Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

SAP | the way we see it

# Packaged Software Implementations Require a Leaner Approach To Agile

**To get maximum value from packaged software, like SAP, apply Agile methods to tailor the small percentage of code that creates differentiating value and reuse everything else.**

People matter, results count.

Every client feels a fundamental tension when starting an SAP implementation project. That is the tension between wanting the rewards that come from leveraging packaged software and the rewards that come from tailoring off-the-shelf code to better align with the client's value proposition. The rewards of a standard code base are the cost savings and lower risks of not reinventing the wheel. Vanilla SAP has been refined over decades of experience and tens of thousands of implementations — very much in the spirit of Lean manufacturing, i.e., improving a product or process by refining it with practice.

Countering those rewards are the ones that can come from tailoring SAP to a client's value proposition. The logic here is simple. The more a company can deliver on its unique way of doing business, the better equipped it is to succeed — whether from lower operating costs, higher customer satisfaction, more streamlined service delivery, or in other ways. But customizations are very un-Lean because they create something new — not something refined because it's been done in a similar way countless times. Custom development, as opposed to packaged implementations, doesn't have that advantage. So, instead of Lean, custom software developers rely on Agile to "get it right."

That is why Capgemini invented an Agile/Lean approach to SAP implementation called iSAP — so clients always get the rewards of both a packaged SAP and a tailored SAP. iSAP achieves that balance by virtue of the proprietary intellectual property captured in the tools and practices Capgemini consultants apply every day on SAP projects. These tools and practices integrate Lean principles from manufacturing (i.e., refinement of a known product or process through practice) with Agile principles from custom software development. That Lean/Agile approach can be summed up in three overarching ideas:

- The best way to know what clients want is to ask them (and keep asking them as new information arrives)
- Focus only on the few business processes that make a difference (the 20% that represent 80% of the benefit and 80% of the effort) — so reuse everything else
- View processes as value streams, not software modules (which value streams traverse), that link raw inputs (like orders) to measurable outcomes (like cash)

Lean optimizes the value stream itself while Agile optimizes the value stream outcomes consisting of continuously updated project deliverables. Clients are happier because they're kept in the loop and because they see they're only paying for tailoring that actually delivers a measureable difference in the marketplace they would not get solely with a generic packaged implementation.

## Where Conventional Packaged Implementations Fall Short

The decades-long success of SAP and other commercial software packages aimed at the enterprise is strong testimony that clients really like the packaged software idea. An idea they have grown to like a lot less, however, is the idea of the packaged *implementation*. If you were to go back in a time machine to the mid-1990s and visit a typical SAP implementation project in a large company, the software would look a lot different, but not so much the day-to-day work of implementing it.

Off-the-shelf software is one thing. An off-the-shelf way to implement the software is something else. The enterprise and the business environment in general have changed radically since the mid-90s. So it makes sense that companies would be looking for something more than just an efficient way to implement software. They want an efficient way to implement their business model.

On the one hand, the enterprise needs to be more agile — able to make big changes quickly to stay relevant in the face of constant marketplace disruption. On the other, it must eliminate waste and optimize processes — to drive down costs in the face of ongoing commoditization.

Here are four key trends:

- **IT staffs are leaner**. The Great Recession, the rise of cloud computing and virtualization, global competition, the increase specialization of IT, and other trends have made it less economical to retain high-paid in-house IT staff full-time. Instead, companies increasingly rely on outside IT consultants — a shift that has changed the nature of the client-consultant relationship. Because consultants are engaged more broadly and more deeply across the organization, they are expected to be more than just implementers; but instead are true partners who collaborate not just with IT, but also with owners and managers.

- **IT has gone from overhead to profit driver**. Back in the 90s, IT was like electricity — something companies paid for to keep the lights on and only noticed if it wasn't working. Today much of the business itself is a digital platform. So, if the software that runs your business is generic so is your business.

- **Rapid commoditization means organizations must be more agile to thrive.** Good new ideas get adopted quickly in a digital world. To stay ahead of imitators (or to adopt new good ideas themselves) organizations must be good at quickly changing how they do business. They can't afford software or a software building process that makes them slow down, or won't help them speed up.

- **Software is the new hardware.** Rapid and massive commoditization also applies incredible pressure to reduce costs everywhere. So just switching to an Agile method won't help much if doing so also adds inefficiencies — say, by leading you down multiple dead ends or to try doing things in a suboptimal way (perhaps because they are new things). That's where software integrators can take lessons from hardware manufacturers, specifically the lessons of Lean manufacturing. One consequence of businesses becoming digital platforms is that the "manufacturing" of enterprise software becomes industrialized — meaning that the same rules and reasons for improving efficiency and reducing errors in producing hardware also apply now to producing software.

These four trends create a conflict for the modern enterprise. On the one hand, the enterprise needs to be more agile — able to make big changes quickly to stay relevant in the face of constant marketplace disruption. On the other, it must eliminate waste and optimize processes — to drive down costs in the face of ongoing commoditization. But rapid wholesale change is the enemy of efficiency. In order to optimize something, you must have done it before — probably many times. And since software is the new hardware, senior management is now looking to its packaged implementation partners to help resolve this conflict, which goes well beyond what they asked them for in the past.

## "I Asked for Packaged; I Really Wanted Lean."

Most executives want Lean manufacturing's benefits (as formulated by Toyota in the mid-1940s) even if they've never formally studied Lean. That's why so many executives over the years have asked for packaged software implementations. In many respects packaged enterprise software is "Lean in a box."

Consider these Lean principles:

- **Organize production into value streams.** Back in the day, ERP packages like SAP were marketed as BPR (business process reengineering) designed to connect functions across departmental silos to deliver something of value in the eyes of the customer. This was a radical change from other applications sold at the time, which was more about delivering a particular function (like, say, accounting or printing payroll checks) than about tying together multiple functions end-to-end (like, say, financial integration, FI, or human resources, HR).

- **Optimize, optimize, optimize!** According to Lean principles, the best outcomes result from optimizing, or eliminating waste from, the value stream, a process called Kaizen — through continuous incremental improvements. Waste can include defects, overproduction, and unnecessary effort. Packaged software is very much a result of Kaizen. Commercial enterprise packages are constantly being tweaked, refined and updated by the vendors, end-user organizations, and third-party system integrators through a process akin to crowd sourcing. In the SAP space alone there are countless blogs, newsletters, online forums, and websites dedicated to finding ways to improve the software.

- **Reuse, reuse, reuse!** This helps achieve efficiency two ways: First, code you reuse is code you do not have to build from scratch, which would take longer and be more expensive (perhaps packaged software's most compelling selling point). Second, reusing existing, which is to say packaged, code enables Kaizen, as in "practice makes perfect."

But just because software is a package doesn't mean its implementation is Lean — so executives are disappointed they're not getting all the benefits they expected. For example, instead of focusing on end-to-end value streams, teams may lose sight of what ultimately has value to the customer and just focus on implementing specific software functions. Instead of Kaizen — optimizing based on client outcomes — they may just focus on getting individual modules to work properly. And when it comes to reuse, they may choose to reuse more than they should — even functionality that does not fit the client's business well — in the name of cutting costs and effort. The result may be a collection of properly functioning, even optimized, modules but a severely suboptimal piece of software overall. That's bound to create disappointment.

Then there is the implementation activity itself, which is its own value stream. If the software is suboptimal after it is implemented, then so too is the process that resulted in the implementation. But the quality of the software that comes out of an implementation is only one measure of the quality of the implementation. Other measures are the speed, cost, and effort it took to achieve that outcome. In many cases historically buyers of packaged implementations have been disappointed on all three of those counts as well.

It was to prevent those disappointments that Capgemini invented iSAP (industrialized SAP). iSAP is a methodology and toolset for implementing SAP. Just as the idea of packaged implementation was invented to expedite the delivery of ERP at client sites, **iSAP was invented to expedite the delivery of packaged implementations of ERP.** It takes over where the package leaves off — by

optimizing the implementation project value stream in order to optimize the client's business value streams. For example, as an iSAP best practice, the test of whether a particular module is implemented correctly is how well the end-to-end value stream enhances the client's competitive advantage with that module present in the stream. iSAP does this by focusing the project team's attention on the 10-20% of the code with the greatest potential for Kaizen or creating measurable, difference-making, and marketable outcomes. Capgemini calls this practice Design By Acception®, a play on the Lean phrase, "design by exception." Whatever is not an exception is therefore a candidate for reuse — which is 80-90% of the code.

As noted, content reuse is a key Lean concept because it allows for continuous improvement and because teams are not wasting time reinventing the wheel.

With iSAP, however, the reuse percentage is typically higher than 80-90% because iSAP teams reuse more than just the code. They also reuse the other project artifacts that complete the entire project value stream — such as design specifications, business objectives, KPIs, and unit tests, as well as the PRICEFW artifacts (portals, reports, inputs, conversions, enhancements, forms and workflows). iSAP provides repositories that store versions of these artifacts prepopulated, and preconfigured into off-the-shelf business value streams — analogous to SAP's off-the-shelf applications. Examples of iSAP value streams include order to cash, procure to pay, and maintain to settle — all of which are end-to-end journeys that start with an input like an order and end with an outcome, such as cash in the organization's bank account.

But content reuse can be neither automatic nor blind. If it were then iSAP projects would always result in generic one-size-fits-all code. In other words, teams would become very efficient (and very Lean) at achieving a generic result — much more so than if they were to just implement SAP the conventional way, i.e., by reusing off-the-shelf modules instead of off-the-shelf value streams. Projects then would not produce the difference-making capabilities that create measurable and marketable impacts. Rather than enable rapid business change in response to the four key trends cited earlier, the software would simply freeze clients in place faster.

## Agility Required

Getting to the wrong result faster is not what iSAP is about. That's why, in addition to adapting Lean from the manufacturing world, iSAP also adapts Agile from the software development world. As originally conceived and as used today throughout the IT industry, Agile is a software development methodology, not a packaged software implementation methodology. It is based on the premise that software requirements are a moving target. Agile assumes that clients' knowledge of their own requirements will evolve as the software takes shape — a situation all too familiar to

> Capgemini iSAP methodology was invented to expedite the delivery of packaged implementations of ERP. It takes over where the package leaves off — by optimizing the implementation project value stream in order to optimize the client's business value streams

anyone who has ever developed software for a client, and for many reasons, including:

- Clients simply forget a requirement until they start using the software and discover features are missing.
- Clients think that the need for something is so intuitively obvious the need for something is so intuitively obvious that it doesn't have to be stated explicitly.
- Only after they see the software in action do clients see the potential for new functionality not previously considered.
- Business conditions change between when the software project started and when the software is finished.
- Requirements leave room for interpretation, and clients prefer the requirements be interpreted differently once they see the software working.

Fully understanding client requirements is especially critical when less than 20% of the coding effort is needed to capture 80% of the requirements — as in a packaged software implementation. In that scenario small units of coding effort have a lot more leverage to satisfy difference-making requirements (more so than in custom development when 100% of the coding effort must capture 100% of requirements). That's why packaged software cries out for an implementation method specifically tailored to meet big impact business requirements rapidly by leveraging small units of code (the "exceptions").

That's Agile. And those smaller units of coding effort are called sprints. Agile is engineered to quickly and continually satisfy frequently updated requirements — whether due to dynamic business conditions or to clients' evolving understanding of what they actually need.

Agile does this by organizing projects into parallel activities called sprints, each driven of a self-organizing team of typically six to 12 consultants, and with each sprint team demonstrating a new working deliverable to the client every two to four weeks. "Working" means the deliverable is:

- A demonstrable functioning piece of the final system
- Something a non-IT business stakeholder can assess
- Something that fulfills a business requirement agreed to in advance
- An "exception" — i.e., makes a measurable contribution to the value stream

This constant stream of updates ensures that evolving requirements stay in sync with the software as the software evolves. If there is a fault in a deliverable the fix can simply be incorporated into the next sprint rather than after the fault either becomes part of — or is invoked by — other software, which would also need to be fixed and only after the original problem was tracked down, diagnosed, and corrected.

Sprint teams can therefore respond quickly and cost effectively to changing requirements, which in turn allows the client to respond very quickly to changing business conditions. And since most of the software is already done at project start (that's why it's packaged), most requirements are already met at project start — so relatively few sprints can fulfill most remaining requirements with relatively minor tweaking of the entire package before go-live.

Agile therefore represents a dramatic acceleration of project timelines versus the conventional Waterfall method employed in most packaged software implementations. That's where clients experience long periods of "radio silence" from their consultants — waiting until the complete implementation is designed,

implemented and tested before the finished result is demonstrated. That can be months or years after project start. Waterfall only works if clients know their requirements precisely at project start and if those requirements (even if known precisely) would not change anyway due to changing business conditions.

The fact that this virtually never happens leads to all sorts of frustrations, delays, and missed opportunities. Clients wait longer to see results. What they do eventually see does not reflect what they need. And the time, effort, and cost to make the needed fixes become prohibitive — perhaps to the point where clients decide to settle for a generic version of the packaged software rather than get the game-changing functionality they really wanted.

### Applying Agile To Packaged Software Implementations

So why hasn't Agile been the standard way to do packaged software implementations all along, and what does iSAP do that makes this possible now? After all, Agile methodologies actually evolved alongside ERP packaged systems in the mid-1990s. However, they are based on exactly opposite ideas. Agile grew up around the idea that you don't know what you want before development starts. Packaged software presumes that you already know what you want (or at least most of what you want).

The fact is most organizations do know 90% of what they want in an ERP package before they buy it or they wouldn't make that large investment. What they may not know are the "exceptions" — the few difference-makers that will require custom, or at least tailored, development.

**iSAP's key insight is that Agile will work in packaged software implementations if you do it in a Lean way.** In other words, it's not about implementing packaged modules. It's about implementing the value stream. So organize your sprints only around the difference-making business requirements. Those are the "exceptions" that most impact the value stream. And, while you're at it, don't limit reuse to just the code. Reuse as many, and as much of, the other project lifecycle artifacts mentioned earlier as you can — so in effect you're optimizing the client's business value stream by optimizing the packaged software implementation project value stream.

iSAP incorporates a very effective tool to do all this, called the Error Proofing Tool (EPT) — that is both a project management tool and a content auto-generation tool. Sprint teams use the EPT to establish and measure their progress against specific milestones (sprint begin-end dates, relevant modules, function descriptions, etc.) and the key issues to be decided for the milestones to be achieved. They can also use the tool to auto-generate project artifacts — e.g., functional design documents, PRICEFW artifacts, unit tests, and more — that have been prepopulated to enable predefined, i.e., off the shelf, value streams. Content that won't be used as-is (i.e., the "exceptions") can then be updated and versioned within the EPT repository. At the end of each sprint the entire updated value stream, any updated parts of the value stream, and any associated updated non-code artifacts can be generated and demonstrated to business stakeholders. The benefits of Agile are thus realized by leveraging rather than opposing the Lean advantages inherent, but not previously exploited, in packaged software.

**Here's the takeaway:** Organizations are increasingly frustrated by the limitations, slow speed, and waste of Waterfall methods. They like the economies of packaged software — reusing code they don't have to write from scratch — but they also don't understand why they can't get the benefits of Agile too. At issue is not the packaged software itself, but the packaged software implementation. The solution is to align Agile with Lean's value stream "exceptions" — the sprint-size project pieces that can be done in two to four weeks for maximum impact, ongoing optimization, and rapid response to change.

For more details contact:

**Herbert Goertz**
SAP Engagement Executive
email: herbert.goertz@capgemini.com
Phone: +1 561-303-9188

**Bart Neal**
Technology Consulting Services
email: bart.neal@capgemini.com
Phone: +1 214-395-9134

## About Capgemini

With more than 180,000 people in over 40 countries, Capgemini is a global leader in consulting, technology and outsourcing services. The Group reported 2015 global revenues of EUR 11.9 billion (about $13.2 billion USD at 2015 average rate). Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at
**www.capgemini.com**