Developing Mobile Browsers in a Product Line

Ari Jaaksi, *Nokia*

Developers at Nokia recently initiated and used a product line to create and deliver mobile browser products. They learned that, to succeed, a software product line must be product and application driven, rather than reuse or platform driven.

A software product line is a set of systems sharing a common, managed suite of features that satisfy a particular market or mission's needs and that are developed in a prescribed way.[1,2] A software product line basically consists of a family of interrelated software products—applications that use a pool of core assets, such as software components, documents, processes, and tools.

The organization developing a product line is inseparable from the systems themselves: Satisfying a particular market segment's specific needs and operating in a prescribed way require solid requirements and operational management. A product line organization must therefore be a complete product creation and delivery entity that collects and analyzes requirements, creates the products, delivers them to customers or distributors, and supports them after delivery. It functions under a management that has full responsibility for maintaining the core assets and creating and delivering software products.

At Nokia, we used a product line to develop mobile browser products that let mobile phone or personal digital assistant users access services over wireless telecommunications networks. We developed the technology first for our own handsets and later distributed it as a software product. We had two reasons to initiate the product line. First, we needed to serve an increasingly heterogeneous customer base. Second, we wanted to benefit from large-scale reuse. Neither of these reasons alone would have justified initiating a new product line. However, the product line approach provided us with tools to achieve both of these goals and was therefore well justified. Our experience at Nokia can serve as a case study for other organizations debating a similar transition.

Mobile browsers and tools

In 1999, we launched the Nokia Browsers and Tools product line. The technology was initially based on the Wireless Application Protocol specification, which specifies, among other things, mark-up and scripting languages and a communication protocol.[3]

We started by developing a WAP browser and toolkit (see Figure 1). We then extended our product family to include three browser products and a multimode toolkit product. We named the generic version of the browser

0740-7459/02/$17.00 © 2002 IEEE

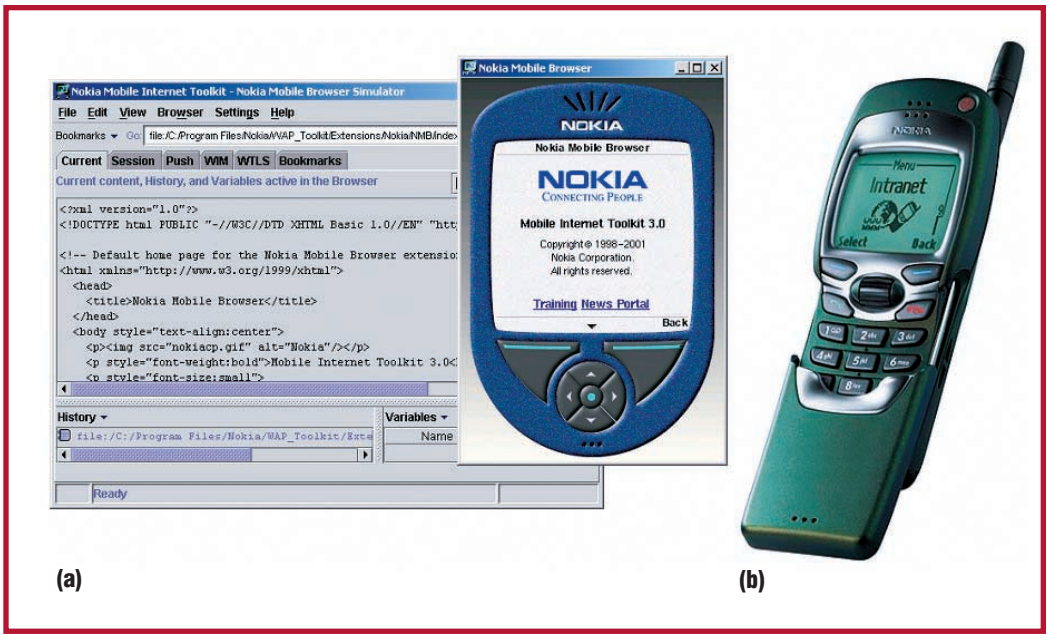

Figure 1. (a) The Nokia Mobile Internet Toolkit and (b) an early browser phone.

Nokia Mobile Browser. We offered it as a software product to external customers seeking a portable mobile browser, and we delivered it as a source code product with a reference implementation on Windows. The Nokia operating system version of the browser included Nokia-platform-specific adaptations and interfaces. We delivered it internally and ran it on Nokia's proprietary phone platforms. We tailored the Symbian OS version of the browser for the Symbian operating system with C++ wrappers and Symbian extensions. We also delivered it as a source code product. The Nokia Mobile Internet Toolkit was a phone simulator coupled with a development environment running on Windows platforms. We distributed it over the www.nokia.com Web site for content developers. By the end of 2001, we had several customers for all the embedded browsers, and the toolkit had approximately 500,000 registered users.

We had many common requirements for all the products. These included requirements for implementing the Extensible Hypertext Markup Language (XHTML) with Cascading Style Sheets and the Wireless Markup Language 1.x browser, supporting both languages natively.[3,4] All products required connectivity through the specified protocol, and we tested all of them for interoperability with wireless gateways.

Several requirements varied from one product to the next. A small phone, such as the Nokia 6210, requires a compact C implementation with small memory consumption. A more advanced device, such as the Nokia 7650, requires C++ interfaces, integration with various personal information management applications, and other more advanced features. Our external customers, such as AOL, Samsung, and Symbian, needed a compact implementation with no dependencies on Nokia's proprietary phone platforms. The toolkit required accurate duplication of phone behavior on the Windows platform, a separate content development environment, and hooks to plug in to various third-party components.

In spite of their differences, all the products used a common product line core, which included elements such as protocol stacks, markup and scripting handling components, and layout managers. We implemented the core in C, and it consisted of approximately 300,000 lines of code.

Developing the product line

A product line must benefit customers by helping the software organization build better products. Therefore, we first concentrated on the specification and speedy delivery of individual products. We built the product line only after our first product releases, which helped us get the first products out quickly.

As an example, we needed to get phone simulators and authoring tools out to con-

tent developers before browser phones were available. We implemented a browser simulator from scratch, using Java for speedy development instead of the existing C-based browser components. In a later release, we replaced the Java-based simulators with C-based simulators that originated from the product line.

The origin of Nokia's product line assets

In 1998, the US Nokia team started to develop mobile browser modules such as a script engine and WML markup handling routines for the Nokia 7110, the first WAP browser phone. There were no requirements to make code available on any other platform or to document or packetize it as a software product. The group simply developed software components to a specific hardware device.

We also started to develop the Nokia WAP Toolkit—a WAP simulation and content development environment—which we released on 17 December 1998, only four months after the project's initiation. The product included full WML and WMLScript development facilities with a WAP phone simulation.

Simultaneously, another team developed WAP gateway products, creating a client-side protocol stack to test gateway protocols. The development happened at Nokia Hungary, with several components coming from other Nokia groups. The protocol stack was not tailored for small devices nor designed for use as part of a software product.

A team in Denmark implemented and adapted browser technology to specific Nokia phone models. The team had another implementation of the WAP protocol stack running in a phone that was small but designed to work only on Nokia's own platform.

Thus, we had many components available for the product line: several core browser components, the WAP Toolkit developed as an authoring tool and simulator, a WAP protocol implementation built as a WAP test suite, and a Nokia platform-dependent protocol stack developed to run only in Nokia phones. We developed and maintained all these components separately. Clearly, it was time for a product line.

Incorporating existing components

In April 1999, we decided to integrate available browser components, put the browser on different Nokia phones, and offer it to external customers such as device manufacturers and telecom operators. This was the first time Nokia licensed out its own phone software—we had already distributed the WAP Toolkit. Now that we needed to develop two different products with a lot of overlapping functionality, we decided to organize the common parts into a product line. Thus began the Nokia Browsers and Tools product line.

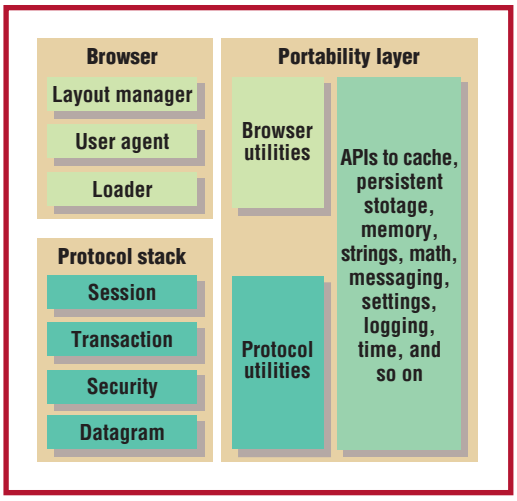

Figure 2. The Nokia product line architecture.

We selected the core browser components and the test protocol stack to form the initial product line. We first enhanced these components to meet our key product requirements of full WAP implementation, portability, and product quality. To meet the implementation requirement, we set up a project to build missing functionality. We also set up an interoperability test suite to verify that we had implemented the standard as specified. To meet the portability requirement, we removed all dependencies to Nokia phone platforms and used only ANSI C and a limited set of standard libraries. We modified all components, especially the protocol stack, to be suitable for small portable devices. Finally, we isolated platform-dependent code and APIs into a portability layer (see Figure 2). At this time, we kept the toolkit separate and didn't use common software components in it. However, the toolkit benefited from the product line's requirements, testing facilities, documentation, and software development processes.

We analyzed the two available stack implementations: the generic test suite implementation and the Nokia platform-specific

We cleaned, tested, verified, and documented the acquired components to achieve product quality.

compact protocol stack. We chose the generic one. The effort of making platform specific code generic and portable seemed to exceed the work needed to tailor an existing generic stack to meet a small device's requirements.

We cleaned, tested, verified, and documented the acquired components to achieve product quality. We concluded that inconsistent error handling, memory management, and coding standards were our most critical problems. We set up a separate maintenance project to clean the memory management, which was all we had time for.

To achieve product quality, we needed first to assess existing quality. We needed the information to focus our development efforts and to set customer expectations right. We constructed test suites, trained personnel, contacted other vendors for interoperability testing, and set up rules and procedures for testing the product line.

We also created customer documentation, which was a major effort: none of the original product line components except the toolkit had any customer documentation. To give an idea of the task's difficulty, the porting guide for the Nokia Mobile Browser consists of almost 500 pages, and the toolkit documentation exceeds 300 pages.

After eight months of cleanup, implementation, testing, and documentation, we released the 1.0 version of the generic browser product from the product line. We now had a pool of core assets on which to build our future products. The fact that most of our core assets were already being used in existing products demonstrated that our technology worked.

Organization and processes

As I mentioned earlier, the technical product line architecture and the developing organization's structure must resemble each other. We had five separate functional entities in our product line organization:

- *Product management* collected, analyzed, and prioritized product requirements; created requirements specifications and roadmaps; and oversaw the functionality of all products.
- *Product development* owned the architecture and implementation resources and ran the development projects.
- *System testing* tested and released the final products.
- *Customer support* assisted customers in using our products.
- *Advanced development* ran the standardization work and provided architectural studies, prototypes, and demonstrations of new technologies and features.

Development took place in the US, but we developed parts of the product line in Finland and Hungary. By the end of 2001, the product line organization employed over 100 people. We released approximately six major product releases each year.

Requirements analysis in product management

During the analysis phase, a product manager responsible for a specific version of a product would create a requirement specification document that analyzed and prioritized requirements for the product release in question. However, this document didn't specify requirements for the whole product line.

During the first two product projects, we tried to allocate product requirements directly to the product line components, which we soon discovered to be difficult. Our customers wanted to discuss features and products, not components. Also, our product management didn't have the technical expertise to allocate product requirements to technical components. Thus, we started to map requirements directly to product releases. Further allocation to separate components happened later in the design phase.

Product managers maintained roadmaps that outlined the characteristics of individual products for the next few years. The product board—composed of participants from all functions of the product line and from major customers—reviewed all the roadmaps together. Such review meetings synchronized different product features: if a feature appeared first in one product's roadmap and then later in others, synchronizing our products provided a means of domain analysis over all the products we developed.

Software creation in product development

Our product development organization

had four teams—namely, the core team, the platforms team, the toolkit team, and the testing team. The teams were competency centers with a certain technical focus, and they allocated their resources to various product development projects.

The core team was further divided into the browser and protocol teams. These teams maintained the core product line—the reusable assets that all the products used. The platforms team specialized in porting the core product to the various hardware and software platforms. The toolkit team developed the toolkit product. The testing team maintained the tools and expertise to test the various products developed from the product line.

Product managers defined a product release's requirements in a requirement specification. The engineering teams analyzed this specification and extracted generic functionality into the product line core, with the goal being to make the core as big as possible. Then, the core team members within the projects handled the common core parts while platform and toolkit teams handled platform- and customer-specific parts within development projects.

The core team had an architect responsible for the product line's architecture. She used the requirements specifications, her expertise, and the platforms and toolkit expertise of the other teams to create a functional specification document. This document was an engineering view of the requirements and specified how to develop products and reusable components. Thus, the core team maintained the product line architecture.

We had only a few projects in which to develop product line components independently from any product release. Typically, we allocated product line maintenance work inside individual product projects, prioritizing component development efforts and embedding them into our tightly scheduled product development projects.[5] This ensured strong product focus but made caring for the product line architecture a challenge.

Our design and implementation phases of the development projects included several cycles that resembled other software projects at Nokia.[5,6] First, each new product release built on the previous one. Second, we built individual product releases incrementally. Our internal customer often wanted intermediate releases during the development project. Finally, we used daily builds,[7] which ensured that one project modifying the product line would not harm another concurrent project using the same components. Daily builds also let us detect problems early.

Developing products from a product line calls for solid configuration management. We stored all the phase products such as documents, test cases, and code in a centralized configuration management system. We shared the system with our internal customers who needed early access to the configuration items.

A test lead developed the initial test plan based on the functional specification. We conducted system and interoperability testing as part of the product projects—different products shared test suites and cases. We also invested a lot in the implementation of reusable test suites. They formed an important part of the product line's core assets and let us reuse test cases, tools, and expertise.

We originally planned to test software components independently and then assemble product releases from these tested components, but we soon realized that this was not enough. Every product needed full testing on different operating systems, with different applications and memory constraints. We never trusted that a component tested in the context of one product would automatically work in the context of another. Instead, we tested all components separately for each delivered product version, using a regression testing approach.

A project board run by the product line management oversaw all the development projects. It reviewed all projects on a regular basis to ensure timely and synchronized project execution, solve resource conflicts between projects, and assign resources to work on common tasks outside the product projects. The board was a key means of maintaining a coherent product line view within individual development projects.

Innovation and standardization in advanced development

Developing products for emerging markets calls for innovation and standardization. We had a dedicated advanced development team that demonstrated new technologies and provided information and ideas to the

Developing products from a product line calls for solid configuration management.

Our XHTML browsing feature is a good example of successful innovation and standardization.

other teams. The team also provided requirements and expertise on emerging standards to the rest of the organization and worked within standardization bodies, such as the WAP forum, to drive standards.

Our XHTML browsing feature is a good example of successful innovation and standardization. To the best of our knowledge, our product line was the first to provide a mobile XHTML browser with Cascading Style Sheet and full WAP 1.x support. Our advanced development team created the basic architecture and demonstrated an XHTML browser for a phone at the end of 2000. The demo provided us with early customer feedback and verified that the implementation was possible within the given memory and computing restrictions. The team also worked closely with the product development that ramped up the XHTML browser's development and released the Nokia Mobile Browser 3.0 in summer 2001. Finally, the team got the core of our XHTML approach approved as the new WML 2.0 standard.[8]

Benefits versus cost

A product line increases quality, shortens time to market, and helps specify the "right" product features. Strong process structures help manage complexity and conflicts between the requirements of individual products and projects. Skilled architects and architectural guidelines can mitigate challenges in architectural development.

Benefits

We experienced increased efficiency through reuse as well as accelerated product implementation, lowered cost, and increased quality. We could not have developed four different products and six annual releases with our given resources without a product line.

We analyzed requirements originating from one product in the context of all other products and used this accumulated domain understanding to benefit the whole product line. Such a process helped us identify new requirements early and let one product benefit from the requirements of others. This also helped us detail feature sets and build competitive products. Understanding the domain helped us specify new products accurately.

The product line provided a clear focus and set the management, processes, tools, and other support elements, which attracted and motivated high-quality personnel. Our product line was highly rated in our annual "Nokia Listening to You" questionnaires in which employees rate their working conditions. We also managed to retain our employees. In 1999 and 2000, when various startup companies offered potentially larger benefits than we did, we lost less than 3 percent of our workforce. That we retained our key experts and team leaders further increased productivity and motivation.

The product line also increased our credibility and demonstrated our long-term commitments. Showing different products developed from the same product line running on different platforms provided a convincing demonstration of our technology. Within Nokia, we became not only the source of browsers and tools but also a center of expertise on XML, XHTML, scripting, and other related technologies.

Costs and risks

Compared to developing a single product, the product line required extra resources in process and tools development, core development, and product management. To minimize costs, we avoided developing core assets that we would not reuse. For example, we considered a potentially reusable library with no need for reuse as a waste. It becomes an inventory that does not benefit the product line and constrains the organization's throughput.[9]

We occasionally witnessed cases where the creation of core assets significantly delayed the shipment of the first products to customers. We considered this to be one of the biggest risks and prioritized getting products out quickly—even if such a move delayed core asset creation. Therefore, we first built product releases and then extracted components from them. Only when a clear case for reuse became apparent did we incorporate the asset into the core.

We had challenges in developing a coherent product line architecture. Such architecture cannot be optimized for a single product but must support the conflicting requirements of many products. Our architecture managed to accommodate different product needs and integrate components originally developed for different architectures. These

sources provided us with a head start but compromised the overall architecture by introducing several different designs and coding styles within the product line. This was visible in the context of our generic browser, which we distributed as a source code product. Our customers had to learn different styles depending on the component they were integrating or modifying.

Our product line supported four different products by providing them with reusable core assets, so modifying these assets could potentially affect all products. We managed such dependencies through controlled requirements management, project synchronization, configuration management, and architecture development. We expected all teams to follow the documented software development life cycle, which implemented several milestones with approval points. Our project board synchronized projects, and our product board synchronized product roadmaps and features. Our architect was a gatekeeper, approving all design and API changes. We also used strict software configuration management processes and tools. Changing plans, components, or APIs required approvals to be acquired and communicated appropriately. All these practices aimed at good quality in predictable product projects at the expense of increased bureaucracy.

We also experienced some conflicts between project teams. Our employees received incentives based on releasing their products on time, yet their work depended on the product line. We had cases in which a project team felt that it didn't get adequate support from the product line, which made blaming others for missed delivery dates easy. We first handled these conflicts through our project board; later, we added product line maintenance into our bonus schemes. However, we never quite managed to solve conflicts between teams relying on each other's work. It seems such conflicts are built into product lines.

A product line can also cause conflicts between customers. They do not always welcome the idea of our developing a product line instead of just their product. As an example, a major customer approved a requirements specification for the next release of its browser product. Later, we showed the client a functional specification and project plans that allocated work to product line components. The client questioned our approach, because it wanted us to concentrate on its product only. Explaining a product line's benefits to a customer of a single product was sometimes difficult.

Moreover, testing a product line is more complex than testing a single software product. We must test the product line in its various configurations, which easily multiplies the number of test cases. To manage this complexity, we system tested individual product releases instead of testing the whole product line. This kept testing simple and guaranteed the quality of product releases. However, we did not build a pool of tested core assets ready for integration, which probably increased the need to test each product release. We had approximately 25 percent of the product line personnel working on testing teams, so the testing phase took approximately 30 percent of the entire time of developing a software product release.

Testing a product line is more complex than testing a single software product.

For Nokia, the product line approach's benefits clearly outnumbered its costs. We reused our core assets extensively, and we succeeded in serving a diverse customer base, from content developers to device manufacturers and software companies. Such experiences are motivating us to launch new product lines and convert existing activities to use the product line approach. An important future challenge—not just for Nokia—is to learn to reengineer existing product lines to accommodate new business, technical, and customer requirements.

The product line increased costs in various support functions, architectural work, and management. We therefore suggest that an organization consider initiating a product line only when it both aims at systematic reuse and serves a heterogeneous customer base with a common domain. Based on our experiences, you can achieve fairly high levels of reuse without developing and maintaining an entire product line—for example, by using independent code components.[5] You might better serve a homogeneous customer base by adapting a single product for different purposes. Moreover, without a common domain, the pool of core assets

About the Author

Ari Jaaksi heads the Nokia Mobile Phones software and protocols research organization in Finland. His research interests include development methodologies, architectures, and software development organizations. He received his PhD in software engineering from Tampere University of Technology. Contact him at Nokia Mobile Phones, PO Box 1000, 33721 Tampere, Finland; ari.jaaksi@nokia.com.

might not grow big enough to be beneficial.

We believe that a software development organization should resemble the development view to the software architecture.[6,10] The processes and organizational structures must support the product line architecture. Components must have owners, deliverables must be allocated to teams and projects to develop them, and the interaction between teams must resemble the interaction of the software components in a product line. Thus, coupling the product line and the organization is important. They need to develop hand in hand.

Building a product line is a long-term effort in which the benefits come through reuse, which can only come after several product releases. We believe that building a few products first is the right way to initiate a product line. Early products provide customer feedback and prevent the construction of useless assets. A product line needs a long-term management commitment together with skills to build it step by step. It is an investment that can be made if business conditions call for a long-term commitment to deliver several products sharing a common domain.

Acknowledgments

I thank my colleagues Raouf Bortcosh, Greg Carpenter, and Konstantinos Kalabokis at Nokia for building and managing the product line with me and helping me write this article. I also thank Ilkka Haikala and Kai Koskimies at Tampere University of Technology for their valuable feedback.

References

1. L. Bass et al., *Product Line Practice Workshop Report*, CMU/SEI-97-TR-003, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1997; www.sei.cmu.edu/publications/documents/97.reports/97tr003/97tr003abstract.html.
2. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, Reading, Mass., 2002.
3. Wireless Application Forum, *Official Wireless Application Protocol: The Complete Standard*, John Wiley & Sons, New York, 1999.
4. World Wide Web Consortium, *XHTML 1.0: The Extensible HyperText Markup Language*, 2000; www.w3.org/TR/xhtml1.
5. M. Laitkorpi and A. Jaaksi, "Extending the Object-Oriented Software Process with Component-Oriented Design," *J. Object-Oriented Programming*, vol. 12, no. 1, Mar./Apr. 1999, pp. 41–50.
6. A. Jaaksi et al., "Tried & True Object Development," *Industry-Proven Approaches with UML*, Cambridge Univ. Press, Cambridge, UK, 1999.
7. S. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, Redmond, Wash., 1996.
8. WAP Forum, *Wireless Application Protocol, WAP 2.0: Technical White Paper*, Jan. 2002; www.wapforum.org/what/WAPWhite_Paper1.pdf.
9. E. Goldratt, *The Goal—A Process of Ongoing Improvement*, North River Press, Great Barrington, Mass., 1992.
10. P.B. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, no. 6, Nov./Dec. 1995, pp. 42–50.