Legacy software systems represent a significant investment of time and resources, and provide important services. The restoration of this legacy telephony system generated some lessons of widespread interest to all developers.

Restoring a Legacy: Lessons Learned

Spencer Rugaber, Georgia Institute of Technology
Jim White, Nortel

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—lying on his back much of that time—to complete the task. In 1538, Pope Paul III called him back to add an enhancement, the Last Judgment, over the altar; this took seven more years. Michelangelo's work on the Sistine Chapel is a legacy—a gift from the past that would be impossible to recreate and warrants preserving. In 1965, the Holy See commissioned a team of scientists, historians, and artists to restore the paintings. Though the restoration decision stirred controversy, work proceeded over the next 30 years. Although no software system compares to Michelangelo's masterpieces, the restoration process offers striking parallels to software reengineering.

Software systems can also be legacies—though in this context "legacy" often connotes a burden rather than a gift from the past. The RT-1000 telephony system was originally built by a team of about 70 developers over five years, and cost 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. Hence the decision was made to restore and enhance it instead.

0740-7459/98/$10.00 © 1998 IEEE

Over the last four years, the RT-1000 development team has worked to improve quality and add features, turning a neglected system into a high-quality software product. With one eye looking back at the legacy of Michelangelo's work, we can learn something by examining the factors that enhanced the RT-1000 restoration process as well as those that hindered it.

System Description

RT-1000 is a telephony software system for automated call distribution. An ACD system employs software features that distribute incoming telephone calls 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 system presents the call that has been waiting the longest.

ACD features allow a supervisor to monitor the quality of service 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 capabilities let customers 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, and a typical installation includes 5,000 agents and 150 supervisors.

Figure 1 shows how the legacy RT-1000 system was configured before restoration commenced.

Initial status

Michelangelo's Sistine Chapel paintings cover more than 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 soot from candles burned in the chapel had done an unknown amount of damage.

The RT-1000 has somewhat similar (if less dramatic) history. It was developed by a third-party software vendor in the late 1980s and acquired by Nortel in 1990. For the next three years Nortel enhanced and maintained it before outsourcing it 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 product's six customer organizations were quite unhappy.

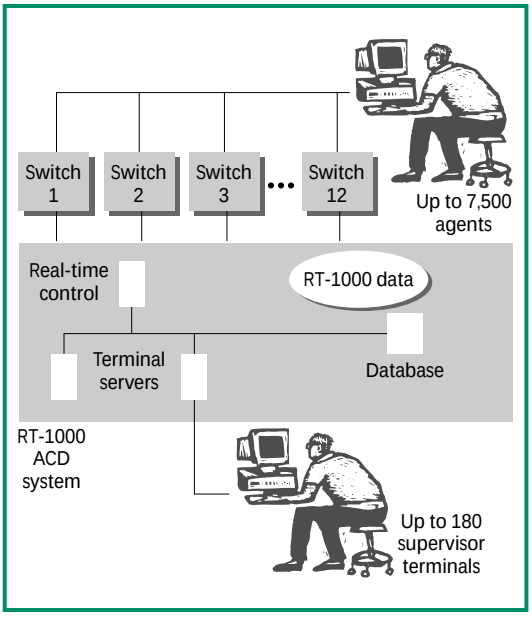


Figure 1. The original RT-1000 system configuration locked customers into outdated hardware, allowed no local area network connectivity, and provided no data transfer protocols to support data archiving.

RT-1000 was assigned to Nortel's Atlanta Technology Park laboratory. No staff members there had any experience with ACD software, and, due to another project's cancellation, staff morale was quite low.

Technical difficulties

RT-1000 is a complex system. Several things made it difficult to deal with:

- it was very large,
- it had a small user base,
- it comprised third-party software and hardware components,
- it lacked formal software process and version control,
- it lacked documentation, and
- it was not Y2K compliant.

Sistine Facts

To learn more about the Sistine Chapel restoration, read "An Account of the Restoration" and other selections in *The Sistine Chapel: A Glorious Restoration* (C. Pietrangeli et al., eds., Harry N. Abrams Publishers, New York, 1994). On the Web, take a virtual tour of the chapel and view restoration efforts at http://www.christusrex.org/www1/sisteen/0-Tour.html.

Size

The system constitutes about 500,000 lines of code that implement 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 system combines multiple architectural styles including real-time (managing up to 50 concurrent processes), MIS, computational, and an event-driven graphical user interface.

Moreover, the RT-1000 must satisfy a long list of difficult, nonfunctional requirements such as data integrity, real-time response, distributed processing, reliability, information security, usability, performance under load, and openness to customer and third-party extensions.

User base

The small user base meant that field trials could be conducted with at most two customers prior to general delivery. And the widely varying ways 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, no one knew exactly how customers used the system. One customer used third-party (custom-designed) software to postprocess reporting data. When the RT-1000 team modified a standard report (to add an extra column for additional precision, as other customers requested), the downstream software could no longer process the report. The customer based its payroll processing on the downstream software output and was therefore highly sensitive to this change, yet the design team often has no knowledge of these off-board systems.

Another customer contracted with a third-party company for display of RT-1000 data on LED-based wallboards in remote offices. When a new screen sequence created a problem with the wallboard display, the wallboard company contacted the development team requesting information, to the complete surprise of the RT-1000 team.

Third-party components

The RT-1000's third-party components were out of date and no longer supported by their vendors. For example, the system included an out-of-date operating system with no vendor support and no means of controlling changes to the field-deployed versions. A commercial vendor's database management system was also out of date. The hardware platform could no longer support customer demands. Customer terminal software was proprietary and failed to conform to the emerging Windows-based industry standard. Finally, the distributed-component interconnectivity mechanism required upgrading the local area network.

Other issues

Software process was nonexistent, as was version control. Product testing was not formalized. And over 300 outstanding customer service requests existed. Little if any documentation of the software architecture existed, and the code itself contained virtually no comments. And, of course, RT-1000 faced a serious Year 2000 exposure.

Restoration Strategy and Lessons Learned

During the first two years, the development team sought to determine its contractual exposures (the classes of deliverables Nortel still owed to its customers), and to categorize its CSRs (the classes of known problems in the system). Armed with this knowledge, we focused on making improvements with the biggest payoff.

For example, we gave priority to a single additional feature that responded to three contractual issues and also addressed several CSRs. Similarly, the two largest classes of CSRs accounted for over half the reported system issues, so we focused on these. As designers incrementally developed expertise in these areas, they were able to address issues with increasing speed. As the major classes were exhausted, repair efforts branched out into other areas.

Restoring the paintings in the Sistine Chapel was also a massive undertaking. The Last Judgment alone required more than a year of preparation, both in a laboratory and experimenting on the painting itself. The restorers had to contend with technical issues that varied across the surface of the painting, and had to develop an elaborate cleaning process including a 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 added over the centuries. Finally, the restorers had to design an entirely new environment for the Chapel, including filtered air and a tailored microclimate.

Likewise, restoring a software system involves far more than updating lines of code. In a sense, it requires constructing a whole new development environment—while continuing to support customers and develop new features.

Current status

Over the past three years, the RT-1000 team has restored the system to address many of the problems mentioned:

- ♦ We reduced CSRs from 300 to less than 15.
- ♦ We placed all RT-1000 software into a standard Nortel version-control library and instituted an automated, reproducible load-building process.
- ♦ We automated over 80 percent of the original 900 test cases, reducing load regression times from around 90 staff-weeks to under five. We introduced formal capacity testing using simulation software to parallel field conditions of up to 200 users (rather than rounding up 10 people on a weekend to stress-test the system manually, as was formerly the case).
- ♦ We negotiated software maintenance and support contracts with the OS and hardware vendors to guarantee that all OS software for delivered systems can come only from the development team, thus ensuring more standard configurations.
- ♦ We upgraded system hardware and the accompanying OS and developed a long-term hardware migration plan.
- ♦ We obtained ISO-9001 certification for the development process.
- ♦ We improved customer relationships by encouraging regular customer visits to the Atlanta lab.
- ♦ We upgraded the database management system to its latest release.

In a sense, restoring a software system requires constructing a whole new development environment.

- ♦ We published the architecture and opened it up to customer and third-party enhancements.
- ♦ We added significant new functionality.

The restoration has enabled a much more open and flexible architecture, as Figure 2 shows.

Tools

At the start of the project, we researched commercial, source-browsing software to support the code-understanding process. Eventually, we purchased the most sophisticated commercially available tool. Although it provided powerful analysis and browsing capabilities, it could not deal with multiple processes: different processes contained subprograms with the same name, and the tool confused these. Moreover, soon after delivery, the providing company went bankrupt. We have yet to find

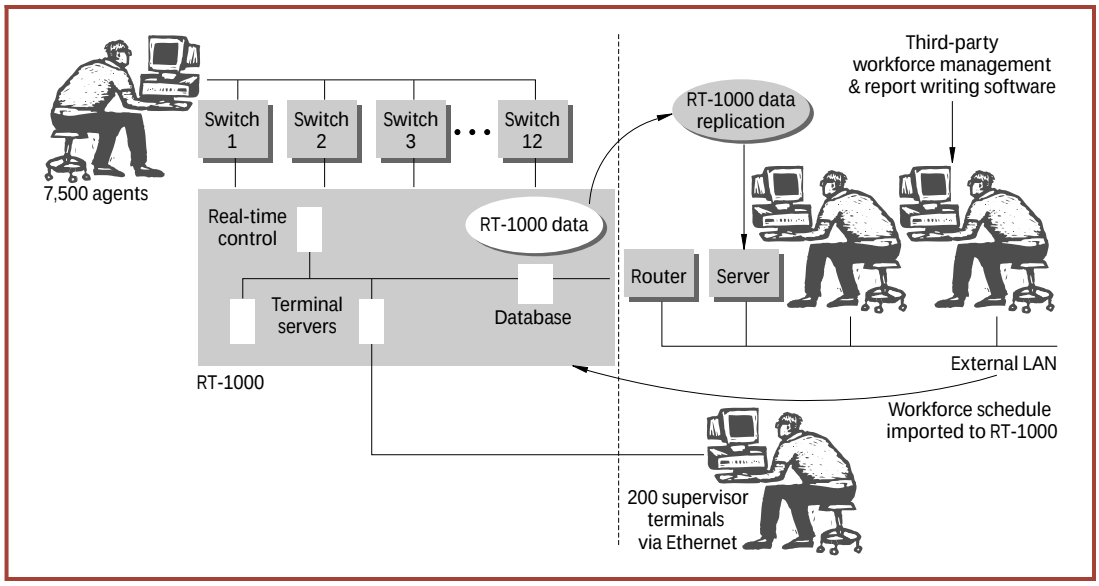


Figure 2. The restored RT-1000 system configuration opened up the system to new hardware, allowed the export of results to third-party report writers, supported the import of customer workforce-management schedules, and provided off-board data replication.

a tool addressing the name collision problem.

We also sought tools to support Fortran-to-C conversion. Approximately half of the original RT-1000 code was written in Fortran, but we felt that C would enhance system performance and be easier to maintain. We used a freeware toolset to support the conversion effort and, while the trial was technically a success, the resulting C code was so convoluted that it was impossible to maintain or modify. As an alternative, we increased the use of configuration scripts (makefiles), which allowed C and Fortran code to coexist in compiled modules. This gave designers a choice of languages for working on any given feature or problem.

We are now using 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 help us periodically assess code releases as they are developed and assure customers that quality is being incorporated throughout the entire development life cycle. The tool will also permit more informed decisions on software architecture modifications by bringing attention to areas that need to be overhauled or eliminated.

Verification

Restoration has greatly improved product verification. In place of marginally documented single-line descriptions, we now have fully described test cases. We have also added a number of our own test cases (both for systemic tests and to test features we have added). We have automated a large part of the 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 can now emulate an entire complement of 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 we learn about rigorous testing, the more we realize we need to learn.

The Web

At the start of the RT-1000 effort, we knew the development team would have to learn a lot about the system. But how could this knowledge be effectively shared? We created an internal hypertext of Web pages to address this issue, and have found the Web to be a real ally in managing this project. We began to use the Web as an information repository and retrieval system, and aligning it with our ISO-9001 processes and goals has made adherence to the standard easy. Also, we added tools to automatically keep updated problem lists for easy reference and historical tracking.

Project transfer

Transferring a project of this size across geographic and managerial boundaries took longer than expected. The first three months included physical transfer and setup of the hardware and software as well as staff training. Interfaces to other groups, such as verification, field support, and product management had to be reestablished. Technical responsibility for the software components had to be par-

titioned and a software maintenance infrastructure built. It took at least a year to gain full productivity.

We found that when the restoration project originates from the outside and differs radically from the original one it replaces, it may cause large staff turnover. In our case, more than half the original staff either left the company or transferred internally. In such a case, it would be better to leave the project at its original geographic location and hire new staff. This would add some complications but 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. But when a disagreement developed, we could not resolve 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 would have had to administer petty details.

CSRs

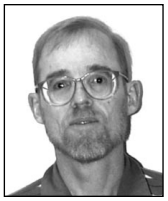
Customer service requests had been piling up within Nortel prior to the project being farmed out to the third-party company, at which time the customers were told that "the rewrite will fix everything!" During the rewrite period, problem reports continued to come in, but there was no longer a design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

We took a phased approach to solving the CSRs. We initiated weekly meetings with senior managers (design, product line management, and testing) to discuss one customer's list that, week to week, we pared down. Also, we made efforts across the board to solve some issues independently. Managers worked to classify the large pool of issues by affected areas (database, reports, workforce management, real-time, and so forth). We targeted the area with the highest customer interest and the most problems for software overhaul via features, which addressed some of the CSRs as well. We also removed the CSR list's duplicate and "non-issue" elements via reviews.

During the subsequent release, we addressed the next most important set of issues with a feature enhancement. We tackled the next most troublesome list once we had reduced the first; within 18 months, we had radically shortened it.

Restoration of the Sistine Chapel painting took over 30 years of painstaking effort. The result enables us to fully appreciate one of the most glorious human artistic endeavors. Legacy software restoration can likewise entail 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, about 20 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. ❖

About the Authors



Spencer Rugaber is a senior research scientist in the College of Computing at the Georgia Institute of Technology. His research interests include software engineering, specifically software evolution and program comprehension. He is currently Principal Investigator in several research projects funded by the National Science Foundation, the Defense Advanced Research Projects Agency, and private industry.

Rugaber received a PhD in computer science from Yale University. He is vice chair of the IEEE Committee on Reverse Engineering and past program co-chair of the Workshop on Program Comprehension.

Jim White has worked at Nortel in Atlanta for more than nine years. His project experience has included developing switching access products and MIS systems, working on both design and management. Prior to joining Nortel, he worked on F-16 flight simulators at General Dynamics.

White received a BSEE and an MSEE from the Georgia Institute of Technology.

Address questions about this article to Rugaber at the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332; spencer@cc.gatech.edu. White may be reached at Nortel, 5555 Windward Parkway, Alpharetta, Georgia, 30004; jwhite@nortel.ca.