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 is described as the implied cost of additional rework by choosing an easy or limited solution now, or building on an older technology stack instead of a more robust approach that may take longer. Sometimes, even the best software solution generates technical debt as the technology stack matures or gets outdated.

Technical debt discussions generally arise in software engineering houses or in large IT organizations - both of which are used to creating their own custom, in-house applications. This conversation typically extends to packaged ERP software providers, like SAP, as well. The bottom line is that any technical development done in-house, including work conducted by consultants, most likely will contribute to the definition of technical debt.

However, like most financial debt, not all technical debt is viewed in a negative light. Technical debt is sometimes necessary in order to meet your business stakeholder's requirements, but it still needs to be maintained and kept low, just like financial debt, or it may cause more issues for you down the road. Keeping your core system as standard as possible is essential while creating and managing technical debt in your SAP system.

When customers are trying to migrate to S/4HANA, technical debt can be a struggle to identify and migrate; it can, at times, be the primary reason why SAP customers are hesitant about migrating from legacy SAP systems to S/4HANA.

THE IMPACT AND CONSEQUENCES OF CREATING TECHNICAL DEBT

Technical debt, similar to financial debt, can have both positive and negative consequences. Sometimes, speed-to-market may be the driving factor, especially when launching a new product, or when a business only wants to test or validate their market. In this instance, it's OK to build up a little bit of technical debt, such as a trade-off in function, features, or quality, in order to bring a solution to market faster. On the positive side, the market validation as well as the revenue or traction generated will be sufficient justification to eventually update the software or even redevelop it entirely.

Comparing this to financial debt, the negative aspect of acquiring technical debt may be the principal and interest you pay on the debt itself. The principal may be the one-time cost required to fix the problems, however, the interest could be the result of continuous inefficiencies, increased maintenance, additional required testing, and attaining specialized resources which could all be cost-accumulating factors.

The impact and consequences of technical debt creation and management 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

High amounts of technical debt are generally visible when an organization wants to upgrade or migrate its application(s). This can be seen via technical analysis, pre-checks, remediation, and other relevant costs that 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, cost, and business justifications needed to be given to decision makers in order to justify the upgrade.

Top Reason Customers Are Waiting 24+ Months

1. Prioritzation
2. Lack of resources
3. Waiting for the product to mature
4. Need to justify to the business

Long and Expensive Maintenance Cycles

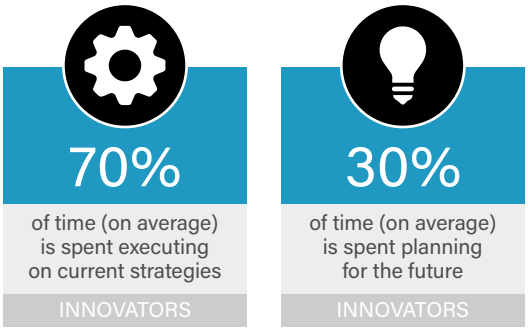
Technical debt not only faces you during a major upgrade cycle but also in everyday life when patches are being applied to keep your systems up-to-date and supported. An example could be applying EhP (Enhancement Packs) and patches to your SAP applications. It's a well-known fact that these EhP and patch upgrades become a major project for every organization, and sometimes can take months to apply with an incredible amount of effort and man power for each upgrade. This typically results in organizations delaying patch upgrades, compounding their problems by keeping their core applications vulnerable and on legacy releases.

Lack of Innovations

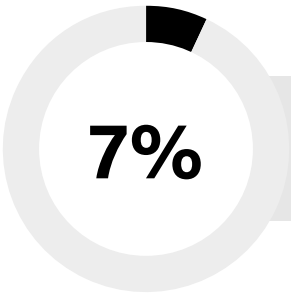
Eventually, excessive or unnecessary technical debt can result in the need for additional resources, time, and effort for IT professionals - resulting in additional organizational spend towards simply keeping the lights on and routine maintenance, instead of organizational innovations.

A recent **survey by ASUG** found that only 7% of SAP organizations consider themselves to be spending time on innovation rather than day-to-day execution, meaning that 93% of SAP organizations are spending additional time, man power, and effort on activities that only encompass keeping the lights on.

This also means that organizations, as a whole, are using some of their brightest minds and best talents on maintenance activities that do not differentiate the business - instead of innovation. SAP experts within the organization know the business processes inside out and are generally the best people to innovate and solve real business issues. Keeping these individuals focussed on maintenance activities is the exact opposite of what an organization should be utilizing these individuals for.



What percentage of time would say your organization spends on the following? n=625



of ASUG members belong to organizations that spend more times innovating than executing on current strategies

Shortage of Bandwidth

The same ASUG survey found that 68% of organizations (77% in the public sector) indicated their single biggest reason for lack of innovation within their organization as the "lack of resources (budget, time and staff)." This is usually the case when the people who are most familiar with your environment are stuck meeting the needs requested in daily support and maintenance activities instead of finding time for differentiation and innovation.

This typically happens when a lot of tribal knowledge has been built in the existing systems and only a few select people know the technology to fix it (e.g. custom programs written in ABAP can be fixed only by the same set of programmers who know the system very well). Soon, the availability of these knowledge experts is scarce and impacts almost every project. Moreover, it's very difficult to add resources to these knowledge experts, not to mention the risk to the organization with all knowledge constrained within a few individuals.

Another result of technical debt are recruitment challenges of finding talent from outside the organization, especially when a system has been built on an older technology stack. For example, the difficulty of finding a high quality resource pool of ABAP programmers is a major recruitment challenge. Given the fact that a new breed of college graduates do not come with ABAP knowledge, we have an ever-decreasing pool of ABAP resources. These resourcing and development challenges also lead to delayed project timelines and may contribute to exceeding project budgets.

SOURCES OF TECHNICAL DEBT

To better understand and address technical debt, it is essential to understand what caused the debt to be acquired in the first place - especially in the context of SAP, when it is known for providing an off-the-shelf, packaged ERP software.
The top 3 sources of technical debt in SAP ecosystems are:

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

SAP is known for its integrated, industry-best practices for business processes. However, all of its out-of-the-box processes generally do not meet the needs of organizations and its implementations need additional customizations in order to meet unique business needs. As a result, IT organizations have spent time, money, and resources customizing their out-of-the-box SAP implementation in order to meet their organizational needs.

Customizations by themselves are not a problem, however, when organizations have been adding customizations to their SAP instances for the last 15-25 years, they tend to become excessive or get lost among the 100's of 1000's of lines of code. Further, SAP is considered a system of record, and a system of record is not where we want to be building or customizing for unique processes. The last major upgrades across the technology stack in the SAP world have been moving from SAP R/2 to R/3, i.e., moving from mainframes to the client-server technology in mid 90's and, subsequently, from R/3 version to ECC / Business Suite in 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 has moved to ECC and subsequent versions and now are ready to be moved to S/4HANA.That could be thousands of custom objects per SAP instance that need to be remediated and moved to S/4HANA. These customizations can be considered technical debt. Per Ward, technical debt by itself is not the problem - the size, complexity, and amount of the technical debt, a.k.a customizations,

Outdated Technology

S/4HANA is the latest product from SAP that has been built on HANA database - the latest database released from SAP. However, the application itself is built on ABAP (Advanced Business Application Programming), which was modeled after COBOL in the 1980's and introduced in SAP R/2. Ever since that introduction, ABAP has been the core language of choice for the building of SAP applications by SAP. There is really no issue in that, however, an issue does arise when customers have to write custom programs for their own needs in the ABAP language.

ABAP is beginning to age at nearly 40 years old, becoming the oldest coding language still in use. The productivity of building in ABAP has not changed, even as new development tools have come into the market. With the continuous evolution of technology, organizations have adopted drag-n-drop development processes, whereas ABAP still requires intensive coding skills that can be completed only by a dwindling pool of professionals. This reliance on specific legacy programming language and approach has brought down the productivity of IT organizations that have built these customizations using ABAP over the years.

S/4HANA is the latest product from SAP that has been built on HANA database - the latest database released from SAP. However, the application itself is built on ABAP (Advanced Business Application Programming), which was modeled after COBOL in the 1980's and introduced in SAP R/2. Ever since that introduction, ABAP has been the core language of choice for the building of SAP applications by SAP. There is really no issue in that, however, an issue does arise when customers have to write custom programs for their own needs in the ABAP language.

ABAP is beginning to age at nearly 40 years old, becoming the oldest coding language still in use. The productivity of building in ABAP has not changed, even as new development tools have come into the market. With the continuous evolution of technology, organizations have adopted drag-n-drop development processes, whereas ABAP still requires intensive coding skills that can be completed only by a dwindling pool of professionals. This reliance on specific legacy programming language and approach has brought down the productivity of IT organizations that have built these customizations using ABAP over the years.

SAP, itself, has been well aware of this problem with ABAP for years. In fact, SAP introduced the Java stack in SAP NetWeaver in the early 2000's and was pushing Java stack in all of its applications until Oracle acquired Sun Microsystems, which owned Java. As a result of that acquisition buyout, SAP started stepping back from Java and began modernizing ABAP. To SAP's credit, they have A modernized ABAP as much as possible, including introducing the ability to build ABAP programs in the SAP Cloud platform. That said, this does not solve the core issue of ABAP being an out-of-date language that may not be the best solution for today's evolving workforce - especially with the need for multi-system business processes.

In addition, the declining availability of ABAP developers adds to the cost of technical debt within an organization. Despite how good the custom programs are, the fact that they're built on old programming languages, like ABAP, creates technical debt for IT organizations in terms of resourcing and the total cost of ownership of maintenance.

Development Practices

It's critical to have defined best practices for software development. However, when it comes to SAP customizations, they have been written over a number of years by various individuals - some by in-house staff, and others by consulting firms, contractors, and offshore developers. As a result, it's 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 these customizations may be so large that they could very well be an application by themselves. With so many hands involved with the code, there's no guarantee that all of the developers were truly qualified to build quality ABAP code. Such inefficient or inconsistent development practices over a number of years typically results in the creation of technical debt.

HOW TO DISCOVER AND ESTIMATE TECHNICAL DEBT

In order to reduce or eliminate your technical debt, it's critical to discover, visualize, and estimate the total cost of the technical debt that you possess. The discovery and visualization should be easy for stakeholders to comprehend, but it's critical that the estimation of technical debt should include the remediation cost to fix it for your migration, as well as the cost to maintain it in your new SAP system.

Discovery and Visualization

All customizations in SAP start with "Z" or "Y." In order to find all technical customization objects, one needs to first find 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 all the custom objects in SAP is by using the Database tables that store all the metadata for the custom objects. For Example:

It should read, To find:

Z Tables:
Goto 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.
https://help.sap.com/viewer/index

In essence, all the SAP custom objects can be found in the SAP data dictionary. However, the big disadvantage is it's not easy to find these objects and it takes a specialized skill set, i.e., ABAP knowledge, and it's not a straightforward report that you get from the system with all the required information. One has to collate all the information from various queries or write a custom report (another Z-report) to get the list of custom objects. There are other transactions available to get the list, but then again, they do not provide a complete list.

Now from SAP ECC EhP7 onwards, 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 the information, it's still a technical effort to run the T-code, collect all the information, and put these into a report for business stake-holders to understand. There are 3rd party tools, including Pillir's NANCI, that discover all custom objects and provide a visualization of all custom objects. It is recommended to evaluate all such 3rd party tools for completeness and compatibility with all the SAP versions and EhP's to discover all the custom objects.

Technical Debt Index (TDI)

In order to improve anything, it's helpful to first understand and document the total cost it is adding to our IT budget. Once we can qualify the total cost of the technical debt, it's much easier to make comparisons and measure our progress in eliminating it. The quantification of technical debt should be a unit of measure that's easily understood by both non-technical and business team members.

There are many Unit-of-Measures (UoM) for the measurement of technical debt that cover various aspects of the debt, such as program complexity, lines of code, maintainability index, and depth of inheritance. Given that SAP is a unique software, the measurement that applies the most is the Technical Debt Index (TDI).

TDI can be defined simply in an equation like this:

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

Where in:

- 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, if you want 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 for said TDI 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 to have - 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.

As new development platforms are created and more productive development frameworks are available, new development and maintenance costs will slowly decrease. In such a scenario where the remediation cost and debt maintenance costs remain the same, but the new development cost drops to $25,000 with a $1,000 per year maintenance, here is what the TDI would amount 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 increased the TDI for ABAP customizations, even though the TCO for these have stayed the same.

This formula provides a uniform ratio in terms of percentage that can easily be understood by anyone within any industry or position - specifically the IT or business side of the house. This formula not only considers your TCO in maintaining ABAP customizations but also puts into perspective that as new technologies arise, there will constantly be new ways to create and manage them.

HOW TO REDUCE OR ELIMINATE TECHNICAL DEBT

Most organizations assume that there is no way to reduce technical debt and begin remediating and moving it onto a new system, such as S/4HANA. What's worse is that some organizations also undertake all of their custom development in SAP and use ABAP as the development platform - resulting in the continuous creation of technical debt.

Best Practices IT Architecture

When migrating to a new technology architecture, such as S/4HANA, it would behove a company to consider their technical debt and begin to think about how to reduce it. The best way to start considering how to reduce it is to rely on the best practices of IT architectures that are applicable to any organization. As an example, organizing your IT applications into something like a Pace layer approach for applications as defined by Gartner.

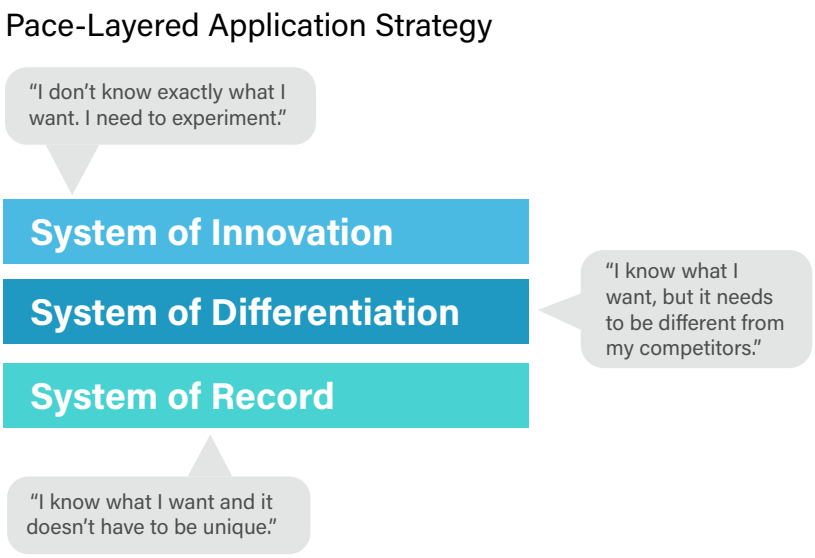
Gartner defines three application categories, or "layers," that distinguish application types and helps organizations develop more appropriate strategies for:

- **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.

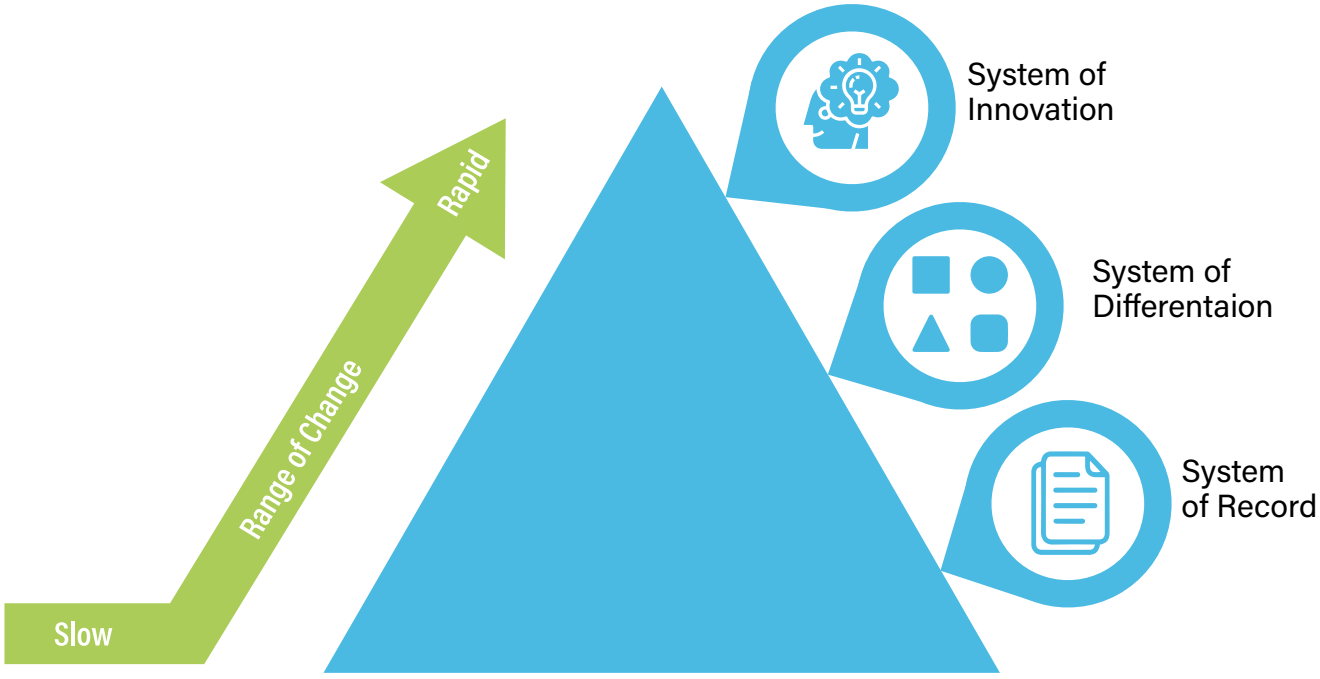
- **Systems of Differentiation (SOD)** — Applications that enable unique company processes or industry-specific capabilities. They have a medium life cycle and a differentiator or a competitive edge to the company.

- **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 **(POI)** or a pilot that is tested by the organization.

Figure 1. A pace-layered view of systems

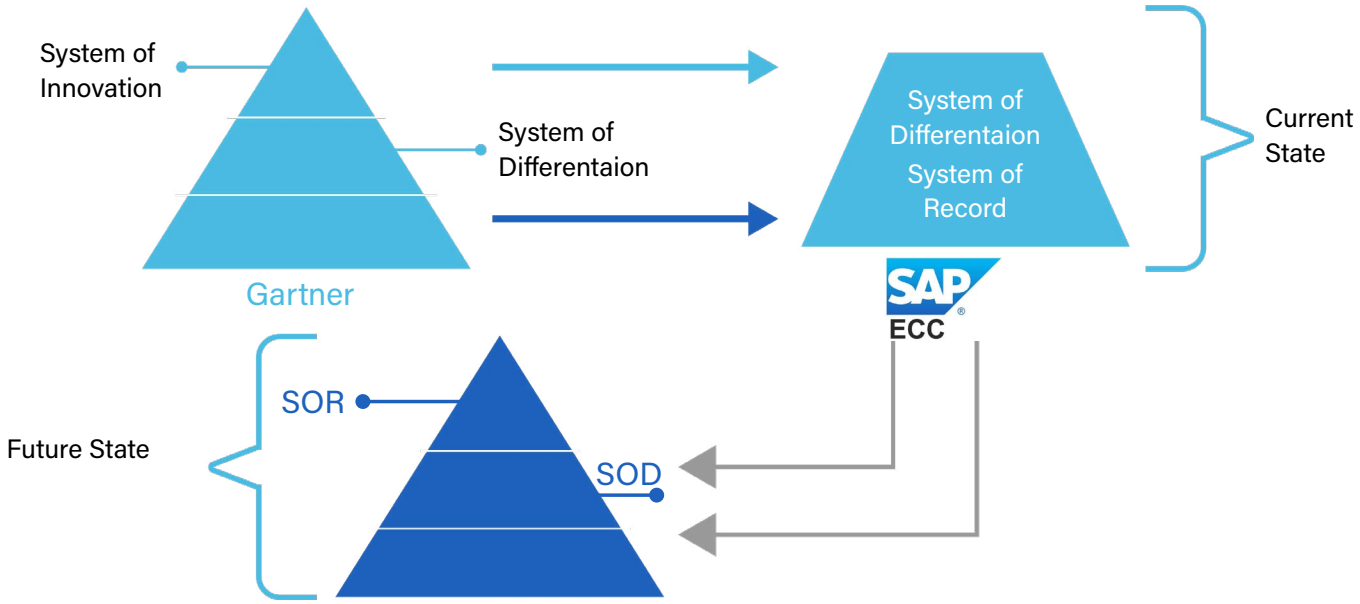


From a typical SAP customer's landscape perspective, an SAP ERP system and other core business applications would fall into the SOR. Differentiation applications and processes such as customer service, Product Lifecycle Management (PLM), and pricing configuration would fall into a SOD. Innovative applications, like ML/AI, Blockchain, and other new technologies, would fall into SOI. The pace of change also differs on the three layers as shown below.



The SOR needs to be the most stable as it runs the core business applications and has the slowest pace of change. An SOD has a faster pace of change to applications, whereas SOI changes rapidly.

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



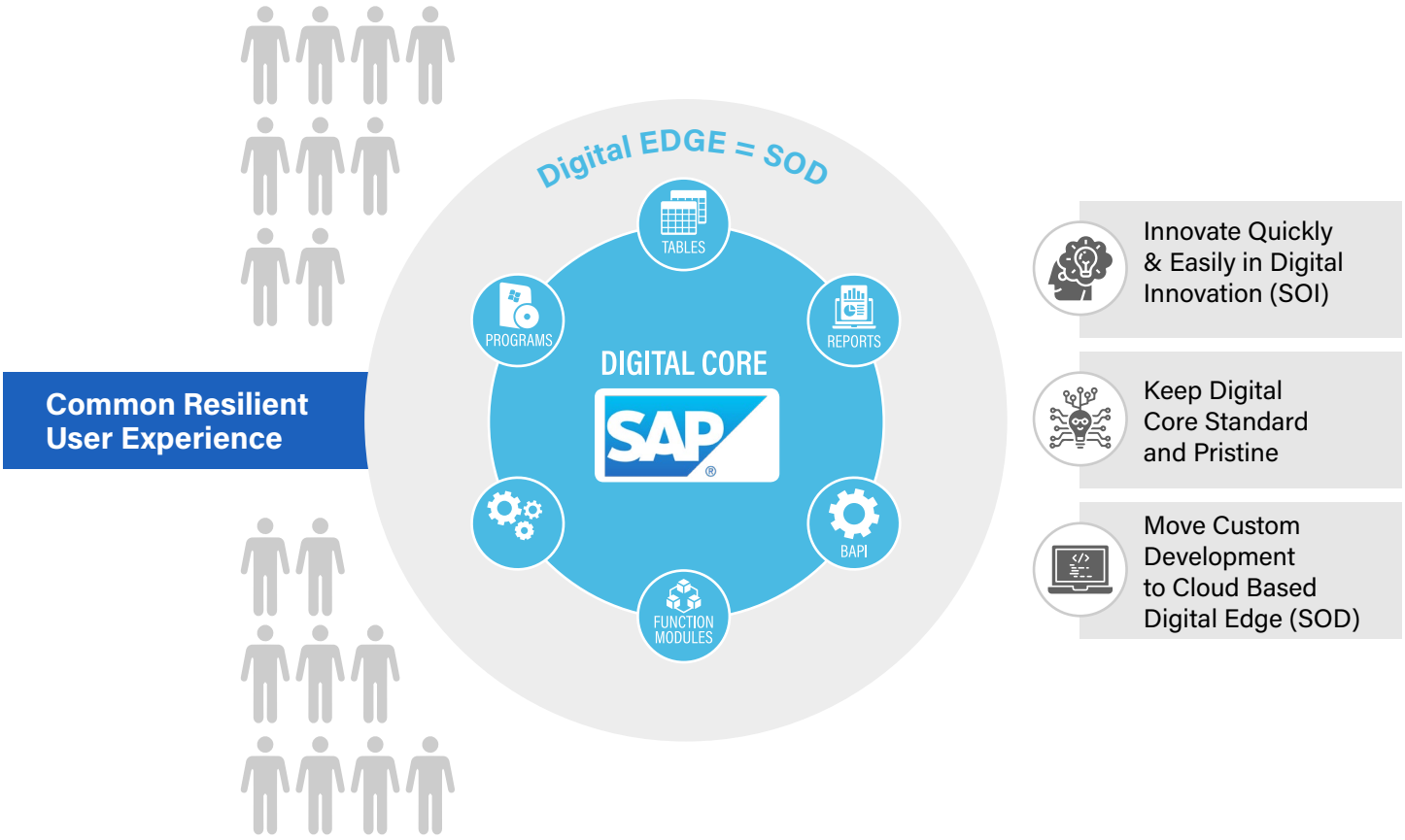
When organizations move to S/4HANA it would be ideal to build the SOD and SOI layer separately from the SOR, in order to help organizations move at a faster pace. This begins with pulling customizations apart and out of SAP.

Digital Wrapper

In order to pull customizations outside of SAP and make them work seamlessly with the existing SAP core, a solution that forms a digital wrapper around your digital core is required. As an example, all customizations need to operate seamlessly with the SAP core without additional effort.

For example, a digital wrapper solution that needs ODATA service to be created and made available for every RFC and BAPI, defeats the purpose of that solution to begin with, as ABAP effort is simply replaced with ODATA effort and time. 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 other objects (such as a Z-TCode when 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 SOD around your digital core, or the SOR.

Figure 1. A pace-layered view of systems



The goal of a digital wrapper solution is to keep your system of record, such as SAP, clean and pristine with limited customizations. All non-core customizations can be done in the SOD (System of Differentiation).

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 you have all access to the data as well as 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 of this, in addition to ones mentioned above for reports, are:

- Process enhancements for the application can be done more easily.
- Ability to make it a composite application to get data from multiple data sources.

Embedded Customizations

Customizations that are done within SAP and are also a part of the overall business process may be tough candidates for removal from SAP. Specifically, any customizations that are BADI / user-exits. Certain customizations should be kept within the SOR as well, for optimal performance.

Several solutions in the market could qualify as a SOD to be moved into your ABAP customizations, like SAP Cloud Platform, Pillir's EdgeReady Cloud, and others. Each solution has its own advantages and disadvantages - and we have yet to meet an organization whose business requirements don't require multiple SODs for different scenarios. It is recommended to conduct due diligence on which solution is the best for your organization.

WHEN TO START ELIMINATING TECHNICAL DEBT

Similar to financial debt, it's best to never have any debt whatsoever; however, that is almost always impossible. The next best possible solution to being technical debt-free is to start reducing your debt as soon as possible. The earlier one can begin to reduce technical debt the better.

Technical debt reduction can really begin at any stage. Whether you are in SAP ECC or Business Suite, ready to move to S/4HANA and beginning a migration project, or are 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 as well and can be prioritized based on TDI.

A TDI above 20% would be a good candidate for moving to the SOD. Any ABAP customization under 20% could be considered as low priority. If you are one of the lucky organizations that average the TDI of all their objects in SAP to be less than 15% then you have nothing to worry about. However, if you have a high percentage, the earlier you start the reduction the better.

Technical debt reduction does not necessarily need to be a large project. Depending on the SOD solution chosen, 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. Not choosing the right SOD may mean that you have a lower TDI but you are building technical debt from the very beginning in the SOD.

Whichever option you choose, once the technical debt is reduced in SAP, you will be able to keep your core clean and pristine by building all customizations in the SOD and not building within SAP.

CONCLUSION

Technical debt in SAP is often an overlooked subject. Most customers simply do not consider all the implications of having technical debt within their SAP system. Over a period of time, the debt adds up and leads to high TCO. In today's day and age, with the availability of various technologies, organizations should keep a lookout for technical debt and keep it under control. This not only pays for itself very quickly but also keeps the costs low for the organization. Keeping low technical debt also adds up on the innovation index and organizations can spend more in differentiation and innovation instead of 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 once you are live in S4H. The tool gives this insight in an easy-to-read format for any and all stakeholders - including non-technical stakeholders.