Simulation and Modeling Driven Software Development
L. Bernstein
E.H. Burke
W.F. Bauer

Background

The development of the software for the Safeguard Anti-missile Missile System was a major challenge. There were no configuration control guidelines or standard software processes when software work began in 1965. The National Academy of Sciences thought that the software would not work. Only NASA’s software program faced the complexity and performance demands that Safeguard faced. Even though many techniques were borrowed from the Apollo project, a unique approach to software development evolved spontaneously. Many projects modeled and simulated system performance, but Safeguard was the first to use the models and simulations to drive its software development. After some early false starts, the entire software effort fell into lock step with the model and simulation program. This is the story of the effectiveness of this approach.

Bell Laboratories was responsible for the design and development of the Safeguard System from 1967-1977. It was to defend US ICBM silos and to make a first attack on missile silos likely to fail. The intent was to deter an enemy ICBM attack. The successful development program resulted in the SALT II treaty that slowed the arms race by halting deployment of ABM systems. It also demonstrated the wisdom of an extensive simulation program calibrated with actual field data.

Safeguard employed long-range and short-range interceptor missiles. The Missile Site Radar (MSR) performed atmospheric target tracking and defensive missile guidance. The radar used phased array technology to form and steer radar beams at electronic speeds under software control. A computer designed by Bell Laboratories capable of multi-processing as many as ten processors in parallel ran the software. A table-driven operating system achieved efficient parallel performance by capitalizing on the predictable sequence of tasks the Safeguard System used to find, track and intercept enemy re-entry vehicles (RVs).

Shortly after the decision to deploy an ABM system, a System Evaluation Department was formed to provide quality assurance. Its objective was to insure that the design met the system objectives and that the implementation met the system requirements. To carry out that objective the department designed field tests conducted on the Kwajalein Test Range. This effort became so important that it became the driving force for the software development effort. Typical mission scenarios were used to measure software progress throughout the entire development cycle.

The system evaluation team took the approach of developing a family of simulations to predict and confirm system performance. The highest level simulation predicted the performance of the entire system to a full scale attack. To facilitate the design of the simulation, the models of subsystems (missiles and radars) were only as detailed as was required to enable the system simulation to model overall system performance. Detailed simulations of all major subsystems validated the high level models. In some cases, the phenomena modeled in those subsystem simulations were based on even more detailed simulations accounting for the fundamental physics involved.

This approach depended on the validity of the simulations used at all levels of system analysis. Real data taken during sub-system and system tests validated and calibrated the simulations. Extensive simulations were run prior to field tests. This approach predicted system performance, found software errors and eliminated surprises. Before any test, multiple simulations were run to obtain a statistical distribution of predicted performance. Since there were many variables in each test, the precise results could only be predicted statistically. Once the test was conducted and the real data were obtained, the simulation was re-run using the measured data. The simulations reconstructed actual missile flight history and miss distance. Differences were noted and the models were re-calibrated. This method worked well and gave high confidence in simulations of battles against enemy RVs.
Each system test cost between $5m and $10m. Consequently, the test program design maximized meaningful information obtained per test and minimized the number of tests. This resulted in a program that stressed the entire system and also its components in specific areas separately. It validated total system performance. In every case, exhaustive simulations were conducted before the field test so that there was high confidence that it would be successful.

**Significant Results**

The MSR had to track RVs at short range. This was a special challenge to the System Evaluation Department. When an RV warhead re-enters the atmosphere, tremendous heat is generated by the friction between the RV and the increasingly dense atmosphere as the RV descends. This heat ionizes the atmosphere, producing a wake of ionized air trailing behind the RV. The radar pulse is reflected by this ionized wake as well as by the RV itself. Consequently, the radar return is a composite of the "clean" reflection from the RV and an irregular reflection from the extended wake. The task faced by the evaluation team was to model the physics associated with the generation of wake with sufficient fidelity to evaluate the effectiveness of the MSR tracking before tests with real RVs.

The MSR used a software closed-loop feedback system to maintain the range gate (a window in time where the radar would “look” for the target) and the azimuth and elevation of the radar beam pointing at the target. The design of the range gate assumed a “clean” target return. As long as a sufficient gap existed between the “clean” return from the target and the extended irregular return from the wake the algorithm worked well. However, if there were no gap between the two returns, the range gate would drift back onto the wake and the software would lose track of the target.

This modeling was extremely complex. It had to account for the interaction of the radar pulse that employed a range of frequencies to improve range resolution (known as “chirp”) with the reflections from the stationary ionized particles of the wake and the reflections from a very fast-moving RV. The Doppler effect shifts the echo of a chirp pulse from a stationary target away from the echo from a moving target. This shift had a significant impact on the degree of separation, or gap, between the target and wake reflections and consequently was critical to the evaluation of the tracking software.

After months of painstaking effort to model the effect of wake on the MSR tracking scheme, the simulation showed the MSR would consistently lose track during the interval of peak wake. This was unanticipated. Had it not been for the simulation, it would have remained obscure until first system tests on Kwajalilen a year later. This analysis prevented a software crisis. The discovery that the tracking algorithm was flawed resulted in a crash effort to redesign it. A thresholding scheme insensitive to the gap between target and wake reflections was developed. Simulation results were used to tune and test the new algorithm that would maintain track on a waking target.

The new tracking scheme could not be available in time for the initial system tests, which therefore were conducted using the original tracking algorithm. The MSR lost track on the target at the altitude that the simulation predicted. Later tests conducted with the revised tracking scheme were successful, again as predicted by the simulation. The new algorithms were implemented in the operational software. The simulation results were then used to check that the algorithms were implemented correctly. The benefit of the evaluation effort was that the development of the revised tracking algorithm began about a year earlier than it would have if the problem had first surfaced during system tests. This advanced the overall system test program by one year since low altitude target tracking was a prerequisite for most system tests. Being able to predict the problem and having a solution ready before the first system tests preserved credibility of the System Evaluation Department.

The software development schedules were based on the need to meet the field tests. Each capability was carefully defined and detailed software verification tests were run to assure that each capability was available in the next software load. After software was assembled for a field test special software certification tests were run to make sure that the system operated properly for combinations of possible target and interceptor conditions. This approach led to incremental development. Progress in the development of the software modules was tracked by
measuring readiness for field tests. The module development plans and tests themselves reflected field test needs as defined by the system simulation engineers.

The newest leading edge tracking algorithms for the RV were validated in the field. The simulations anticipated the failure of the first tracking test due to errors in the software tracking programs. The developers, to their regret, ignored the simulation results. It took another failure before the software developers took the simulation results into account.

Computer roundoff errors led to a second tracking problem. Early simulations were conducted on a commercial computer, but the computer used in Safeguard was a special purpose computer. Of particular was concern was the algorithms used to track the incoming RVs. The simulations predicted that the tracking algorithms would work well under all of the expected tactical situations. However, system evaluation engineers grew suspicious of the difference in the computers and suspected it would affect the validity of the simulations. The commercial computer represented numbers with 36 bit accuracy, while the special purpose computer used in Safeguard had 32 bit accuracy. After the simulation was modified to model the 32 bit accuracy of the Safeguard computer, roundoff errors emerged. The tracking algorithms required the inversion of a 9-by-9 matrix. This matrix was characterized by very large diagonal terms and very small off-diagonal terms. At first, it appeared the algorithms would be adequate, but that was only because the initial target trajectories came directly at the radar. Only after "fly-by" trajectories were simulated did the discrepancy between the 36 bit and the 32 bit accuracy become apparent. On such trajectories, the off-diagonal terms became important in predicting the target position and, with the 32 bit accuracy, the software would lose track of the RV target as it began to fly by the radar. The simulations showed the need for double precision arithmetic in the matrix inversion operation. This was discovered at the same time that the field test was conducted. The software dropped track of the RV. The software was then changed to conform to the model. The model and simulations became the standard for software performance.

After several years of validation with data from live tests, confidence grew in the ability to predict test outcomes. One parameter predicted was target impact. As the difference between predicted and actual target impact was plotted for ten tests, a bias became apparent in one direction. By now the quality of the simulation and the accuracy of the tactical software in all other respects were highly regarded, so other sources of error were suspected. The location of the Kwajalein Atoll relative to Vandenberg Air Force Base was in error. A resurvey, treated with great skepticism by the contracting officer, showed that the location was off by precisely the bias detected by the simulations.

In another instance, the quality of the simulations allowed us to assure success of a missile test that might otherwise have failed. After intercept, special tests were performed on the interceptor missiles to see how they performed under the greatest stress. An order to the interceptor to turn right as fast as it could was followed in two seconds by an order to make a sharp left turn. This was called the "tail wag" algorithm. It was changed to avoid a tight flight safety boundary. Instead of ordering a "zigzag" (full right command for two seconds followed by a full left command) the opposite, left then right, "zagzig" was ordered. If the interceptor chosen for the field test happened to be faster than average, it had room to avoid the range safety boundary.

Whenever an order to self-destruct was sent to an interceptor missile the missile tracking software continued to track one of its fragments to assure that the fragment would not fall in an inhabited area. The simulation showed that if the sine of an angle became greater than 1, the software would 'abend.' This would not stop the computer because special code was available to handle recovery from abends, but software checks were added to prevent the abend and avoid polluting the mission data.

The field test scenarios were run hundreds of times before each live test with only the random noise generators changing the details of the scenario. These were 'Mission Reliability' tests designed to stress the computer hardware and software before each field test. There was some controversy about the need for the software reliability tests until an operating system race condition was found in the 108th running of one scenario. This latent bug could have stopped data recording during a live field test leading to failure. It proved the need for such software testing.

The results of the simulation runs were compared with tests of the tactical software to ensure software correctness. When there was a difference between the simulation and the
software tests, comparative tests would be run to isolate the problem. This moved debugging from detective work to analysis. The data reduction approaches invented for the simulations were a model for testing the software loads and eventually led to automated analysis for the software reliability tests. When an exceptionally stressful multiple RV and multiple interceptor scenario was run, a design error was found in the software scheduler. It did not show up in the simulators, but caused a system failure when software tasks were improperly dispatched in the Safeguard computer. The simulation data helped to isolate the problem rapidly.

Near the end of the test program the simulations became so trustworthy that a major catastrophic problem was almost overlooked. Designers would tune the range gates size in the tracking algorithm for each field test based on the simulation results. Early tests in the program stressed RV tracking with low altitude intercepts. Those conducted later stressed interceptor guidance with high altitude intercepts. As the altitude of the intercepts increased, designers tightened the range gates to decrease miss distance. In one test the interceptor actually hit the RV! The gates became so small that the software could lose track at the RV at low altitudes thereby invalidating all previous tests. When this problem was found, the size of the range gates were increased and all the tests were re-simulated validating system operation. Because of the quality of the simulation repeating all the field tests could be avoided.

**Lessons Learned**

The system evaluation experience validated the overall approach of predicting system performance by extensive analysis and simulation. Validating the simulations with data from live system tests worked. The approach proved effective in four ways:

First, exhaustive and detailed simulations smoked out requirements flaws and made them easy to fix. There were no program slips due to these flaws.

Second, it minimized the number of tests.

Third, it validated the family of simulations used to model the performance of the tactical Safeguard System under full scale enemy attacks.

Fourth, it produced a set of scenarios used to test the tactical software, find bugs and track software development progress.

This simulation and modeling technique evolved spontaneously during the development of the Safeguard System. It can be used to good effect in the development of any large complex system. It was used successfully in the development of systems used to operate telephone networks. Harlan Mills showed that scenario based testing is thirty times better than classical coverage testing at the University of Florida during the 1980s.

The technique is to model system components and the entire system in a hierarchy of models. Then use the models to systematically simulate system performance with typical operational scenarios. Design the operational software based on functional requirements embedded in the models and run the operational scenarios against the production software. Compare the results of the simulation with those from the operational tests and resolve any differences.

Validating requirements and providing them to developers unambiguously, moving debugging from detective work to comparative analysis and measuring software development progress in terms of completing scenario tests led to a successful Safeguard software development program. These were the most important lessons learned.

**Acknowledgment**

We wish to thank C.M. Yuhas for helping to make this story clear.