Restoring a Legacy

Spencer Rugaber
College of Computing, Georgia Institute of Technology
and
Jim White
Nortel, Atlanta Technology Park

In 1508, Pope Julius II commissioned Michelangelo Buonarroti to paint the ceiling of the Cappella Sistina (Sistine Chapel) in Rome. Michelangelo labored for five years, much of it while lying on his back, to complete the task. In 1538, he was called back by Pope Paul III to add an enhancement, *The Last Judgment*, over the alter. This took seven more years. Michelangelo's work on the Sistine Chapel is surely a legacy–a gift from the past–impossible to recreate and worthy of preserving.

Software systems can also be legacies. The system you will read about, RT-1000, was originally built by a team of up to seventy developers, over a five year period, at a cost of millions of dollars. In its current incarnation, it produces millions of dollars of revenue, provides service to thousands of users, and would be prohibitively expensive to recreate.

In 1965, the Holy See commissioned a team of scientists, historians, and artists to restore the paintings in the Chapel. Even though the restoration decision was controversial, work proceeded over the next 30 years. Although no one would compare any software system to Michelangelo's masterpieces, the process of restoring the paintings has striking parallels with the process of reengineering software.

Over the course of the last four years, the RT-1000 development team has worked to improve quality and add features, turning what was a derelict into a high quality software product. This article describes the RT-1000 legacy restoration process, comparing the original to its current version and discussing factors that added and hindered the restoration process.

System Description

RT-1000 is a telephony software system for Automated Call Distribution (ACD). An ACD system comprises a set of software features that allows a large number of incoming telephone calls to be distributed to a designated set of agent positions. An agent might handle service requests, accept sales orders, or supply information to callers. If all agents

are busy, the calls are queued according to their priority and order of arrival. When an agent becomes available, the call that has been waiting the longest is presented to the agent. ACD features allow a supervisor to monitor the quality of service being provided to incoming callers. Status displays indicate how well different queues or individual agents are performing and where potential problems may exist. Detailed management reports highlight service factors such as average waiting times and the number of abandoned calls. To match fluctuations in call traffic, ACD supervisors can fine-tune employee schedules or even reconfigure an entire call center in real time. Load management capabilities allow a customer's ACD administrator to monitor and manage call load and configuration. Management information system (MIS) capabilities give the customer the ability to generate real-time displays and statistical reports on the performance of a group of agents. A typical customer for an ACD is a telephone company that uses it to handle customer service requests for a whole state. A typical installation includes 5000 agents and 150 supervisors. Figure 1 illustrates how the legacy RT-1000 system was normally configured before the restoration process commenced.

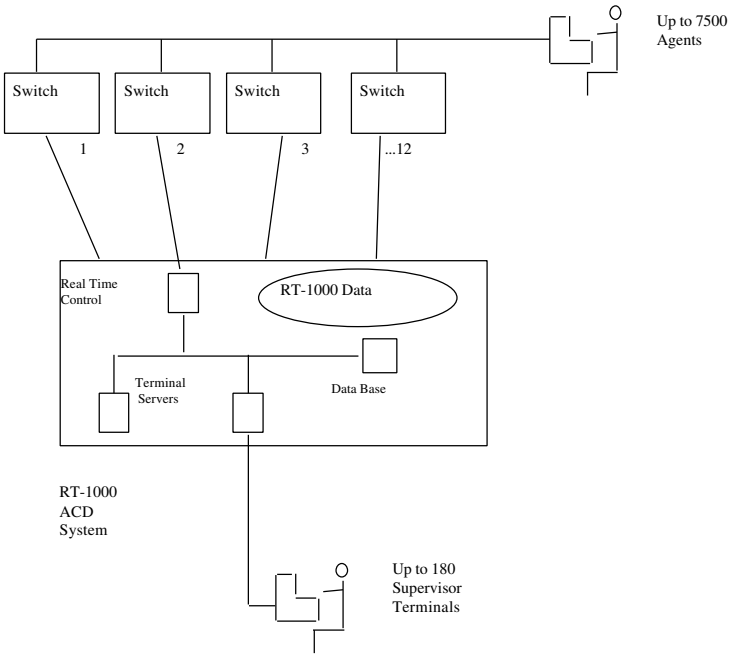

Figure 1: Original RT-1000 System Configuration

Initial Status

The paintings by Michelangelo in the Sistine Chapel comprise over 9,000 square feet. Previous restorations, including efforts to censor parts of the nude figures, left doubts as to the original conception. And centuries of grimy soot from candles that had been burned in the chapel had done an unknown amount of damage.

RT-1000 History

RT-1000 was developed by a third-party software vendor in the late 1980's and acquired by Nortel in 1990. For the next three years it was enhanced and maintained by Nortel before being outsourced to another vendor to be systematically rewritten. This effort failed and the system was returned to Nortel in mid-1994. By this time, the original design team had been disbanded and scattered, and the customers were quite unhappy.

RT-1000 was assigned to the Atlanta Technology Park laboratory of Nortel, where the cancellation of another project had left a department without a project. No staff members had any experience with ACD software, and, due the previous project's cancellation, staff morale was quite low.

Technical Difficulties

RT-1000 is a complex system. Among the things that make it difficult to deal with are the following.

- The overall size of the system is about 500,000 source lines of code which implements a long list of powerful and interrelated features. The system is written in multiple programming languages including Fortran (for the computational components), C (for the real-time part), an SQL-like fourth generation language (for the MIS), and various UNIX scripting languages (for system configuration). The architecture of the system comprises multiple architectural styles including real-time (managing up to fifty concurrent processes), MIS, computational, and an event-driven graphical user interface. Moreover, the system must satisfy a long list of difficult, non-functional requirements including data integrity, real-time response, distributed processing, reliability, information security, usability, performance under load, and openness to customer and third-party extensions.
- The small user base (six customer organizations) means that field trials can be conducted with at most two customers prior to general delivery. And the widely varying

way in which customers use the system presented further difficulties. For example, some customers use the workforce management feature as a key part of their business, while others do not use it at all. This makes field trials problematic in terms of guaranteeing that features are exercised from a true user perspective prior to making the software generally available. Moreover, when the system was first moved to Atlanta, it was not known exactly how customers used the system. For example, one customer used third-party (custom-designed) software to post-process reporting data. When a standard report was modified by the RT-1000 team (to add an extra column for additional precision, as requested by customers), the downstream software could no longer process the report. The customer based its payroll processing on the output of the downstream software and was therefore highly sensitive to this change. The design team often has no knowledge of these off-board systems. Another example is of a customer contracting with a third-party company for wallboard development (display of RT-1000 data on LED-based wallboards in remote offices). When a problem developed with the wallboard display due to a new screen sequence, the development team was contacted by the wallboard company requesting information, to the complete surprise of the RT-1000 team.

- The third-party software and hardware components of RT-1000 were out of date and no longer supported by their vendors. For example, the system included an out-of-date operating system, no longer supported by its vendor, with no means of controlling changes to the version on the field-deployed machines. The database management system, supplied by a commercial vendor, was also out of date. The hardware platform was no longer capable of supporting the customer demands placed upon it. Customer terminal software was proprietary and failed to conform to the emerging, Windows-based, industry standard. Finally, the distributed-component, interconnectivity mechanism required a local area network upgrade.
- Software process was non-existent. There was no version control on the software. Product testing was not formalized. And over three hundred outstanding Customer Service Requests (CSRs) existed.
- Little if any documentation of the software architecture existed, and there were virtually no comments in the code itself.
- And, of course, RT-1000 faced a serious Year 2000 exposure.

RT-1000 Restoration Strategy

The development team's basic approach over the first two years of the project was to determine its contractual exposures (the classes of deliverables Nortel still owed to its cus-

tomers), and to categorize its CSRs (the classes of known problems existing within the system). Armed with this knowledge, we targeted those areas of the system that would have the biggest payoff in terms of the two goals for immediate enhancement. For example, if a single additional feature responded to three contractual issues and also addressed a number of CSRs, it was given priority. Similarly, the largest two classes of CSRs accounted for over half the reported system issues, so these areas were given priority. Designers incrementally developed expertise in these classes/areas and were thus able to address issues with increasing speed. As the major classes were exhausted, repair efforts branched out into other areas.

After Restoration

Restoring the paintings in the Sistine Chapel was a massive undertaking. For the *Last Judgment* alone, more than a year of preparation was required, both working in a laboratory and experimenting directly on the painting itself. Restoration involved a team of participants including scientists, historians, and artists. The restorers had to contend with technical issues that varied across the surface of the painting. They had to develop an elaborate cleaning process including bath of distilled water and chemical mixtures, sometimes filtered through layers of paper and sometimes applied with sponges. Restoration involved not only cleaning the painting, but removing retouchings that had been added over the centuries. Finally, the restorers had to design an entirely new environment for the Chapel, including filtered air and a tailored microclimate.

Restoring a software system is not merely a matter of updating lines of code. In a sense, a whole new development environment needs to be constructed. And this has to be done while continuing to support customers and to develop new features.

Current Status

Over the past three years, the RT-1000 team has restored the system to address the problems mentioned above. Among the most significant improvements are the following.

- The CSR level was reduced from three hundred to under fifteen.
- All RT-1000 software was placed into a standard Nortel version-control library and an automated, reproducible loadbuilding process.
- Over 80% of the original nine hundred test cases were automated, reduced load-regression times from around ninety staff-weeks to under five. Formal capacity testing using software simulation packages was introduced to closely parallel field conditions of up to two hundred users (rather than rounding up ten people on a weekend to "stress test" the system manually, as was formerly the case).

- Software maintenance and support contracts were negotiated with the operating system and hardware vendor to guarantee that all operating systems software for delivered systems can only come from the development team, thus ensuring more standard configurations.
- System hardware and the accompanying operating system were upgraded and a long-term hardware migration plan was developed.
- ISO-9001 certification was obtained for the development process.
- Regular customer visits to the Atlanta laboratory were begun in order to build customer relationships.
- The database management system was upgraded to its latest release.
- The system architecture was published and otherwise opened up to customer and third-party enhancements.
- And significant new functionality was added.

The restoration of the RT-1000 system has enabled a much more open and flexible architecture, as illustrated in Figure

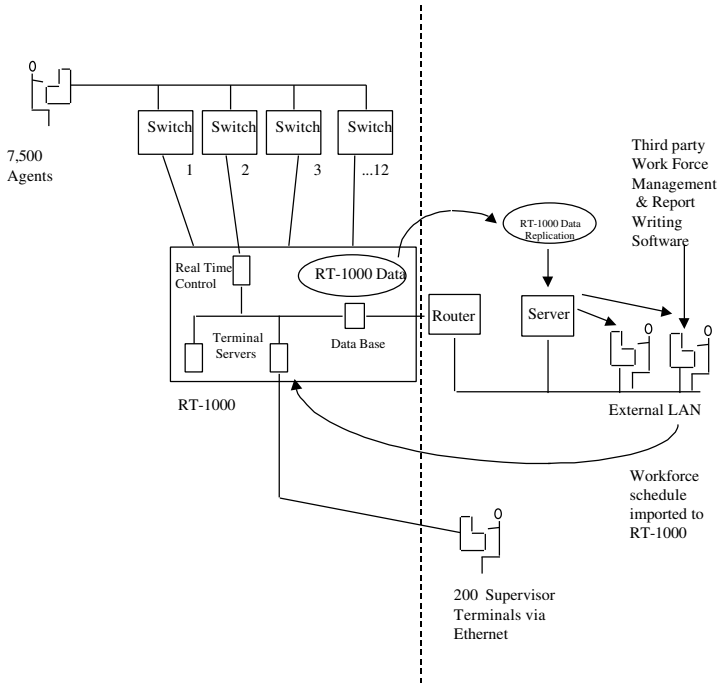

Figure 2: Revised RT-1000 System Configuration

Lessons Learned

Tools

At the start of the project, a study was undertaken to determine what commercial, source-browsing software might be purchased to support the code-understanding process. Eventually, the most sophisticated commercially available tool was purchased. Although it provided quite powerful analysis and browsing capabilities, it could not deal with multiple processes. That is, different processes contained subprograms with the same name and the tool confused these. Moreover, soon after delivery, the providing company went bankrupt. No tool addressing the name collision problem has yet been found.

Another tool-based effort was originally undertaken to support Fortran-to-C conversion. Approximately half of the original RT-1000 code was written in Fortran, but it was felt from both a performance and maintainability perspective that C was a better language in which to develop the product. A freeware toolset was used to support the conversion effort and, while the trial was technically a success, the resulting C-code was so convoluted as to be totally unmaintainable and unmodifyable. As an alternative, the use of configuration scripts (**makefiles**) which allowed C and Fortran code to co-exist in compiled modules was increased. This providers designers a choice of languages in which to work on any given feature or problem.

Currently, the team is initiating the use of an automated, web-based, metrics tool that evaluates the quality of software systems by measuring complexity and maintainability using standard statistical techniques. This tool will be used periodically to assess code releases as they are developed. The product team can thus assure customers that quality is being incorporated throughout the entire development life-cycle. It will also allow more informed decisions as to software architecture modifications in terms of areas which need to be overhauled or eliminated.

Verification

Product verification has been greatly improved. In place of marginally documented single-line descriptions, we now have fully described test cases. We have also added a number of test cases of our own (both for systemic tests and to test features which we have added). We have automated a large percentage of the entire test suite so as to easily compare new versions of the code with past runs to detect GUI or data errors. Finally, we have added true capacity testing to our suite, and are now able to emulate an entire complement of live supervisors, rather than being limited to the number of physical PCs

we could connect to the system in the same way customers do. With a product as complex and flexible as the RT-1000, however, the task of verification is daunting. The more that we learn about rigorous testing, the more we realize that we need to learn.

The Web

At the start of the RT-1000 effort, we knew that the development team would have to learn a lot about the system. But how could this knowledge be effectively shared? An internal hypertext of web pages was used to address this issue. The web has been a real ally in managing this project. We began to use the web as an information repository and retrieval system, and aligned it with our ISO-9001 processes and goals such that it made adherence to the standard a natural part of getting our job done. Also, tools were added to automatically keep updated problem lists for easy reference and for historical tracking.

Project Transfer

Transferring a project of this size across geographic and managerial boundaries took at least a year. In addition, when the type of project is radically different than the one it replaces, large staff turnover can be expected. In such a case, it would likely be better to retain the geographic location and merely hire an all-new staff. This would add some complications, but would reduce others and, in the end, probably be simpler and less costly than moving the project.

Also, higher management needs to offer extra support to a project such as this to reassure those assigned to it that the corporation deems it a worthwhile effort. A lack of corporate support to back up the organizational decisions can further delay and complicate the project.

Organization

The most important problem we had to overcome was the lack of a single point of managerial control. At first, every element of the team (design, verification, technical assistance, and product management) reported upwards through different management structures. When a disagreement developed, there was no way of resolving it quickly. The only practical solution was to have reporting lines converge at a managerial level closer to the working-level groups; otherwise, a very high level manager will need to devote appreciable amounts of time to administer petty details.

CSRs

Customer service requests had been piling up within Nortel prior to the project being farmed out to a third-party company. The customers were told that "the re-write will fix everything!". During the re-write period, problem reports continued to come in, but there was no longer a design group tasked with addressing them within Nortel. When the re-write failed to materialize at all, customers demanded action on their backlog of problems, which had now grown to appreciable lengths.

We took a phased approach to solving the CSRs. Weekly meetings were begun with senior managers (design, product line management, and testing) to discuss just one customer's list. Week-to-week, it was pared down. Also, efforts were made across the board to solve some of the issues independently. Managers also took time to characterize the large pool of issues into the affected areas (database, reports, workforce management, real-time, etc.). The area with the highest customer interest and the most problems was targeted for a software overhaul via features; which addressed some of the CSRs as well. In tandem, the CSR list's duplicate and "non-issue" elements were removed via reviews. During the subsequent release, the next most important set of issues was addressed in parallel with a feature enhancement. Also the next most troublesome customer's list was tackled, once the first was reduced. Over a period of about eighteen months, the list was radically shortened.

Restoration of the Sistine Chapel painting took over thirty years of painstaking effort. The result enables us to fully appreciate one of the most glorious instances of artistic endeavors ever realized by man. Legacy software restoration likewise entails a massive commitment of resources by management and software developers. In both cases, the effort requires careful planning, a well thought out process, deep technical understanding, and a great deal of patience. In the case of RT-1000, up to twenty developers labored over three years to restore the system to full functionality. The resulting increase in customer satisfaction and product revenue has made the effort worthwhile.

References

To learn more about the Sistine Chapel restoration, the reader can visit the following web site: `http://www.christusrex.org/www1/sisteen/0-Tour.html`.

The restoration of the Sistine Chapel is described in the following article. The book in which the article is contained also includes over two hundred and fifty high-fidelity photographs of Michaelangelo's paintings.

Carlo Pietrangeli.
"Introduction / An Account of the Restoration."
in *The Sistine Chapel / A Glorious Restoration,* Pierluigi De Vecch, editor, Harry N. Abrams Publishers, 1992.