Pillir

HOW TO ELIMINATE TECHNICAL DEBT IN SAP

by Vaidya Aiyer, C.E.O. & Founder

TABLE OF CONTENTS

DEFINING TECHNICAL DEBT

Ward Cunningham, a pioneer in software development, inventor of "the wiki," and co-author of the Manifesto for Agile Software Development, once said, "Some problems with code are like financial debt. It's OK to borrow against the future, as long as you pay it off."

Technical debt in software development can be described as the implied cost of future rework from choosing the wrong solution today or the compounding costs of building on top of a legacy technology stack. Sometimes, even the best software solution can generate technical debt as the technology stack matures or becomes outdated.

Technical debt discussions generally arise in software engineering houses or in large IT organizations - both of which create their own custom, in-house applications. This conversation also extends to packaged ERP software providers, such as SAP. The bottom line is that any technical development done in-house, or done by consultants, will most likely contribute to technical debt.

However, similar to financial debt, technical debt is not always viewed negatively. For instance, technical debt might be the necessary result of having to meet a business stakeholder's requirements or timeline. But, technical debt still needs to be maintained and monitored, as it can create bigger problems down the road. Keeping a core system like SAP as standard as possible is essential to minimize an organization's technical debt.

THE IMPACT AND CONSEQUENCES OF CREATING TECHNICAL DEBT

Technical debt can have both positive and negative consequences. Sometimes, organizations create technical debt for speed-to-market - especially when launching a new product or testing a new feature. Building a little technical debt, as a trade-off for functions, features or quality, can occasionally be appropriate. Additionally, quick user validation or new product revenue can help justify a new release or help accelerate a product roadmap.

However, the impact and consequences of technical debt can be immense - costing organizations millions of dollars. Technical debt can accumulate slowly and almost without notice, until it suddenly becomes a huge issue for organizations. There are several ways that technical debt can impact your organization.

High Upgrade or Migration Costs

Technical debt becomes more apparent when an organization attempts to upgrade or migrate its application(s). This is evident via technical analysis, pre-checks, remediation, and other costs that all add up to the final upgrade cost. As a result, organizations tend to delay their upgrade which only results in the continued accumulation of technical debt, as business requirements still have to be met. A recent survey from ASUG notes that SAP customers, on average, wait 24+ months to upgrade to S/4HANA - indicating the breadth of planning, costs, and business justifications required to hit the go button on an upgrade.

Long and Expensive Maintenance Cycles

Technical debt is not only disruptive during a major upgrade cycle but also in everyday life such as patches being applied to keep systems up-to-date and compliant. For example, applying EhP (Enhancement Packs) and patches to SAP applications. It's well-known that EhP and patch upgrades become a major project for every

TOP REASONS CUSTOMERS ARE WAITING 24+ MONTHS

#1 Prioritization

#2 Lack of Resources

#3 Waiting for the product to mature

#4 Need to justify to the business

organization, and can take months to apply requiring an incredible amount of effort and man power for each upgrade. As a result, organizations typically delay patch upgrades, which increases problems, while core applications are vulnerable and on legacy releases.

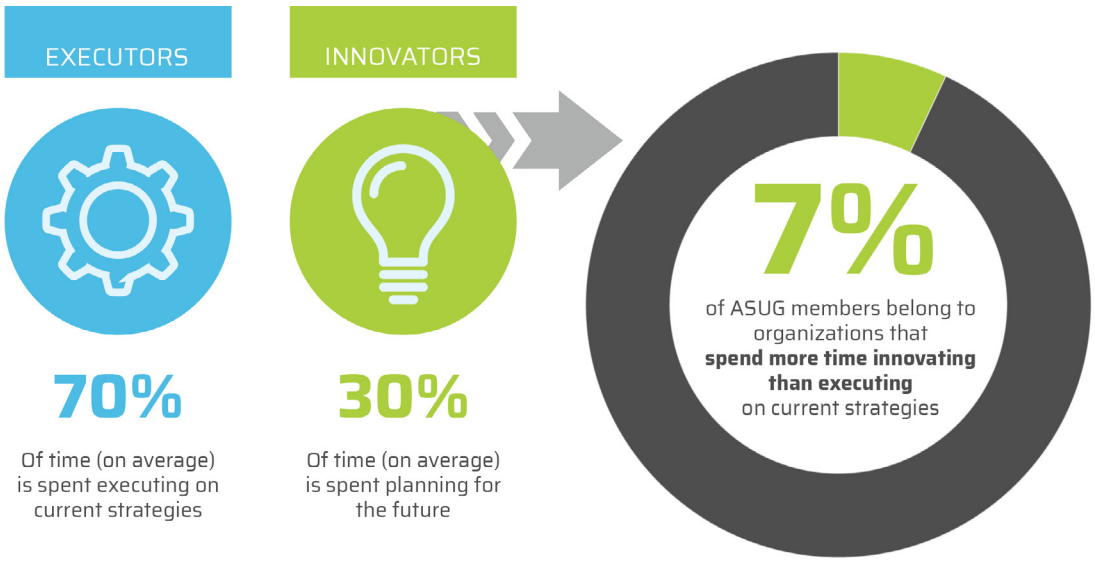
Lack of Innovation

Over time, technical debt creates the need for additional resources, time, and effort for IT professionals - meaning organizations spend more on simply keeping the lights on and routine maintenance, than on innovation. A recent survey by ASUG found that only 7% of SAP organizations spend time on innovation vs. day-to-day execution. Accordingly, 93% of SAP organizations spend their time, man power, and effort on maintenance and keeping the lights on. As such, organizations are using some of their brightest minds and best talent on maintenance vs driving innovation. SAP experts within the organization know the business processes inside out and are generally the best people to uncover innovative ways to solve real business problems. Keeping these individuals focused on maintenance activities will reduce the organization's ability to differentiate from their competition and stifle their most technical resources.

Shortage of Bandwidth

The same ASUG survey found that 68% of organizations (77% in the public sector) indicated the single biggest reason for a lack of innovation within their organization as a "lack of resources (budget, time and staff)." This is currently the norm because the people most familiar with an environment are stuck answering daily support requests or performing maintenance activities rather than driving tech innovation across business lines. Over time, tribal knowledge gets built up amongst a select group who effectively "own" the environment and are the only people who can make changes to the technology (e.g. any and all custom ABAP programs). In the not too distant future, these knowledge experts will move on and it will become increasingly difficult to find and identify replacement talent. Moreover, when the institutional knowledge of legacy systems is tied to a select group, massive challenges will arise as an organization roles out new digital initiatives or attempts to innovate legacy processes.

Technical debt also creates recruiting challenges. It is hard to find the right talent outside the organization when a system has been built on an older technology stack. Quality ABAP programmers are hard to come by and not many people are sharpening their skills in order to become an expert ABAP programmer. Furthermore, recent college grads are not joining technology teams with the hope of maintaining and troubleshooting legacy ABAP code.



Q: WHAT PERCENTAGE OF TIME WOULD YOU SAY YOUR ORGANIZATION SPENDS ON THE FOLLOWING? N=625

TOP 3 SOURCES OF TECHNICAL DEBT IN SAP ECOSYSTEMS

Customizations (a.k.a. non-core modifications)

SAP is known for its integrated, industry-best practices for business processes. However, these out-of-the-box processes generally do not meet the needs of organizations, so the system needs to be customized in order to meet unique business needs. As a result, IT organizations have spent time, money, and resources to ensure their out-of-the-box SAP implementation meets their organizational needs.

A custom line of code is not a problem, however, as custom lines of code accumulate over time, organizations lose the ability to track the customizations within these 100's and 1000's of lines of code. Further, SAP is considered a system of record, and organizations should not be building and accumulating unique processes in their system of record. The most recent major upgrade across the SAP technology stack in the SAP world have been moving from SAP R/2 to R/3 in the mid-90's, i.e., moving from mainframes to the client-server technology and, subsequently, moving from R/3 to ECC / Business Suite in the mid-2000's. Many customers have been running SAP since R/3 and that means any customization that customers have put in place in R/3 have moved to ECC and will now likely move to S/4HANA. That means thousands of custom objects, or technical debt, per SAP instance will need to be remediated and moved to S/4HANA. These customizations can be considered technical debt.

Outdated Technology

S/4HANA is the latest product from SAP that has been built on HANA database. S/4HANA is built on ABAP (Advanced Business Application Programming), which was modeled after COBOL in the 1980's and introduced in SAP R/2. Since its introduction, ABAP has been SAP's core language of choice for building all of their applications. That is not necessarily a negative, but issues typically surface when customers write custom programs for their own needs in the ABAP language.

ABAP is nearly 40 years old and quickly becoming the oldest coding language in use. Developer productivity when building in ABAP has not changed, even as new development tools have come to market. Some organizations are evolving and beginning to adopt new technologies, like low-code software tools that provide drag-n-drop development capabilities. However, many other organizations are stuck in their ways using ABAP, which requires intensive coding skills from a dwindling pool of professionals. Unfortunately, legacy programming languages and ABAP customizations are destroying IT and business productivity.

SAP is well aware of the problems that ABAP customization can cause. In fact, SAP introduced the Java stack in SAP NetWeaver in the early 2000's. They pushed the Java stack in all of their applications until Oracle acquired Sun Microsystems, the owner of Java. As a result, SAP moved away from Java and has attempted to modernize ABAP. To SAP's credit, they have made a concerted effort, and ABAP programs can now run in the SAP Cloud platform. That said, this does not

solve the core issue of ABAP being an out-of-date language that is not suited for today's world and the evolving workforce.

Development Practices

It is critical to define software development best practices. Unsurprisingly, this is extremely challenging within SAP because custom lines of code are created over a number of years by various contributors, such as in-house staff, consulting firms, contractors, and offshore developers. As a result, it is very difficult to design and implement consistent development standards, such as naming conventions and documenting the logic and variables. Given that most SAP installations are 10-20 years old, it's also likely that programmers have lost the tribal knowledge required to maintain custom developments. Some of the customizations may be so large that they could be considered stand-alone applications. With so many hands involved in writing the code, it is highly unlikely that all of the developers were capable of writing quality ABAP code. The inefficient and inconsistent development practices create technical debt. Per Ward, technical debt, by itself, is not always a problem – but significant problems will come up as the complexity and amount becomes unwieldy. These issues are directly tied to the amount of custom code.

HOW TO DISCOVER AND ESTIMATE TECHNICAL DEBT

Discovery and Visualization

All customizations in SAP start with "Z" or "Y." In order to find all technical custom objects, first find the objects starting with "Z" or "Y". There are several custom object types that can be created in the ABAP dictionary such as Z*Tables, Z*Data elements and Domains, Z*reports, Z*function groups and Z*function modules, Z*SAP Scripts and Z*Forms, Z*TCodes, and many more.

SAP provides various mechanisms to find all the custom objects in the system. One of the most common ways to find custom objects in SAP is using the Database tables that store all of the metadata for the custom objects. For example:

To find...

- **Z*tables:** Go to TCode SE11; Run a Query on table TADIR where PGMID = R3TR and Object = TABL
- **Z*data elements:** Goto TCode SE11; Run a Query on table TADIR where PGMID = R3TR and Object = DTEL
- **Z*domains:** Goto TCode SE11; Run a Query on table TADIR where PGMID = R3TR and Object = DTEL
- **Z*Reports:** Goto TCode SE11; Run a Query on table TRDIR where SUBC = 1 (for executable programs)
- **Z*Module pool programs:** Goto TCode SE11; Run a Query on table TRDIR where SUBC = M (for module pool programs)
- **Z*Function Modules:** Goto TCode SE11; Run a Query on table TFDIR and select the field FUNCNAME
- **Z*Function Groups:** Goto TCode SE11; Run a Query on table TADIR where PGMID = R3TR and Object = SSFO
- **Z*SAP Scripts:** Goto TCode SE11; Run a Query on table TADIR where PGMID = R3TR and Object = FORM

NOTE: This is not an exhaustive list to find all the custom objects in SAP and more information can be found in SAP's help site.

In essence, all of the SAP custom objects can be found in the SAP data dictionary. However, it is not easy to identify these objects and it takes a specialized skill-set, i.e., ABAP knowledge. Also, the system does not provide easily digestible reporting and more importantly it does not include all of the necessary information.

From SAP ECC EhP7 onward, SAP provides a new tool called ATC (ABAP Test Cockpit). The ATC is a code management and analyzer tool that can be used to retrieve the custom objects' information. It was released to run static check and ABAP unit tests for ABAP programs. The ATC is compatible with SAP's Code Inspector, which can be used to check SAP repository objects. More information on SAP's ATC can be found here.

Although SAP's ATC has made it a little easier to collect all of the information, it's still a technical undertaking to run the T-code, collect the information, and create a report that business stake-holders can understand.

3rd party tools, like Pillir's NANCI, can discover all of the custom objects and visualize all of the custom objects within a SAP environment.

Total Cost with the Technical Debt Index (TDI)

In order to reduce technical debt, it's imperative to understand and document the total cost that it adds to the IT budget. By identifying the total cost of technical debt and applying a unit of measure, organizations are able to compare and measure progress on the path to elimination. The unit of measure should be easily understood by both non-technical and business team members.

There are many Unit-of-Measures (UoM) to help translate technical debt. They cover various aspects, such as program complexity, lines of code, maintainability index, and depth of inheritance. Given the intricacies and complexities of SAP, the most applicable measurement is the Technical Debt Index (TDI).

TDI can be simply defined with the following equation:

TDI for (Y) years = [Remediation Cost + (Debt Maintenance Cost x Y years)] / [Development Cost + (New Maintenance Cost x Y years)] x 100%

Wherein:

- TDI = Technical Debt Index
- Y = Number of Years
- Remediation Cost = Total Cost to Fix or Revise the Code
- Debt Maintenance Cost = Total Cost to Maintain the Technical Debt per Year
- Development Cost = Total Cost to Redevelop the Code Again in the Latest Technology Stack
- New Maintenance Cost = Total Cost to Maintain the New Development per Year

For example, to calculate the TDI for 5 years:

- Remediation cost = $10,000
- Debt Maintenance cost = $2,000
- Development cost = $190,000
- New Maintenance cost = $2,000

The calculation would be: *[10,000 + ($2000 x 5 Years)] / [$190,000 + ($2,000 x 5 Years)],* equaling 10%

A TDI averaging 10% over a period of five years is actually a positive debt ratio - similar to a financial debt ratio. However, using that example, if remediation costs increase to $100,000.00 and debt maintenance increases to $4,000.00 per year then the TDI becomes:

TDI = [$100,000 + ($4,000 x 5 years)] / [$190,000 + ($2,000 x 5 years)], equaling 60% for 5 years

Similar to a Financial Debt Index (FDI), a TDI of 60% is a very high ratio. If an organization's TDI is this high, IT department professionals should seriously consider opportunities to reduce the debt.

Organizations can actively reduce maintenance costs by adopting modern development platforms and more productive development frameworks. When the remediation cost and debt maintenance cost remain the same, but the new development cost drops to $25,000 with a $1,000 per year maintenance, here is what the TDI amounts to:

TDI = [$10,000 + ($2,000 x 5 years)] / [$25,000 + ($1,000 x 5 years] = 67% for 5 years

In this scenario, new technologies, such as cloud and low-code development platforms, have actually increased the TDI for ABAP customizations, despite having lower development costs.

This formula not only considers the total cost to maintain ABAP customizations but also considers the impact of adopting new technologies. This formula provides a uniform ratio so IT and Business leaders can easily understand the Total Cost of Ownership (TCO) of custom and legacy code.

BENEFITS OF ELIMINATING OR REDUCING TECHNICAL DEBT

After establishing the Technical Debt Index (TDI), organizations should establish a path towards reducing their technical debt with a goal of reducing it altogether.

There are many benefits associated with reducing technical debt within mission critical business applications like SAP, including:

- **Drive innovation:** custom code forces the organization, and most technical talent, to focus on maintaining the status quo vs creating innovative solutions to solve business problems.
- **Increase productivity:** Eliminating or reducing technical debt increases productivity for all of your SAP personnel, both technical and functional team members. They are often some of the most knowledgeable personnel about business operations and can be highly productive in solving business problems.
- **Repurpose or Reduce headcount:** Reducing non-core customizations will reduce the need for expensive ABAP developers (in-house or consulting partners). Additionally, these highly technical and business savvy ABAP'ers can move on to new products and solutions in SAP or focus on other technologies across the organization.
- **Improve projects:** Fewer customizations means lower costs and quicker timelines to implement new features in SAP. This also means less testing and better maintenance.
- **Lower cost to upgrade:** Fewer customizations means quicker turnaround time and lower costs to remediate custom objects.
- **Enhance Change management:** When upgrading to newer SAP versions, custom code creates roadblocks and friction at the business level. By reducing custom code, personnel and unique business processes are less likely to be impacted by version upgrades and new systems.
- **Reduce training:** during upgrades, customizations need to be modified according to the new SAP version. This usually requires additional training. Accordingly, reducing or eliminating non-core customizations will decrease specialized training needs.
- **Reduce change requests backlog:** Custom development often perpetuates custom development. whereby business users require more changes as the business grows, leading to a backlog of change requests. Eliminating technical debt reduces the change request backlog.
- **Simplify M&A and Divesture effort:** During an M&A event or Divestiture, systems are either combined or split across companies. The more custom code in SAP, the more difficult these processes are on everyone involved.

- **Improve Working Capital:** A lot of non-core customization usually requires a large support team, be it in-house or outsourced, consuming a large portion of the IT budget meaning there is less working capital for the rest of the business.
- **Adapt to market changes:** Custom ABAP code in SAP makes it more challenging for businesses to be nimble and responsive. Configuring a standard SAP system is hard and customizations add more time and effort when a business needs to rapidly respond to changing market demands.

HOW TO REDUCE OR ELIMINATE TECHNICAL DEBT

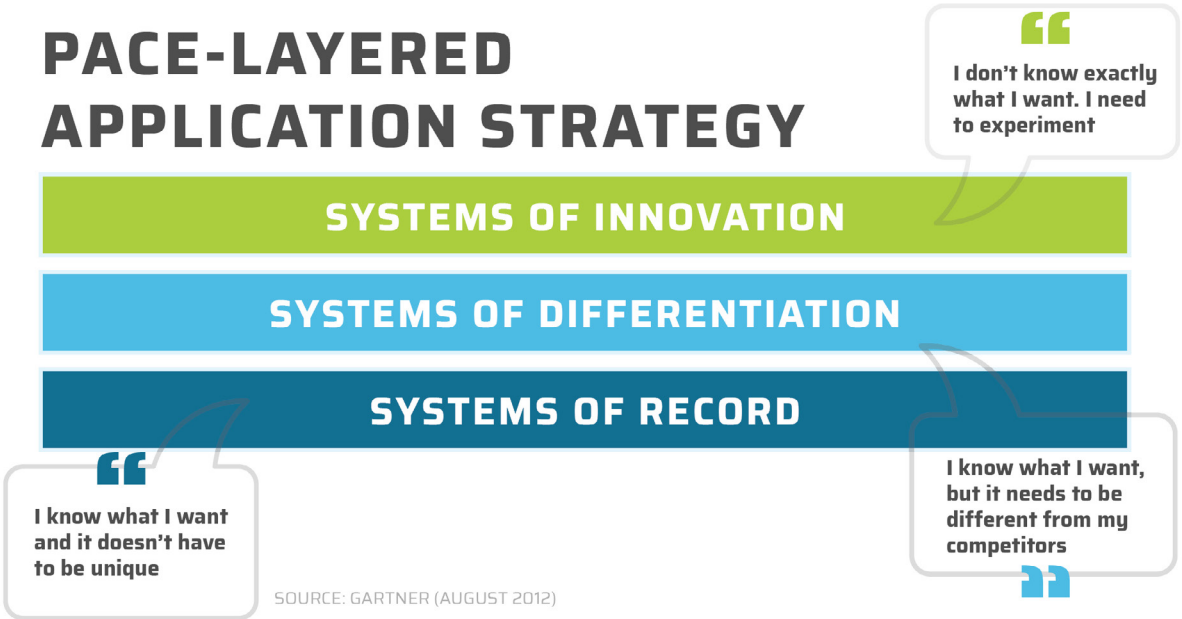
Many organizations assume that there is no way to reduce technical debt, so they feel obligated to remediate and migrate to a new system, such as S/4HANA. What's worse is that organizations will use ABAP as their development platform to remediate and customize their SAP system creating additional technical debt.

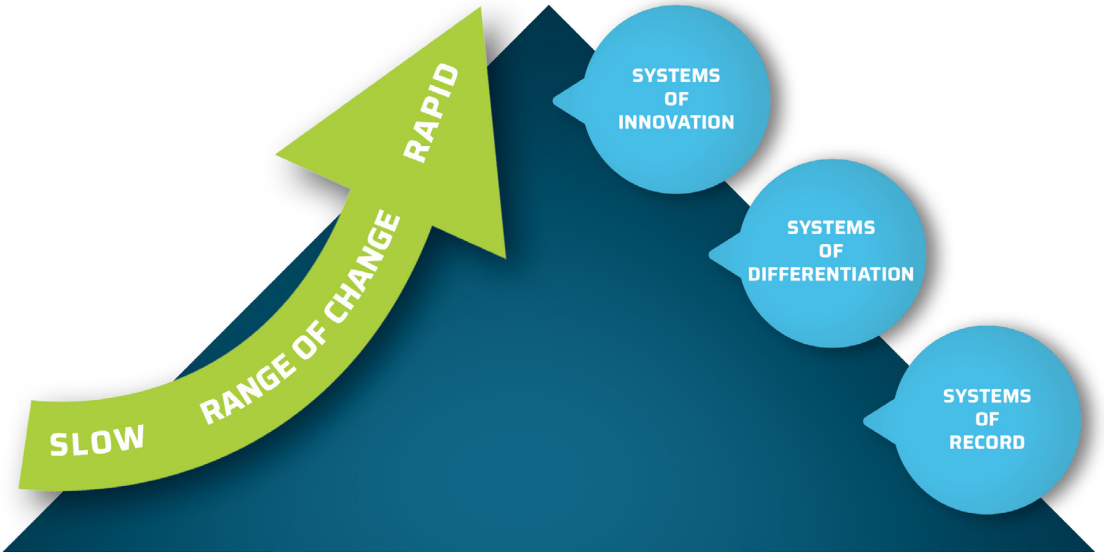
Best Practices IT Architecture

When migrating to a new technology architecture, such as S/4HANA, a company should consider their technical debt and begin to think about how to reduce it. Creating an architecture framework based on IT best practices is a great place to start and is applicable to any organization. For example, organize IT applications into something like a Pace Layer approach for applications as defined by Gartner.

Gartner defines three application categories, or "layers," that distinguish application types and helps organizations develop more appropriate strategies for:

- Systems of Record (SOR) — Established packaged applications or legacy homegrown systems that support core transaction processing and manage an organization's critical master data. The rate of change is low, because the processes are well-established and common to most organizations and often are subject to regulatory requirements.
- **Systems of Differentiation (SOD)** — Applications that enable unique company processes or industry-specific capabilities. They have a medium life cycle and a differentiator or a competitive edge to the company.
- **Systems of Innovation (SOI)** — New applications that are built on an ad hoc basis to address new business requirements or opportunities. These are typically experimental and start with a Proof-of-Concept or a Pilot that is tested by the organization.



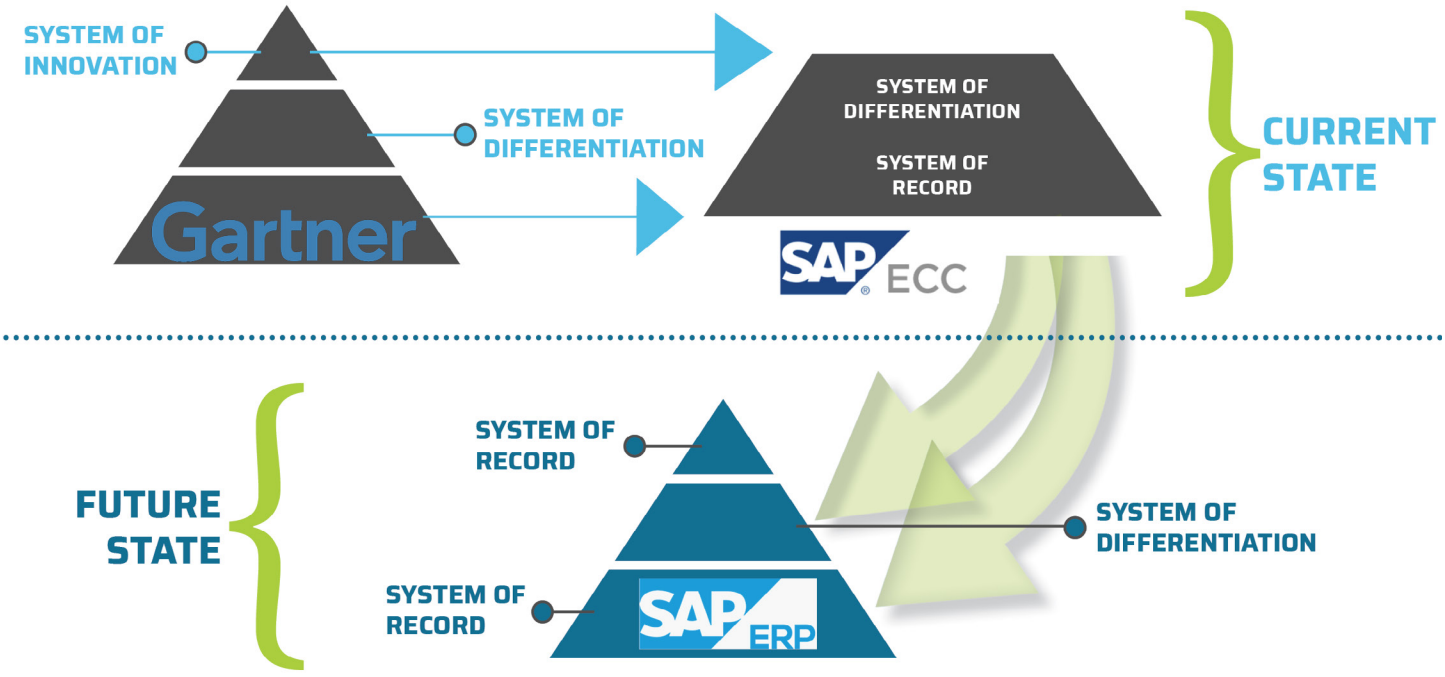


An SAP ERP system and other core business applications would fall under SOR. Differentiation applications and processes such as customer service, Product Lifecycle Management (PLM), and pricing configuration would fall under SOD. Innovative applications, like ML/AI, Blockchain, and other new technologies, would fall under SOI. The pace of change also differs on the three layers as shown above.

The SOR needs to be the most stable as it runs the core business applications and has the slowest pace of change, while SOI needs to be the most nimble and responsive as it requires the most rapid pace of change.

Unfortunately, most SAP customers have collapsed their SOR and SOD into one big block by using ABAP to build highly complex customizations in SAP. This approach diminishes an organization's ability to move quickly.

When organizations move to S/4HANA it makes the most sense to build the SOD and SOI layer separately from the SOR, in order to help organizations move at a faster pace. This begins with pulling customizations apart and out of SAP.

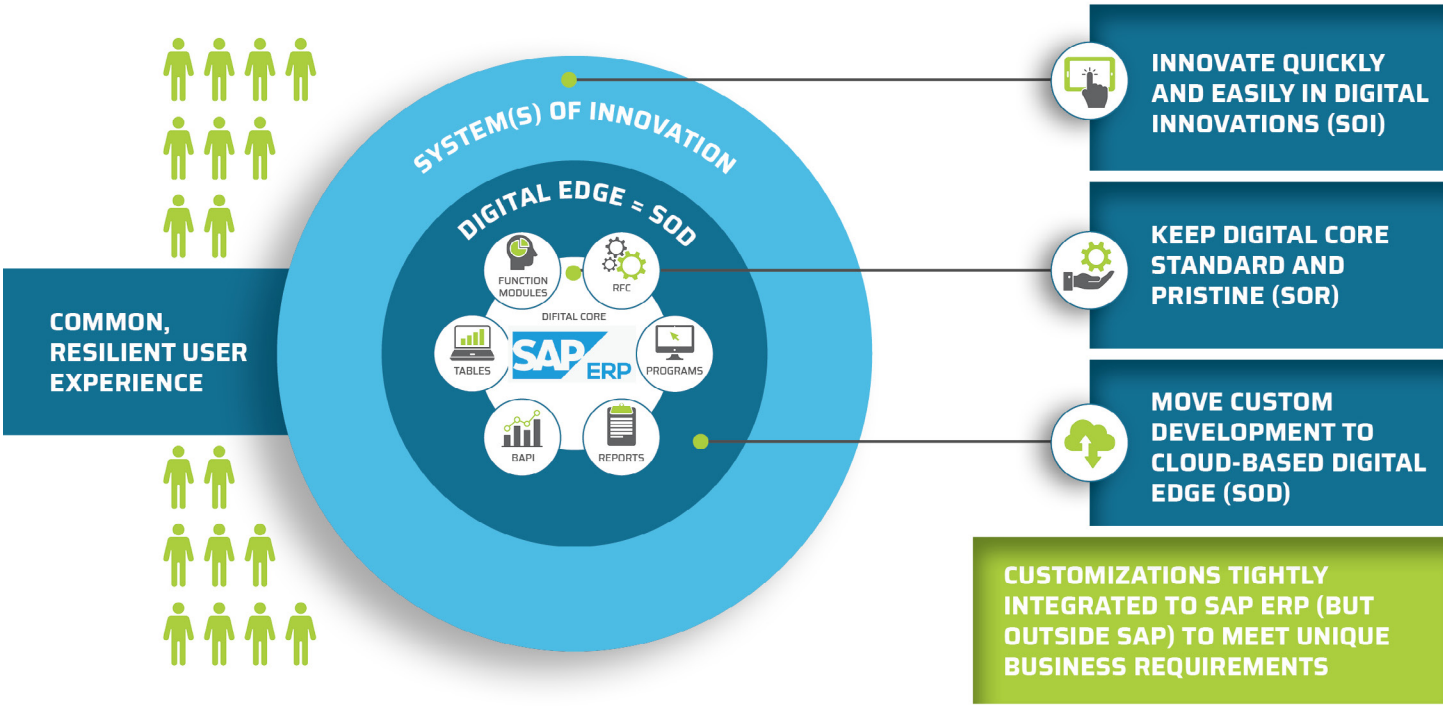


Digital Wrapper

By creating a digital wrapper around the digital core, organizations are able to pull customizations outside of SAP and allow them to work seamlessly with the existing SAP core. All customizations would be able to operate seamlessly with the SAP core and require no additional effort.

For example, a digital wrapper that needs ODATA service for every RFC and BAPI, defeats the purpose of the solution, as ABAP effort is simply replaced with ODATA effort. A digital wrapper needs to have native integrations within an SAP core such that it can integrate seamlessly with SAP's standard function modules, tables, and objects (such as a Z-TCode written inside of SAP). Ideally, a digital wrapper solution would use the EII (Enterprise Information Integration) model as opposed to EAI or ETL integration models. The digital wrapper would form the System of Differentiation (SOD) around your digital core, or the System of Record (SOR).

The goal of a digital wrapper solution is to keep the SOD, like SAP, clean and pristine with limited customizations. All non-core customizations can be done in the SOD.



Types of Customization

All ABAP customizations in SAP can be classified into three categories:

- Reports (Z-Reports)
- Self-contained applications (Z-Tcodes)
- Embedded customizations (User-exits, BADI, etc...)

Not all customizations will be the right fit to migrate out of SAP into a SOD. Each one needs to be carefully categorized and assessed to either be moved out of SAP (but still tightly integrated) or kept inside of SAP.

Z-Reports

The advantage of shifting reports to a SOD is the ability to modernize without losing any significant advantages of it running within SAP. Modernizing reports could provide the following benefits to an organization:

- Run it as a Web-App in modern browsers or even in mobile devices.
- Make quick enhancements and changes as required, without going through the traditional development cycle in ABAP.
- Feed data from multiple sources to the report to generate comprehensive reports.

A report is a stand-alone, custom development in SAP that can be executed separately, as long as it can access the data and maintain the processing power of SAP (i.e., the ability to process data within the SAP application itself).

Z T-Codes

Stand-alone applications, or Z T-codes, are the easiest candidates to move into a SOD. They would need access to master data and all transactional data within SAP; however, standalone applications would be considered the primary target to be moved outside of SAP in order to reduce technical debt. Some of the benefits, in addition to those mentioned above for reports, are:

- Easily complete application process enhancements.
- Ability to make it a composite application in order to get data from multiple data sources.

Embedded Customizations

Customizations that are done within SAP and are also part of the overall business process may be more challenging for candidates to remove from SAP, specifically any customizations that are BADI /user-exits. Certain customizations should be kept within the SOR for optimal performance.

Most businesses require multiple Systems of Differentiation and there are a few solutions available today that are also great alternatives to creating ABAP customizations, such as SAP Cloud Platform and Pillir's EdgeReady Cloud.

WHEN TO START ELIMINATING TECHNICAL DEBT

Similar to financial debt, it's best not to have any debt, but unfortunately that is nearly impossible. So, the next best alternative to having no technical debt is to actively work to reduce the amount and its implications. The quicker an organization can move to reduce their technical debt, the better.

Technical debt reduction can begin at any stage of an SAP journey. Whether an organization is in SAP ECC or Business Suite, ready to move to S/4HANA and beginning a migration project, or already in the process of moving or have moved to S/4HANA, technical debt reduction can start at any time.

Technical Debt Index (TDI) is the leading indicator that can be used by organizations to determine when to start the debt reduction process. TDI can be evaluated at the application level, (i.e., the average TDI of all objects in an application or at each non-core modification or customization). TDI can be calculated for each Z-report or each Z-Tcode and can be prioritized based on TDI. A TDI above 20% would be a good candidate for moving to a System of Differentiation (SOD). Any ABAP customization under 20% could be considered low priority. If the average TDI of all the objects in SAP is less than 15% then there is nothing to worry about!

That said, technical debt reduction does not necessarily need to be a large project. Depending on the SOD solution, it can be a small project that can be done in weeks and will most likely pay for itself. It is recommended to do a thorough evaluation of the SOD in order to receive maximum benefit.

Once the technical debt is reduced in SAP, the core will be clean and pristine, and all customizations will be built in the SOD.

Conclusion

Technical debt in SAP is often an overlooked subject. Most customers simply do not consider all of the implications associated with technical debt in their SAP system. Over time, the debt adds up and leads to an unnecessarily high Total Cost of Ownership (TCO). Organizations now have the right tools and resources in order to identify and manage their technical debt. Investing in these tools and actively pursuing a clean System of Record pays for itself immediately and dramatically reduces overall costs for an organization. A low technical debt level allows organizations to spend more on differentiation and innovation instead of spending on maintenance and keeping the lights on.

To learn more about how Pillir can help you reduce your technical debt, please schedule a discovery call with our team here. Once we connect, you will have free access to our ABAP Discovery Tool, NANCI - which will discover all of your ABAP Customizations in under an hour - as well as the TCO for remediating and migrating and, of course, maintaining. The tool provides immediate insight in an easy-to-read format for any and all stakeholders - including non-technical stakeholders.