A Small Matter of Programming. The MIT Press, Cambridge, MA. 1993.

[13] O'Reilly, T. Lessons from Open-Source Software Development. Communications of the ACM, Vol.42, No.4, ACM, New York, NY., pp. 33-37, April, 1999.

[14] Ousterhout, J., Free Software Needs Proft, Communications of the ACM, Vol.42, No.4, ACM, New York, NY., pp. 44-45, . April, 1999.

[15] Raymond, E. The Cathedral and the Bazaar, http://www.ccil/org/~esr/writings.

[16] Sakamoto, K., Nakakoji, K., Takagi, Y., Niihara, N., Toward Computational Support for Software Process Improvement Activities, Proceedings of the 20th International Conference on Software Engineering, IEEE Computer Society, Kyoto, Japan, pp.22-31, April, 1998.

[17] Takashima, T. Kurakawa, K., Yamamoto, Y., Nakakoji, K., Representing and Interacting with Complex Data with Temporal Variations, IPSJ-SIG-HI-92-5, pp.31-38, January, 2001 (in Japanese).

[18] Torvalds, L. The Linux Edge, Communications of the ACM, Vol.42, No.4, ACM, New York, NY., pp. 38-39, April, 1999.

[19] Wall, L. The Origin of the Camel Lot in the Breakdwon of Bilingual Unix, Communications of the ACM, Vol.42, No.4, ACM, New York, NY., pp. 40-41, April, 1999.

[20] Wright, W., Information Animation Applications in the Capital Markets, Proceedings of InfoVis'95, IEEE Symposium on Information Visualization, New York, pp. 19-25, 1995.