

TIPS & TECHNIQUES

What Everyone Needs to Know About SOA: Diving Into "The Next Big Thing" while reusing Legacy Software

By Curt Finch

Service-Oriented Architecture (SOA) is a style of information systems architecture that enables the creation of applications that are built by combining loosely-coupled and interoperable services. These services inter-operate based on a contract that is independent of the underlying platform and programming language
- *Wikipedia*

I've been writing software for a long time.

When I was 14, I got an HP programmable calculator that I taught to tell time by flashing hours, minutes and seconds repeatedly. It used a loop of null operations to take up the time between each second. Slight changes in room temperature would cause it to speed up or slow down the rate at which it performed these loops, ultimately making it a poor timepiece.

But I was hooked – I have always loved programming since those days.

That was almost three decades ago and since then, I've been writing some kind of software almost every day or managing those who do. For the last decade, I've been running a company that creates and sells software.

In all that time, there has been a recurring theme in our industry that keeps asserting its head over and over again: reusing already battle-tested software is preferable to writing new software. How can we do a better job of reusing legacy software?

Reusability – The Holy Grail

That same wonderful HP company that got me started on this path (and is now one of my customers) put out a great book in 1992 with the obtuse title, "Practical Software Metrics for Project Management and Process Improvement". I know it sounds like a snoozer, but it's not. Although this book is more than 14 years old, it could have been written this year because the concepts are still relevant. Among the research are these two nuggets of wisdom:

1. *Projects created primarily from reused software experience only about 1/3 the defect density of those that are new.*

- 2. Projects created primarily from reused software take about 1/4 the time and resources of those that are new.*

There are many other nuggets. It's a great book.

Wires

The first computers were programmed by electrical engineers moving jumper cables around in the back of a giant mainframe. This is how the concept of a 'bug' in software got started. A bug crawled into some of these wires, shorted a couple of them, and changed the logic of a program. (http://www.jamesshuggins.com/h/tek1/first_computer_bug.htm)

In addition to the insect problem, reusability of this kind of software left much to be desired, as you might imagine.

Binary

Later, binary, octal and hex programming and, after that, assembly language, appeared. These were huge improvements over moving wires around. Copying and pasting text was the best that you could do here if you wanted to reuse code. When I say 'copying and pasting,' I mean this, in some cases, quite literally. I've seen people tape or glue pieces of paper punch tape together to make a complete program from parts.

Spaghetti Languages

The first language I ran across once I moved beyond programmable HP calculators was Basic. Like COBOL, Basic didn't initially have subroutines or parameterized functions, which led to hideous gobs of icky spaghetti code. Copying and pasting - thankfully in some sort of text editor so no glue was required - was the most common way of code reuse here. But languages such as Basic and COBOL, though primitive by today's standards, had the enormous advantage of allowing for code reuse on different chip architectures, which you couldn't do with assembly or binary.

Structured Programming

And then structured programming was born. Pascal, Modula 2, and related languages influenced Basic, Fortran and others to allow for libraries of parameterized functions that could be reused. Memory management, string manipulation and device access were common targets of code reuse. Structured programming allows for code reuse quite well and it is still in wide use today.

Objects

It was limited, however. Higher levels of abstraction and reuse were sometimes hard to achieve, especially on very large software projects. And so, Object-Oriented Programming (OOP) with the concept of inheritance was born. It was going to save us all. Great books like “Design Patterns” showed us how. And, in fact, it does help. Some languages are great for it, like Python. Some can push you towards suicide, like C++. Yet there are places where OOP isn't helping much. One indisputable place where OOP by itself is useless is in working with ancient code that isn't OOP.

We want more ability to reuse more of the software that has been out there working for years.

Ultimately, the idea behind SOA is to do for web services what MyYahoo did for static data. SOA should allow you to build dashboards of active web services components and create applications on the fly that behave *your way* – the way you want them to behave.

Open your minds (and your wallets) - here comes the **next big thing**.

Enter Service-Oriented Architecture (SOA) - *the next wave of reusability*.

Run your Legacy IT Shop Like a Startup

Whenever “the next big thing” is announced, CIOs are understandably skeptical. At a recent IBM SOA conference that I attended in Dallas, one CIO of a Fortune 500 firm said to me, *“Is there really a standard here or is this like the OSF where all the Unix vendors bickered endlessly, producing little agreement while Microsoft chortled in the background? SAP says they have thousands of companies participating, IBM says they have 300 ISVs and thousands of 'assets' - are they counting the same things? Are they in agreement over definitions? I don't think so.”*

Another queried, *“Why isn't this just another layer on top of all the other layers that won't be the magic bullet either and, in the end, will also require endless maintenance?”*

These are good questions. While agreement between vendors is often imperfect, Microsoft's .NET, IBM's Websphere and SAP's Netweaver programs all agree on SOAP, XML and WSDL, and these web services technologies do enable the kind of architectures that SOA is creating.

IBM in particular is making serious headway. They have a blizzard of products to enable SOA and have bought a number of companies to add to the capabilities, like Manoj Saxena's company, Webify. Mr. Saxena – now vice president of SOA middleware at IBM - admits that SOA is not a silver bullet.

“It’s more like a pewter bullet,” says Saxena. “But it’s aimed at alleviating the IT hairball we’ve built up over the last four or five decades in IT shops.”

That phrase “IT hairball” is so wonderfully accurate and descriptive.

The SOA idea is simple: componentize pieces of your legacy applications and provide a WSDL layer on top of them. You will then be able to use tools like IBM’s Portlet Factory and Dashboard Framework to drag and drop services into new dashboards to create new applications.

You can pull client lists, project lists, project costs and employee data from SOA-enabled applications like Journyx Timesheet and plug them together with data from Oracle or SAP or Salesforce.com to create new views of data and new applications based on these standardized services. Sounds easy, right?

The benefit of doing this is that you can innovate not just your products and services but even your business models. You can create new ways of selling that the IT hairball would have previously made impossible. Without destroying the value that exists in your legacy applications, you can, in fact, unleash it – thereby getting your IT shop to be as agile as a Silicon Valley startup.

What’s the Catch?

Ah yes, the catch. The catch is that the myriad of applications that you’re plugging together in this way may have disparate security semantics and different performance capabilities, and, of course, testing all this is a whole new ball of yarn. After all, you will have fragmented your application logic to an extreme degree with SOA. It could very easily result in finger pointing. SOA allows more direct communication to potentially vulnerable code residing on the backend. It creates a level of connectivity that was never designed for by the makers of the legacy applications it often exposes.

Perhaps your legacy application is not accustomed to the new load. From IBM or SAP’s perspective, this is an opportunity to sell more hardware, more software, and many more services to get it all rolled out. SOA blurs the lines between infrastructure, applications and business processes. This could lead to an overall decrease in application manageability. IBM’s answer to this will be, “buy Tivoli.” See how this works? But at least they have an answer.

I frequently give speeches to IT audiences around the country and one of the questions I ask people is, “Are you looking into rolling out SOA technologies in the next 12 months?” The number of hands going up on that one has increased substantially over the last quarter or so. Gartner states that “Service-oriented architecture (SOA) will be used in part in more than 50% of new, mission-critical applications and business processes designed in 2007 and in more than 80% by 2010” - Predicts 2007: SOA Advances, November 2007, Yefim V. Natis, et al.

There's no doubt about it, SOA is coming soon to a theater near you.

Challenges for Project Managers

From a project management perspective, SOA presents additional risks to project success. Legacy applications mean legacy application owners and an increase in political risk. How are you going to get all these departments to work together effectively? IBM predicts that the services related to SOA over the coming years will dwarf the software and hardware purchases. So start with a very small project and demonstrate success before you implement the big-bang theory. Technical and engineering risks will be high if your organization hasn't done this before. The testing and verification portion of the project gets more difficult, as does the communications management. Certified project managers become more necessary in an SOA environment.

How Can Small Businesses Get in on the Action?

Consultancies and technology companies need to be educated about SOA. Eventually, software like Quickbooks and SaaS providers like RightNowWeb will become SOA-enabled. Many SaaS providers like Journyx and Salesforce.com already provide WSDL interfaces. Dashboards to enable the blending together of these technologies in an easy SOA fashion can't be that far away. But for now, SOA is primarily aimed at the giant IT organizations served by IBM and SAP. Sun and BEA provide certification classes to get small businesses trained in this new way of thinking. Software consultancies of all sizes can certainly help with the effort.

Conclusion

Any powerful new concept entails risks. The Internet poses risks of fraud, such as Google's nemesis – pay per click fraud – that no one could have conceived of before. SOA is powerful, and it will have some flameouts too. Proceed cautiously, but proceed. You'll get the reusability benefits for which we've all been searching.

About the Author:**Curt Finch**

Curt Finch is the CEO of Journyx (<http://pr.journyx.com>), a provider of free Web-based software located in Austin, Texas, USA that automates billing, payroll & project management by tracking time, expenses and mileage. Finch is a software industry veteran. In 1997, Curt created the world's first Internet-based timesheet application and the foundation for the current Journyx product offering. Curt has managed development teams creating enterprise-level software solutions since 1985, with a focus on distributed workforce management. In 1992, Finch led the team porting Tivoli's product line to the AIX operating system, which led to the company's acquisition by IBM. As a member of the executive team, Curt helped launch The Kernel Group (TKG), a venture-backed firm that grew to 50 employees and \$7.5 million in sales during his tenure. Curt has a B.S. in Computer Science from Virginia Tech University in the USA. Curt Finch can be reached at curt@journyx.com