

Agile and SOA, hand in glove?

A lot of people feel that SOA and agile development bite each other; to agile developers architecture represents big upfront design and 'death by PowerPoint'. To architects, agile developers are like pirates, violating rules and regulations. But if you look past the bias, they actually have a lot in common. We see a lot of value in both and we are passionate about bridging the seemingly widely separated worlds together with you.

For that purpose we examine, in this article, the compatibility between SOA and Agile development. Before we can do that, let's define the two concepts briefly.

Service Oriented Architecture is an architectural style that is a means to achieve business agility using services that deliver business value.

Agile development is a software development methodology that focuses on human capabilities to deliver business value fast.

Agile, SOA and management

Both Service Oriented Architecture and Agile development processes are relatively new and don't adhere to traditional management paradigms like scientific management or bureaucracy [Taylor, Weber, Simon].



I even believe that they go beyond contemporary management theories like contingency- (Vroom) and system theory (Checkland). Both represent a fundamental change in the way things are done: the way people look at:

- organizations (functional & bureaucratic vs. cross functional & process centered)
- ownership (sole vs. shared),
- responsibility (forced upon & divided vs. taken upon & shared)
- management (top down & power vs. meet in the middle & facilitating) and most importantly,
- at people (people as resource of the organization vs. people that are the organization, [P. Senge, M. Buelens]).

So, we state they have a lot in common: Let's test this premise, comparing the 12 agile principles (<http://agilemanifesto.org/principles.html>) to SOA principles. To see how well the two fit, and where the potential mismatches occur.

Unlike agile principles, there is not one source for SOA principles, so we picked a source that is often used by architects; the principles that Thomas Erl has published. (<http://www.soapprinciples.com/default.asp>).

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

This is not as straightforward as it looks on our end of the world. In Anglo Saxon countries (USA, UK) a customer is the end customer, the one that buys the product. In Rijnland countries (The Netherlands, Germany) the customer doesn't have to be the customer of the company. For example, if you build software for an online bookstore, the customer can be the sales department of the online bookstore, not the person that buys a book.

Valuable is often misinterpreted. The software in the case of the bookstore is valuable if the customer of the bookstore can use it easily to buy books, or if the books are cheaper because we don't need a bookstore. Depending on the operating model of the company, we need a cheap solution, or a very sophisticated solution.

Early is also often misinterpreted. If the online bookstore offers online payment, customers of the bookstore won't be happy if you deliver the software without security in place, or with an ever changing user interface. But the customer (sales department) needs early and continuous delivery to prioritize the requirements, run beta tests, adjust their sales process etc. So, to us this principle is about making software that is valuable to the business.

SOA is about the architecture of the enterprise, using services as a defining concept. A service is an activity that adds *value* in a process. This activity can be automated by a piece of software, is a human activity or a combination of the two. Tying software to a service, ensures that it adds value, so this principle fits nicely with SOA. There are also differences: In SOA you check if a service already exists before building it (http://www.soapprinciples.com/service_reusability.asp) even when reuse might take longer than rebuilding it. Development time is not necessarily the most important factor, but just one of the factors important for a short time to market. In a SOA you have a holistic approach to the enterprise, not a local development perspective. This means that license cost, maintenance cost and reusability are all factored in. So sometimes you might decide not to build anything, because it does not add any value to the business. Then you might configure what is already there.

Although Agile is more focused on a (single) project or product and SOA focuses on services and the enterprise as a whole, this agile principle, and the SOA principles are perfectly aligned.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

With SOA as a means to realize business value we create a landscape that consists of smaller, traceable services that allow you to assemble your business processes (http://www.soapprinciples.com/service_composability.asp), rearrange and compose them again and again according to market and customer needs. Because the code isn't tightly coupled to distinct processes (loosely coupled, http://www.soapprinciples.com/service_loose_coupling.asp) and services are autonomous

(http://www.soapprinciples.com/service_autonomy.asp) you can reuse services easily. An important business driver for SOA is corporate agility. By decoupling service providers from consumers, it becomes easier to change implementations. By using standardization it becomes easier to change. By using separation of concerns, it becomes easier to change. SOA is all about change...as is Agile development.

The editor of this article wanted me to address 'generalizations' because it seems that on this topic the two approaches clash. If you take the approaches literally, SOA will answer the question "when do I generalize?" with "Always". This of course makes no sense. Sometimes a problem is very specific to a certain department or capability, or it is very new. In both cases (when it is very specific, or when the problem is not well understood), it makes no sense to generalize the solution. Agile development will answer the question "when do I generalize?" with "Never". This of course, makes no sense either. Sometimes a problem is very generic in a business. Take for example the need for customer data. If you have the data available in a open well performing application, it makes no sense to duplicate this in every application that needs customer data. It is much easier to integrate with the existing application and focus at the business problem at hand.

So, the extremes of agile and SOA both have disadvantages. Combining the two, will mitigate the risks of both these approaches.

On a side note, SOA is not just about reuse and generalization; it is about creating small units (services) that deliver value instead of hard to change silos. This way it is both easier to change and easier to reuse. This design principle is very similar to agile design principles (separation of concerns, avoid waste etc.) that are often used in software development.

To conclude: We can learn a lot from each other here: SOA's got the big picture, agile delivers the small steps (think big, start small, think global act local). Which brings us to the next Agile principle:

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

To gain as much momentum and to prove services work, you best deliver working software regularly and celebrate to help to energize and motivate people. This can be accomplished by working iteratively and incrementally.

Get sponsoring for top down identification of valuable business processes one after the other and translate them into, amongst others, reusable technical transformations and data services. Start small and build your SOA one service and one process at a time.

In an Agile software project usually small (elementary) services are delivered. Standish group calls these stepping stones. To make sure that the stepping stones are actually taking you in the direction you need, and not just moving in circles, calls for adherence to some pretty firm principles. The target architecture and guidelines for the services can be used for that purpose (http://www.soapprinciples.com/standardized_service_contract.asp for example). This means that architecture needs to be taken into account by the product owner and is a constraint in the product backlog. Because of the loose coupling (http://www.soapprinciples.com/service_loose_coupling.asp) in SOA, technology can be chosen at the moment a concrete problem arises but architecture is a guideline for all. By

delivering these stepping stones one by one regularly and frequent in your agile project you can eventually build you SOA. A large organization's Service Oriented Architecture can't be built at once. To use a famous analogy: the way to eat an elephant is one bite at a time. Again think BIG but take small steps.

A good practice from Agile development is to make sure that every effort results in value for the customer immediately. If we apply this to SOA, we only build services as they are needed in the context of a project that is needed by the business. Services can be identified either top down or bottom up. But they are always built in context of a project with end to end requirements. This mitigates the risk of any SOA endeavor: building lots of services that are never going to be used anywhere.

A challenge for both Agile and SOA is to integrate iteration planning with program management and portfolio management. It is also challenging to tune releases to what is manageable by operations, business users and administrators. Often these environments need to adjust to the different pace and smaller chunks of functionality (services) that is delivered.

Business people and developers must work together daily throughout the project.

This principle has implications for the software development process. You can argue that the same is true for architecture; business people and architects must work together daily. Information and technology are more and more of strategic importance in a lot of industries, like government, financial services, utility companies etc. Therefore, Business and IT should form a corporate strategy *together*. Projects only succeed when people of several disciplines work *together*. Services can only be realized in *cooperation* between humans in IT and business. Services are important to business and must have business value; architects, developers and testers must understand this to deliver those magnificent services to the business. This should apply to any project. A common vocabulary or *common understanding* can help all in communications.

So working together is something that agile development and SOA share. Of course there are some important differences. The most notable is, that architecture is a process, not a project. Of course, you can start projects to realize parts of the architecture, and architecture is an important part of a project as well. The added value of SOA in this case, is that architects talk and work together with other business people as well, not just the project stakeholders. This enables them to spot possible improvements and inconsistencies in the features on the product backlog.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Agile clearly mentions the value of people and their craftsmanship. Only motivated people with a clear understanding of vision, mission and strategy towards company goals, encouraged and facilitated by their leaders and colleagues, will get the job done. However SOA doesn't address this it is understood that different process- and / or domain owners have to be motivated to use each other's services. To cooperate and reuse services you need to trust decisions made, and rely on the services others deliver. In a specific project, developers need to trust a colleague's expertise to reuse services that aren't designed by them. Service

contracts by itself or enforcing reuse will never replace motivation or trust. It is just a fact of life.

In real life both 'Agile' and SOA are confronted with fear of the unknown and doubt whether Agile and SOA principles are the right thing to implement. So you have to teach and coach and then let your trusted people decide how to implement.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

This has to do with good communication. Within a development team, face-to-face communication works just fine. Between teams (either in time or place), you need to communicate differently. The important thing is, that you only add value. In order to be able to reuse services, discoverability is an important principle (http://www.soaprinciples.com/service_discoverability.asp) You write whatever documentation you need for other people to (re)use your products and services. Nothing more, nothing less. Again, this is another item on the product backlog. Of course, that is not all you need to do, to make sure that people understand what the services are all about. Architects should spend most of their time in face-to-face communication. Communication with the business, to understand the strategy and objectives. Communication with the developers to talk about the trade-off between fast delivery and easy maintenance, explanation of the meaning and use cases of the existing services etc, etc. Shortcomings on highly technical oriented SOA people is that they think that SLA's and registries etc can replace human communication. However SOA doesn't say anything about this condition for success, it still depends face to face conversation, like any other achievement, for that matter.

Working software is the primary measure of progress.

In projects for realizing services or SOA, the ultimate goal is to build business processes that add value to the business. This can be implemented by a piece of software or by a human service. In SOA you can make progress by building nothing, but by just changing some human activities. The measure of progress is a business measure: like the number of sales, profit margins, or cost cuttings.

So, this is a matter of scope: Agile development is about software development. If you build software, working software should be the primary measure of progress. SOA is about an Enterprise Architecture, so progress is measured by accomplishing business goals in general.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

This principle is about making the life's of workers good or bad. Happy workers, whether or not they are architects or developers, are much more successful, work smarter and deliver better work when they don't have to work under pressure, have to work late, have to work on more than one or two projects at the same time or make longer hours than is healthy for them. This principle should be promoted in any project. This is very sensible and doesn't clash with SOA.

Continuous attention to technical excellence and good design enhances agility.

Agile teams will make technology meet the expectations of the business and make it work: ubiquitous and reliable. SOA demands a high level of technical excellence and design because it demands loose coupling, reusability and abstraction. These principles require extra attention and learning. Agile developers will have to not just design their classes and methods, they have to involve themselves designing the whole system by helping architects and business to design the whole system, together. This gives the Agile design sessions a compelling goal: To look beyond the scope of a single project.

Some developers tell me that high degree standardization (and architecture, for that matter) is seen as unnecessary constraint which is killing their creativity. To use an analogy again: if you need a package delivered as fast as possible, it might be better if there were no speed limits. For the individual delivery company it might seem faster to just ignore all the rules. But for the customer that ordered the package, there is more in life than just the delivery of this one package. He or she also want your kid to be able to walk to school safely. So, while the package delivery company can achieve technical excellence and good design they will have to take into account the environment as a whole. This is where SOA can be very helpful. Because of the notion of small units, there are only a couple of guidelines that any project needs to adhere to. Of course, these guidelines need to be evaluated and improved all the time. Standards should be enabling, not restricting just for the sake of it. Compare this to Sweden, where they decided it would be easier to drive at the right side of the road, like the rest of Europe. They changed the rule when this was the right thing to do.

For both SOA and Agile it is very important to apply and adopt practices and principles consistently and disciplined.

Simplicity--the art of maximizing the amount of work not done--is essential.

SOA defines services as the main deliverable. This is a realization of simplicity, having smaller building blocks that adhere to certain principles (loose coupling, autonomy, composability etc). This approach also helps to define when and what type of software is actually needed. When you only build the software that supports the identified and not yet realized services, you can maximize the amount of work not done. We have a saying. "*Je gaat niet iets bouwen als je het ook op de achterkant van een sigarendoosje kunt bijhouden*". Literally it says: ' You're not going to build something (software) if you can track it by writing it down on the back of a cigar box". We apply this on our SOA practices too.

The best architectures, requirements, and designs emerge from self-organizing teams.

This principle could be applied to SOA as well as to agile software development. In fact, SOA in itself does not prescribe how it should be achieved. You can do this with self organizing teams, or by prescribing an architecture that is ordained by the CIO.

Realizing a SOA, as any other accomplishment, requires people of all disciplines that are all knowledgeable in their respective fields. These people don't need to be steered, if they like to work together. Coordination between business people, architects, process modelers, testers and the service developers is required and multidisciplinary teams need to be created. The important thing here is that, architecture is required because the solutions (services) have to fit into the big picture. Architecture has to be part of the self organizing team. This has to be

organized in some way, it is taken from team to team. Agile practices and self organizing alone can't solve those architectural challenges.

Solution architecture for instance, can be applied in an agile manner, through refactoring, designed in small increments and expressed mainly in code. Architecture then emerges. This is the bottom up part of architecture that meets in the middle with the top-down part.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Because SOA does not say anything about the way to achieve it, just what principles and building blocks we need, this principle could easily be applied to the architectural effort.

In general you have to continuously test and evaluate you efforts, measure quality, learn, learn more and refactor for designing and building reusable and valuable services. Some high level but defined quality standards must be put in place. SOA gives the team a concrete product (the service) to talk about and a set of principles to measure its quality. A retrospective centered on the services that were build, can contain questions like:

- How can we improve reusability?
- How can we make our service more valuable in the eyes of the stakeholder (customer)?
- Which parts of the system could we replace by existing services?
- How can we deliver our next service faster with fewer bugs?
- Can we generalize?

Daily stand-ups, continuous integration and iteration demo's can help to foster inspection and adaption of the architecture as well as the development of software.

In general, Agile is the hand that works in the glove. SOA is the glove, the scope is enterprise wide, agile development is about the way you can develop the part that is supported by software. Most principles of SOA and Agile are not in conflict. Whenever they are, they keep each other sane. Agile development without a clear vision of the goals and objectives of the company is futile. Service oriented architecture without a clear vision how to make it real using agile development principles is a waste of time and money.

SOA and Agile are about agility. (this applies to a number of principles that don't clash) SOA is about architectural structure and agile development is about delivering fast. This way they keep each other in balance. One does not make sense without the other.

About the author

Mary Beijleveld graduated in business administration (MScBA) at the University of Groningen. She works as a senior business consultant, business architect and Agile project manager at Approach in Nijkerk, the Netherlands. Her focus area is the business value of architecture in general and SOA and BPM in particular. She prefers working at the intersection of business and technology, where complex issues have to be addressed and pressure to make practical improvements is high. In addition to that she has a passion to Agile project- and development methods, networking, blogging and writing opinionating articles.

She is a certified scrum master and has long-term experience in managing projects, issue control and has broad knowledge of several project management methods.