A Technology Transfer Retrospective

Roy Levin
Microsoft Research, Silicon Valley
January, 2003 (revised July, 2003)

Many have written about the challenges that industrial research organizations face in trying to transfer the technology they create to other organizations. Research pursues a long and winding road from the proof of concept of a technology in the lab to the adoption of that technology by others and its use for corporate benefit. To follow the road to its end requires persistence, determination, flexibility, and (when, as is often the case, the road ends short of the destination) good humor. In this short paper, I offer a personal recollection of a part of one such journey — one in which the destination reached wasn't the one originally sought.

The Road

My story tells of the Vesta system, the eventual result of an extraordinarily long research activity that spanned more than twenty years and three companies. The focus of this research was software configuration management, especially the problem of building large-scale software systems incrementally and reproducibly. (An *incremental* build is one in which the minimum amount of compiling and linking occurs, exploiting as much as possible the results of previous compile/link steps.) Butler Lampson sparked my initial interest in this topic in the early 1980's at Xerox PARC. At that time, the software environment in which we were working differed significantly from those in general use elsewhere, since it had been constructed around a custom programming language and operating system (both called Cedar). Nevertheless, the overall problems of system-building were largely the same as one would have encountered under Unix or any other programming environment at the time.

Many researchers had investigated tools to build software incrementally, and some commercial systems of the time included them. Perhaps the best known was **make** [1], a simple tool originally built for Unix but subsequently adapted in many other environments. **Make** provided facilities for two essential aspects of system-building: (1) a concise way to express dependencies between components of a software system, and (2) a script of rebuilding actions for each component, to be executed when a predecessor in the dependency relation was updated. **Make** was designed and worked well for systems of a few tens of thousands of source code lines, but its limited notion of dependency did not extend well beyond that. Systems at the next order of magnitude or larger typically required build tools that supported branched development and/or multiple target platforms and/or a geographically dispersed organization. Developers of such systems still wanted to build incrementally — the value of doing so was even greater with large systems — but **make** could not do so reliably. As a result, developers of larger systems had to abandon incremental building and, while they might still use **make** as the mechanism for scripting the build actions, they reverted to "scratch" building which, for large systems, was an overnight

activity conducted by a "release management" organization. As Lampson observed, this was effectively a return to the 1960's, when such systems were built overnight by submitting large card decks as a batch process[1].

This unsatisfactory state of affairs had not gone unnoticed in the research community, and many variants of **make** were developed that sought to address the problem. Mindful of Roger Needham's maxim to do research "with a shovel rather than a tweezers", and unburdened at PARC by existing build processes based on **make**, we embarked on a line of research to rethink software system building from first principles. An early result of this research was the Cedar System Modeler [2], built by Ed Satterthwaite. However, this tool focused less on the problems of scale and incremental construction than on the use of a strong type system to minimize errors in building.

Before the Cedar System Modeler could see any significant use, Lampson and others (including me) left Xerox to found the DEC Systems Research Center (SRC). This group immediately set about creating a programming environment incorporating some of the features to which we had grown accustomed in Cedar. However, while this environment had a custom operating system (Taos [3]) and programming language (Modula-2+ [4]), the software development tools came from Unix and **make** was the system builder. We thus became acquainted first-hand with **make**'s characteristics, and I soon initiated a new project to attack "the system building problem" afresh. The project was named Vesta[2].

The Vesta research project produced a practical system that was deployed at SRC around 1989. It used a modular, functional programming language to express the build "script" and was able to build all of Taos, the Modula-2+ compiler and tools, and hundreds of libraries and applications built on them, all incrementally and reproducibly [5]. This body of code comprised nearly 1.5 million source lines, well beyond what **make** could reliably build incrementally. It was language-independent — that is, programs written in languages other than Modula-2+ could be built by Vesta — and supported both branched and cross-platform development.

The Vesta developers were excited by this successful demonstration of the feasibility of large-scale, incremental, reliable software system building.[3] As a result, we embarked on a series of visits to DEC product organizations that we hoped would embrace the technology. DEC had two substantial programming environment product suites, one based on VMS, one on Unix. Both used **make** or its relatives as their build

---

[1] Those too young to have experienced system construction in the days of batch processing can glean a sense of it, and much more besides, from Frederick Brooks' classic retrospective on software development *The Mythical Man-Month* (Addison-Wesley, New York, 1995).

[2] According to Bulfinch, "Vesta (the Hestia of the Greeks) was a deity presiding over the public and private hearth." That duty struck me as an apt characterization of the role of a configuration management tool.

[3] To be fair, the initial Vesta system was not without problems. Its build language was difficult to use, the builder's performance was quirky, and the whole system's ability to scale was limited, although still much better than **make**'s. Indeed, these problems led us to conduct an internal user study to understand how Vesta might be improved, but that's another story.

engine, and we believed the demonstrable superiority of the Vesta approach would be appealing. The tools purveyed by these groups were DEC products and were also used internally by the VMS and Unix operating system and layered product development groups for their very large code bases.

We returned from these visits sadder but wiser. While these groups found the Vesta technology attractive, they could not adopt it. There were several show-stoppers. For expediency, we had implemented Vesta by exploiting features of the Taos operating system that made it impractical to port Vesta to other platforms. We believed this could be fixed[4], but it nevertheless put off the potential recipients. Furthermore, the whole Vesta system was implemented in Modula-2+, a language unsupported by DEC and unknown to most of its developers. More seriously, Vesta's idiosyncratic build-scripting language, uncertain scalability beyond systems of a few million lines, and inability to support geographically dispersed development made it an inadequate replacement for the **make**-based build systems that the product development organizations had cobbled together. We were disappointed, but went back to the drawing board, and began a new project to address these shortcomings.

The result, several years later, was Vesta 2. While continuing the original research goal, Vesta 2 had different technical objectives and substantially new personnel. Goaded by Bill McKeeman, we recast the syntax of the build-scripting language to resemble C, while retaining the underlying functional semantics that were essential for Vesta's incremental building machinery[5]. We completely redesigned the storage system and language interpreter to accommodate systems of 10 million (or more) source lines and to support geographic distribution of their development. We implemented Vesta 2 in C++ on top of DEC's Tru64 (Unix) operating system and equipped it with Unix-like management tools. The resulting system is described in detail in [6].

By the time that Vesta 2 was completed, DEC had largely ceased to invest in software development tools as part of its product portfolio. Some of the organizations we had previously visited no longer existed, but the operating system groups did, and the Unix organization expressed some interest in Vesta 2. Ultimately, however, they decided not to use Vesta for a combination of reasons, most of which are familiar to researchers who have followed the technology transfer road. Two in particular deserve note:

---

[4] Indeed, by that time, SRC had shifted from Taos to Unix as its research platform and some of our colleagues were encouraging us to reimplement the Taos-dependent parts of Vesta so that they could continue to use it on Unix.

[5] An explanation of the language semantics would go far beyond the scope of this paper. The key idea, however, is that the function calls of interest in a Vesta build script are invocations of tools (e.g., compiler, linker). The arguments to these function calls are all the dependencies (e.g., included files); there are no global variables and, because of the functional language, no side-effects. Consequently, the function calls can be cached, and a cache hit indicates that a tool invocation can be bypassed and the cached result (e.g., compiler or linker output) can be used instead. This is the semantic basis of incremental building in Vesta. For an in-depth discussion, see [6].

- Vesta 2, while technically superior to existing build tools, represented too radical a departure from **make**. To adopt Vesta would require rethinking the entire building methodology of the Unix organization, not to mention the structure and function of its release management group. Despite Vesta 2's evident benefits, the conversion effort and retraining necessary to adopt it were simply too much to consider.

- Vesta 2 came from a research group, not another product group or external vendor. The Unix organization would need long-term assurances of support before adopting the system, and (justifiably) didn't believe that the research organization could provide that assurance.

We could not make headway against these objections. To us it seemed ironic that the operating system organizations periodically revised their build processes, occasionally even building specialized tools to enable them to continue to build their systems from scratch overnight or over a weekend, but they would not consider a systematic rework that could have a major impact on their productivity[6]. We were about to shelve Vesta 2 when we encountered an unexpected bend in the technology transfer road.

An Unexpected Destination

I was sitting in Chuck Thacker's office sometime in 1997 complaining about our inability to find an outlet for the Vesta 2 technology. Chuck reminded me that modern hardware development had become software-intensive and that DEC of course was fundamentally a hardware company. The company was sharply reducing its formerly broad investments in software to focus on its core line of Alpha-based computers. The software involved in development of an Alpha chip was not quite on the scale of an operating system, but it was well beyond what **make** could comfortably handle. Chuck thought Vesta 2 might help.

I realized that I had been wearing blinders. As the Vesta group had considered applications of Vesta and potential organizations for technology transfer, I had always focused on enhancing a conventional C or C++ programming environment. The Vesta group, being software developers ourselves, had never really considered the applicability of our system to hardware development. Moreover, we had generally focused on transferring Vesta technology to a group that already produced software development tools, since we knew that the support of Vesta would have to be assumed by the receiving organization. We didn't expect that a receiving organization would be willing to incur the support cost (or acquire the expertise) for the Vesta system simply in order to use it — our experience with the operating systems groups had taught us that. But we were wrong.

DEC's Alpha division had two teams, each developing a new version of the Alpha processor chip. One of these teams was finishing up its current chip and beginning to prepare for the next one, code-named Araña. The build system they had been using

---

[6] This syndrome was familiar to some of us from our time at Xerox, where analogous events spawned the lament: "There's never time to do it right, but there's always time to do it over."

was based on CVS, RCS, and **make** and had significant operational problems. Matt Reilly, who had responsibility for the development tools that the chip designers would use for Araña, was looking for something better. With a colleague, Walker Anderson, he created a list of desiderata, then Walker prepared a comparative analysis of some potential replacement tool suites, including Vesta 2. After some stress testing showed that Vesta 2 could meet Arana's needs, Matt initiated a series of exploratory meetings with us. In the course of these discussions, we revisited all the issues that had prevented the transfer of Vesta 2 to other DEC organizations. Many were significant, but none proved to be show-stoppers. What was different this time?

- Because the Araña designers were beginning a new chip, they had the opportunity to take a fresh look at their development environment and revise or revamp it. Development of a modern CPU chip is a multi-year task involving hundreds of people, so an investment in new tools that will improve the process and resulting product merits serious consideration. Thus, Vesta 2 arrived on the scene at a propitious moment.

- While some of the basic development tools carry over from one generation of chip design to the next, many need to change to reflect advances in the underlying process technology. Moreover, little of the previous design (expressed as software) carries over; there is, in effect, a new "code base" with no legacy code. This stands in sharp contrast to the situation in the operating system groups, which have an ever-growing legacy code base.

- Despite their interest in Vesta, the Araña group could not risk wholesale introduction of a new system, with the attendant training and inevitable adoption problems involved. But, in part because they were getting a fresh start, they could structure their development to introduce Vesta in a small subgroup (about 20 engineers) first, fitting the outputs of that group into those of the rest of the organization, which continued to use older build processes. Over time, as they developed confidence in Vesta 2, they could scale up its use by introducing it to additional subgroups.

- Matt Reilly found Vesta 2's functionality (incremental build, scalability, reproducibility, parallel builds, branched development) sufficiently compelling that he was prepared to lobby his organization to commit an engineer, Ken Schalk, to become their local Vesta expert. Ken understood the needs of the Araña developers far better than we did, and could both convey problems back to us and help the Araña developers to use their new system-building tool to maximum effect[7].

- Because the Araña group committed to taking on Vesta 2 maintenance eventually, the Vesta researchers could agree to support the Araña group until they could "go it alone". By contrast, the operating system

---

[7] Ken became intimately familiar with the Vesta 2 implementation and eventually became the primary support engineer for the system on-site. In fact, he ultimately took overall responsibility for porting Vesta 2 to Linux and making an open-source version available. See www.vestasys.org .

organizations were looking for a customer/vendor relationship, which a tiny research group could not provide. An atmosphere of mutual commitment between the Vesta and Araña groups was thereby established from the outset.

The transfer of Vesta 2 technology thus began. The Vesta implementers (Allan Heydon, Tim Mann, and Yuan Yu) worked closely with and through Matt and Ken to provide training and support, which was occasionally challenging because the Araña group was in Massachusetts and the Vesta group was in California. The groups took advantage of Vesta's support for geographically dispersed organizations, using it to exchange updates between their sites and with a small remote branch of the Araña group (also in California). This worked smoothly, enabling fast and orderly response by the Vesta implementers to problems the Araña group uncovered and thereby delivering on the support commitment required to make the technology transfer succeed.

Gradually, the daily involvement of the Vesta implementers decreased; within a year the Araña team had become essentially self-sufficient. By this time the user base had grown from an initial cadre of about 20 to over 130, and a large fraction of the Araña tools and code had come under Vesta 2's management. By the time that the Alpha business was sold by Compaq (which acquired DEC in 1998) to Intel, the Araña team had come to depend on Vesta 2 and was even using it to build software outside the scope of their original plan. They obtained permission for Vesta to be released under an open source license before they left Compaq, and the system went to Intel with them. We had reached the end of our technology-transfer road, though the destination turned out to be an unexpected one.

Lessons

Our repeated attempts to transfer Vesta technology, and our eventual success, lead me to draw the following lessons.

- Successful technology transfer depends on finding a window of opportunity. Candidate recipient organizations have development cycles and, during most of a cycle, they cannot absorb new technology. In our case, the window was the "clean point" between Alpha chip generations, across which little code and few tools are carried forward. Only when the window is open is the development organization receptive; when the window is closed, they can't hear the researchers, no matter how loudly they shout. We found the window open largely by accident. If I had it to do over again, I certainly would seek to understand the development organization's schedule well enough to respond if/when the window opens.

- Appearances matter. Researchers often look for intellectual or aesthetic purity and ignore ugly details that are conceptually straightforward to clean up[8]. By contrast, development groups want things that work, and therefore

[8] This is not a character flaw. Rather, it is an often necessary aspect of getting research done with a small team – non-essential corners should and must be cut. Nevertheless, what gets the research

they care about the details. Those details tell them how carefully the researchers have thought about their needs, which amounts to a litmus test of the practicality of the system under consideration. So, the lesson for researchers seeking to transfer a software system is: remove the twigs over which the developers will otherwise trip. In Vesta 1, the language syntax repeatedly tripped up potential adopters. We resisted, essentially on aesthetic grounds, marrying C syntax with functional language semantics. When we finally did so, we removed a place to stumble[9].

- Having a champion within the candidate receiving organization is essential. Matt Reilly and Ken Schalk were our champions. The old adage that "you can't push on a rope" applies; without pull from the technology recipients, the transfer will fail. Some believe that successful technology transfer requires people transfer. I don't subscribe to this view — Vesta 2 is a counter-example — but I do believe that technology transfer requires a champion, who pulls on the rope. An influential champion is especially important when a methodological change is involved, as with Vesta, because that change must be embraced and promulgated by management.

- Commitment by the research group to make the transfer succeed is equally essential. As Allan Heydon put it, "While you can't push on a rope, if the other side pulls and you're not holding on, things won't go very well either." Supporting technology transfer can be very time-consuming; the Vesta 2 implementers each spent the better part of a year supporting the Araña group. (This is the alternative to people-transfer.) Therefore, both researchers and their management must believe this is time well-spent.

- When the technology transfer requires a substantial change in thinking or operation, success depends on finding a small, somewhat separable group as the point of introduction. Even the forward-thinking Araña group couldn't swallow Vesta 2 all at once; they had to adopt it incrementally. Success is contagious, and once the initial group has had a successful adoption experience, they then become champions for the new technology within the rest of their organization.

- Technology transfer must take bounded time; there must be a plan for making the recipient organization self-sufficient. This generally means that either the receiving organization or some other non-research group commits to ongoing support of the technology. In our case, it was the former, in the person of Ken Schalk.

---

done faster can be an impediment to subsequent technology transfer, and researchers need to recognize the trade-off.

[9] Matt Reilly confirmed that the old Vesta language syntax would have been a significant impediment, giving the Araña developers one more new thing to learn. By hiding the functional semantics under C syntax, we removed that impediment and enabled many developers to read the standard build scripts without being immediately aware of the non-C semantics. Going even further, Ken Schalk created user-interface tools that made it possible for most Araña developers to manipulate build scripts without having to write in the scripting language at all!

None of these lessons is particularly earth-shaking. Some have been noted by others, and no doubt other travelers on the technology-transfer road have encountered them along the way. However, if in recording the Vesta 2 experience I have helped to straighten the road for some future researcher, I will be well satisfied.

References

[1] S. I. Feldman. **Make — A Program for Maintaining Computer Programs.** *Software Practice and Experience*, vol. 9 #4, 1979, pp.255–265.

[2] Butler W. Lampson and Eric E. Schmidt. **Practical Use of a Polymorphic Applicative Language.** *Proceedings of the Tenth Annual ACM Symposium on Principles of Programming Languages (POPL)*, 1983, pp. 237–255.

[3] Paul McJones and Garret F. Swart. *Evolving the UNIX system interface to support multithreaded programs.* Research Report 21, Systems Research Center, Digital Equipment Corporation, September, 1987. Available as `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-21.html`

[4] Paul Rover, Roy Levin, John Wick. *On Extending Modula-2 for building large, integrated systems.* Research Report 3, Systems Research Center, Digital Equipment Corporation, January, 1985. Available as `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-3.html`

[5] Roy Levin and Paul R. McJones. *The Vesta Approach to Precise Configuration of Large Software Systems.* Research Report 105, Systems Research Center, Digital Equipment Corporation, June, 1993. Available as `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-105.html`

[6] Allan Heydon, Roy Levin, Timothy Mann, Yuan Yu. *The Vesta Software Configuration Management System.* Research Report 177, Systems Research Center, Compaq Computer Corporation, January, 2002. Available as `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-177.html`