Salesforce Anti-patterns (Part 1)

Svet Voloshin

What are anti-patterns?

In the context of technology, software development, or systems design, an anti-pattern is a common response to a recurring problem that is usually ineffective and counterproductive.

Anti-patterns are strategies that may **seem to solve** a problem in the **short term** but can cause additional **problems** in the **long term**.

They are typically used when the best practice or appropriate solution is not known or not understood. The term is often used in contrast with the term "pattern," which refers to a **repeatable** and effective approach to a common problem.

Anti Patterns
Product Owner
Scrum Master
Developers
Leadership and Management
Manage by metrics
Comparing teams
Hijacking Scrum Events
Telling the team how to do things
Proxy Product Owner
No Sprint Goal
Not a change agent
No themes or epics
Peopl are not dedicated to one team
Trying to go around problems instead of solving them
Getting Developers to work on "side projects"
Don't understand what an Agile Mindset is
Having leads on Scrum Teams
Component based stories
Scrum Master Cancels the Retro
Unvailable Product Owner
Project focus instead of Product focus
No time for improvement
Hiding technical debt

Why Should We Care About Anti-Patterns?

The **impact** of anti-patterns on the Salesforce ecosystem can be **significant**, affecting a range of aspects from productivity and efficiency to cost and customer satisfaction. Here are some specific ways anti-patterns can impact the Salesforce ecosystem:

1. **Reduced Efficiency:** Anti-patterns often involve taking a **roundabout approach** to solving problems, or using solutions that aren't optimized for the task at hand. This can reduce efficiency, wasting resources, and slowing down workflows.
2. **Increased Costs:** Anti-patterns can lead to **increased costs**, both in terms of development and maintenance. For example, over-customization can cause higher development costs, more complexity, and increased maintenance costs in the long run.
3. **Poor User Experience:** Certain anti-patterns can lead to a poor user experience. For example, neglecting user experience and designing overly complex interfaces can **frustrate users** and lead to **low adoption rates** of the system.
4. **Data Quality Issues:** Anti-patterns like poor data management can lead to serious data quality issues, which can affect everything from reporting accuracy to **decision-making capabilities**.
5. **Security Risks:** Anti-patterns can potentially introduce security risks. For instance, not using roles and profiles appropriately could result in unnecessary access to sensitive data.
6. **Reduced Scalability:** Anti-patterns can limit the scalability of the Salesforce ecosystem. For example, excessive use of code and over-customization could make it harder to **scale the system** as the organization grows.
7. **Upgrade Challenges:** Salesforce releases **three** major updates every year. Over-customized Salesforce environments might face challenges during these updates, and new features or improvements might not work as expected due to custom-built functionality.

These impacts highlight the importance of avoiding anti-patterns and adhering to Salesforce best practices for development and management.

Consequences of not addressing anti-patterns

Over-customization in Salesforce refers to the practice of heavily tailoring the out-of-the-box functionality beyond what is needed or useful, often using code. This often results in a system that is difficult to maintain, upgrade, or use effectively. The system also becomes more prone to bugs and errors due to the added complexity.

Examples of over-customization in Salesforce include:

- **Excessive Apex Triggers:** Writing Apex Triggers for functionalities that can be achieved using **out-of-the-box** features like flows or validation rules.
- **Custom Objects for Standard Functions:** Creating custom objects for functions that standard Salesforce objects could handle. For example, creating a custom object for managing contacts instead of using the standard Contact object.
- **Complex Visualforce Pages:** Developing complex Visualforce pages where standard Salesforce UI could have been used or a simpler Lightning component could have sufficed.
- **Unnecessary Code:** Writing code for automation where declarative (clicks-not-code) solutions like Flow or Approval Processes could have been used.
- **Custom Interface:** Building a custom interface or custom navigation on top of Salesforce, causing users to lose the familiarity of the Salesforce UI and causing potential confusion.

Common Salesforce Anti-Patterns

1. **The Hero:** The "Hero" anti-pattern in Salesforce, and in the software development world more broadly, refers to a situation where a single person (the "Hero") takes on a disproportionate amount of the work, often handling complex problems or emergencies single-handedly.
2. **God Object:** This refers to an object that knows too much or does too much. The god object is an example of an anti-pattern because it is a large, complex object that is difficult to maintain and understand.
3. **Spaghetti Code:** This refers to unstructured and difficult-to-maintain source code. Spaghetti code is often the result of several rounds of edits by different developers, with each developer making changes without fully understanding the existing code.
4. **Golden Hammer:** This is the assumption that a favorite solution is universally applicable. This is an anti-pattern because it discourages developers from finding the best solution for a particular problem.
5. **Cargo Cult Programming:** This refers to the practice of using patterns and methods without understanding why. It can lead to unnecessary code complexity and can make the code hard to maintain or modify.
6. **Design by Committee:** In this anti-pattern, a system is designed by a group of people with no clear leader, resulting in a system that doesn't satisfy the requirements of any individual user or is too complex to be practical.

The “Hero”

While this might seem beneficial in the short term, it can lead to serious issues, including:

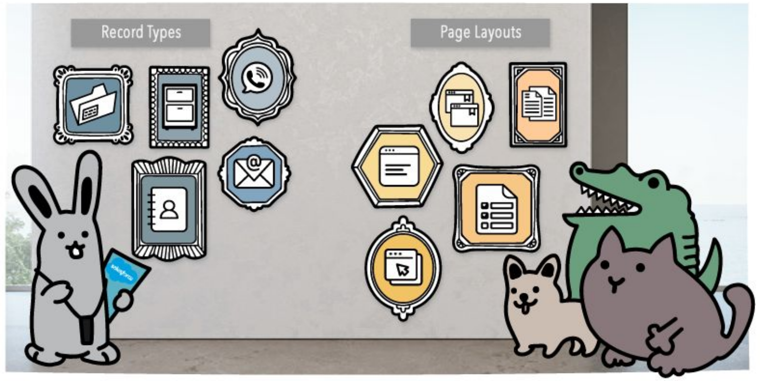
- **Knowledge Silo:** If only one person understands certain parts of the system, this can be problematic if they leave the organization or are otherwise unavailable.
- **Over-reliance:** A team that overly depends on one individual may struggle to handle issues or make progress when that individual is not available.
- **Burnout:** The "Hero" individual is at high risk of burnout due to the excessive workload and responsibility.
- **Inhibition of Team Growth:** Other team members may have less opportunity to learn and grow if the "Hero" is always taking on the challenging tasks.

The way to mitigate this anti-pattern is to foster a collaborative team environment where knowledge is shared, tasks are distributed fairly, and all team members are given the opportunity to develop and contribute. This can be done through pair programming, code reviews, regular knowledge sharing sessions, and good documentation practices.

The "God Object"

The "God Object" anti-pattern refers to an object that knows too much or does too much. This anti-pattern is not specific to Salesforce, but can be applied to any object-oriented system. In Salesforce, this could manifest as a single object (or class) that has too many responsibilities or dependencies.

In this scenario, the "God Object" might control too many aspects of the system, resulting in a large, complex, and tightly coupled class. This makes it hard to maintain, test, and understand. Additionally, it can lead to a lack of modularity in the system and inhibit reuse of code.



To avoid this anti-pattern, it's best to adhere to principles like Single Responsibility Principle (SRP) and separation of concerns. This means each object or class should have a single responsibility, and the functionality of the system should be distributed across multiple, smaller classes. This way, if a change is required, it can be done in a localized manner without impacting a large part of the system. This design also makes the system easier to understand, test, and maintain.

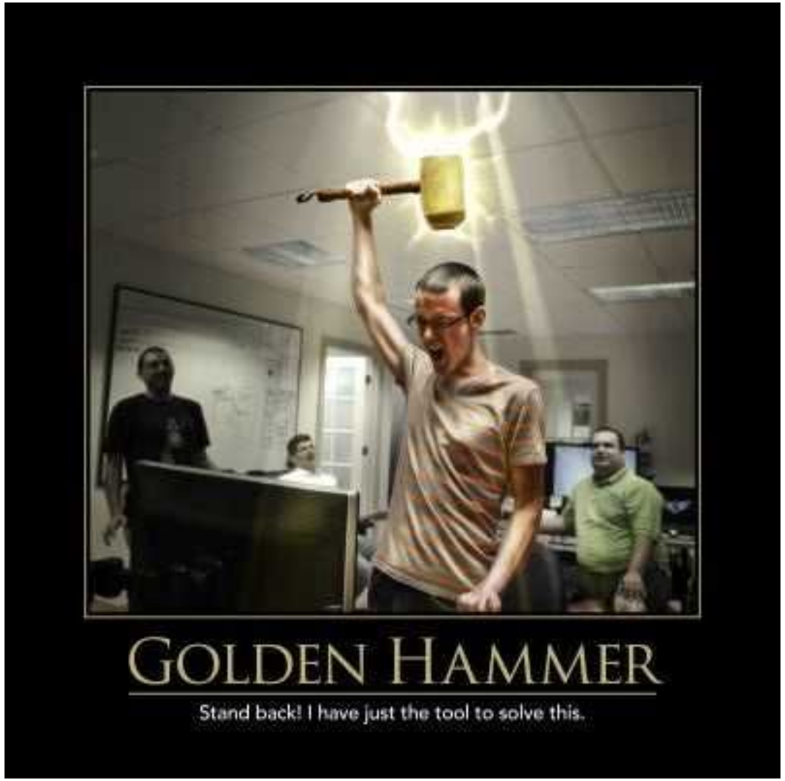
The "Golden Hammer"

The "Golden Hammer" is an anti-pattern that occurs when a development team uses a particular technology or methodology (the "Golden Hammer") to solve all kinds of problems, regardless of whether it's the best fit for the situation. The saying associated with this anti-pattern is: **"if all you have is a hammer, everything looks like a nail."**

In the context of Salesforce, this could manifest in several ways. For instance:

- **Overusing Apex Code:** Salesforce provides a variety of tools to customize and extend its functionality, such as workflows, process builder, and declarative tools. However, if a team defaults to using Apex code for every customization, they may end up with a more complex, harder to maintain system, and may miss out on some of the benefits of the platform's declarative features.
- **Over-reliance on a Particular Feature:** A team might overuse a particular feature, like Process Builder or Flow, even when it's not the most efficient or effective tool for the job. This could lead to performance issues, maintainability challenges, and could make the system harder to understand.

To avoid the Golden Hammer anti-pattern, it's important for development teams to have a **good understanding of all the tools and features at their disposal**, and to **choose the right** one based on the **specific requirements and context** of each problem they are trying to solve. This often requires a good understanding of Salesforce's capabilities, as well as ongoing learning as the platform continues to evolve and introduce new features.



"Cargo Cult Programming"

"Cargo Cult Programming" is an anti-pattern that can occur in any programming environment, including Salesforce. It refers to the practice of using code or programming techniques without understanding how they work. The term is derived from the historical phenomenon of "cargo cults", where indigenous people created makeshift runways and airports, hoping to attract airplanes and their cargo, without understanding the real reasons why planes landed.

In the context of Salesforce, this could manifest in several ways:

1. **Copy-Pasting Code:** A developer might copy and paste code from Salesforce's documentation, Stack Overflow, or other resources, and use it in their application without understanding what the code does. While this might work in the short term, it can lead to issues later on if the code has side effects or doesn't work as expected in a different context.
2. **Using Patterns Blindly:** A developer might use design patterns or practices they've seen used elsewhere, without understanding why those patterns are used. For example, they might overuse triggers or use bulkification patterns even when they're not necessary.
3. **Overusing Certain Features:** Just because a feature or functionality is available, doesn't mean it's always the right tool for the job. For instance, overuse of Process Builder or Flow without understanding their limitations and impact on system performance can lead to problems.

The solution to avoid Cargo Cult Programming is education and understanding. Developers should strive to understand how and why certain code works, the reason behind specific patterns, and the implications of using certain features. Code reviews, pair programming, and continued learning can help foster this understanding.

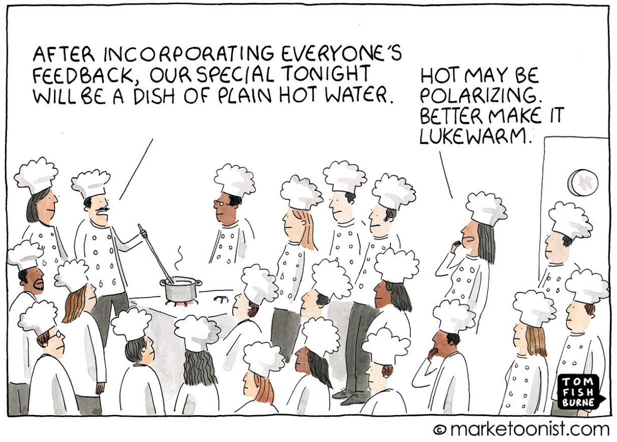
“Design by Committee”

"Design by Committee" is an anti-pattern that can occur in any software development project, including those involving Salesforce. This anti-pattern happens when a group (the committee) is responsible for making design decisions, often leading to a lack of clear direction and focus, and resulting in a product that tries to please everyone but might not meet any of the intended needs particularly well.

In the context of Salesforce, here's how this anti-pattern could manifest:

1. **Too Many Opinions:** When a Salesforce implementation involves many stakeholders from different departments, each may have their own ideas and requirements. If every suggestion is incorporated without a clear strategy, the result can be a Salesforce org that is overly complicated, difficult to navigate, and hard to maintain.
2. **Lack of Ownership:** If everyone is responsible for decisions, then no one is truly accountable. This can lead to slow decision-making, inadequate attention to details, and challenges in driving the project forward.
3. **Compromised Design:** Trying to accommodate all viewpoints might lead to a compromised design, with features that are not well integrated with each other, creating a system that doesn't effectively address the core business needs.

To avoid the "Design by Committee" anti-pattern, it's crucial to have clear decision-making processes in place. While it's important to get input from all stakeholders, final design decisions should typically rest with a smaller, dedicated team or individual. This team or individual can take into account all the input, but also make decisive choices that are aligned with the overall strategy and goals of the project. This approach also helps ensure accountability and clear lines of responsibility.

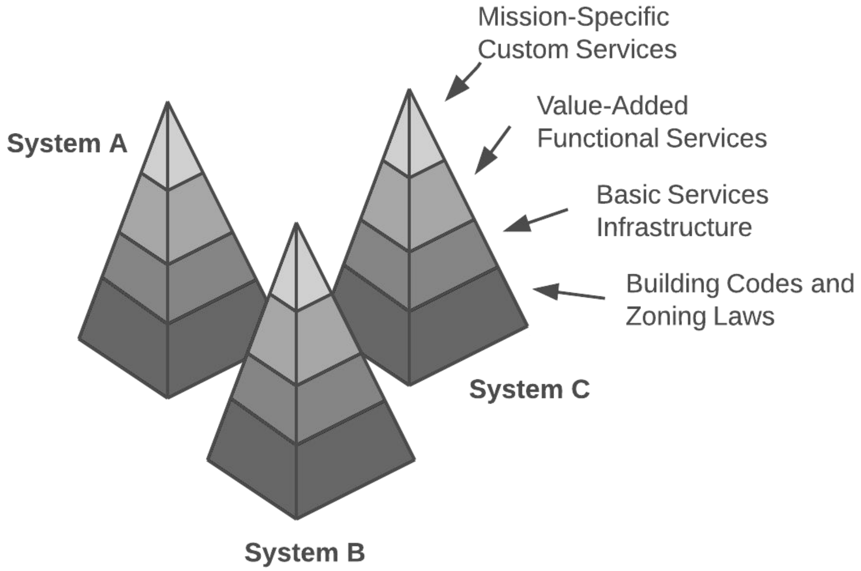

AFTER INCORPORATING EVERYONE'S FEEDBACK, OUR SPECIAL TONIGHT WILL BE A DISH OF PLAIN HOT WATER.
HOT MAY BE POLARIZING. BETTER MAKE IT LUKEWARM.
TOM FISH BURNE
© marketoonist.com

Stovepipe Anti-pattern

The stovepipe or silo anti-pattern refers to a situation where several systems or subsystems have their own specific designs and do not interact with each other, much like stovepipes or silos that are tall, isolated, and stand apart.

This anti-pattern can occur in a variety of scenarios, such as software development, business processes, or organizational structures. In software development, for instance, it might occur when various components of a system are developed independently and without considering how they should interoperate.

The stovepipe anti-pattern often leads to inefficiencies and a lack of synergy, since the same work may be duplicated in different silos, and potential benefits of integration and interoperability are missed. It can also make troubleshooting and system enhancements more challenging, since changes in one area may have unforeseen effects on others.


Mission-Specific Custom Services
Value-Added Functional Services
Basic Services Infrastructure
Building Codes and Zoning Laws
System A
System C
System B

“Big Ball of Mud” anti-pattern

The "Big Ball of Mud" is a software engineering anti-pattern that refers to a system with no discernable architecture. In such a system, the structure is haphazard and lacks organization, often as a result of continuous changes over a significant period of time.

This type of system is often the result of several factors, including:

- Continuous development without a clear plan
- Deadlines forcing developers to choose quick, easy solutions over well-architected ones
- Lack of documentation or lack of understanding of the existing design
- Inadequate refactoring, leading to code becoming increasingly complex and difficult to maintain

While a "Big Ball of Mud" might function and even meet the needs of the users, it's likely to be costly and challenging to maintain or scale. Furthermore, it increases the likelihood of introducing bugs because it's harder to understand the interactions between different parts of the system. It's often used as an example of what to avoid in software design.

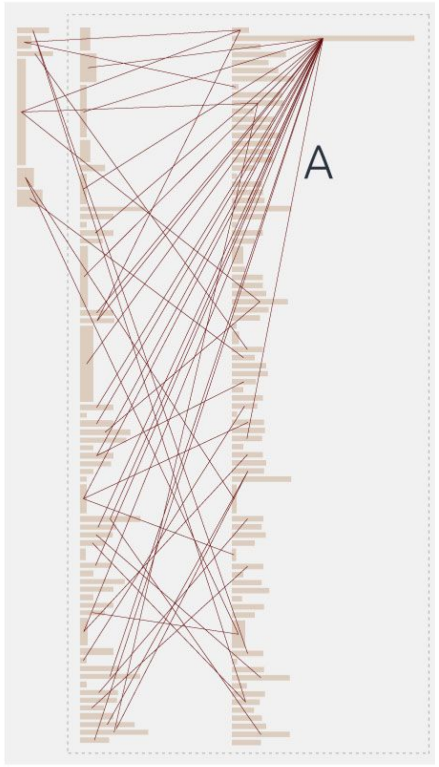
Intense coupling anti-pattern

The Intense Coupling anti-pattern in software design refers to a situation where modules or components are too tightly linked to each other. In a tightly coupled system, individual modules are highly dependent on each other, often because they directly manipulate the data and methods of other modules.

This intense coupling leads to several issues:

1. Lack of Flexibility: Changes to one module could impact multiple other modules, making the system hard to modify or extend.
2. Harder to Understand: Understanding any single module requires knowledge of its dependencies, making the system difficult to understand and maintain.
3. Harder to Test: Testing individual modules can be challenging because each one operates closely with others.

The opposite of this is loose coupling, which is generally favored because it makes components easier to test, understand, and modify. However, achieving loose coupling requires careful planning and design, especially when the software is expected to grow in complexity over time.



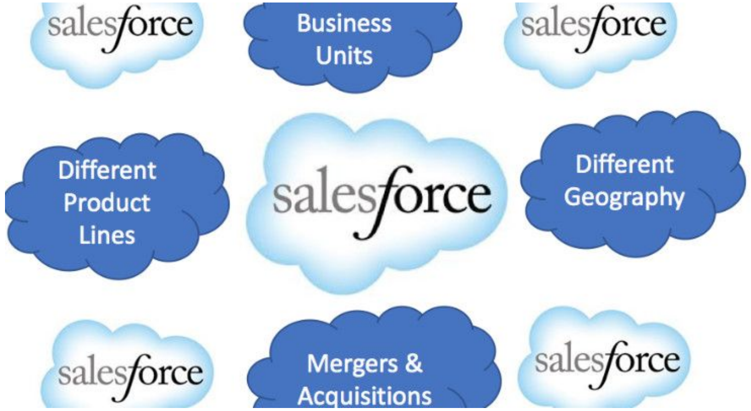
Ungoverned Org Proliferation

Ungoverned Org (Organization) Proliferation is a term used to describe a situation where multiple Salesforce environments (orgs) have been created without proper planning, oversight, or governance. This anti-pattern is usually seen in large organizations where different departments or teams may independently create their own Salesforce orgs to meet their specific needs.

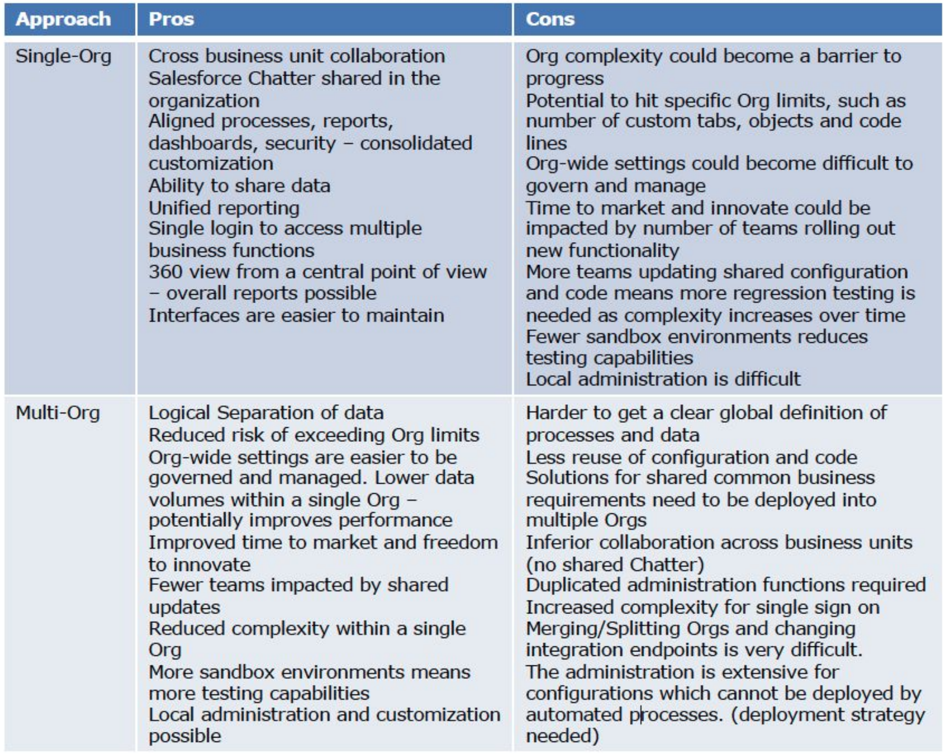
Issues that arise from ungoverned org proliferation include:

1. **Data Silos:** Important data may be stored separately in different orgs, making it hard to get a consolidated view of business information.
2. **Inefficiency and Waste:** Multiple orgs can lead to redundancy in development and administrative efforts as the same functionality may be created in different orgs. Licensing costs can also increase unnecessarily.
3. **Inconsistency:** With separate orgs, there might be inconsistencies in processes and data structures, leading to confusion and errors.
4. **Security Risks:** With many orgs, it's harder to enforce consistent security policies and controls.

To address this anti-pattern, organizations should adopt a robust governance strategy for Salesforce, which includes establishing clear policies for when and how new orgs can be created, and having a centralized architecture review board to enforce these policies. In many cases, Salesforce's various multi-tenant features, like sandboxes and namespaces, can be used to avoid the need for multiple separate orgs.



Org Strategy Cheatsheet



Approach	Pros	Cons
Single-Org	Cross business unit collaboration Salesforce Chatter shared in the organization Aligned processes, reports, dashboards, security – consolidated customization Ability to share data Unified reporting Single login to access multiple business functions 360 view from a central point of view – overall reports possible Interfaces are easier to maintain	Org complexity could become a barrier to progress Potential to hit specific Org limits, such as number of custom tabs, objects and code lines Org-wide settings could become difficult to govern and manage Time to market and innovate could be impacted by number of teams rolling out new functionality More teams updating shared configuration and code means more regression testing is needed as complexity increases over time Fewer sandbox environments reduces testing capabilities Local administration is difficult
Multi-Org	Logical Separation of data Reduced risk of exceeding Org limits Org-wide settings are easier to be governed and managed. Lower data volumes within a single Org – potentially improves performance Improved time to market and freedom to innovate Fewer teams impacted by shared updates Reduced complexity within a single Org More sandbox environments means more testing capabilities Local administration and customization possible	Harder to get a clear global definition of processes and data Less reuse of configuration and code Solutions for shared common business requirements need to be deployed into multiple Orgs Inferior collaboration across business units (no shared Chatter) Duplicated administration functions required Increased complexity for single sign on Merging/Splitting Orgs and changing integration endpoints is very difficult. The administration is extensive for configurations which cannot be deployed by automated processes. (deployment strategy needed)

Spaghetti Sharing Model

The "Spaghetti Sharing Model" is a term used to describe a highly complex and disorganized sharing structure in Salesforce. Salesforce is built to be highly customizable, allowing different levels of access and sharing settings to be applied to data. This can become very complicated when not managed correctly, leading to a sharing model that is convoluted and difficult to manage, hence the term "Spaghetti".

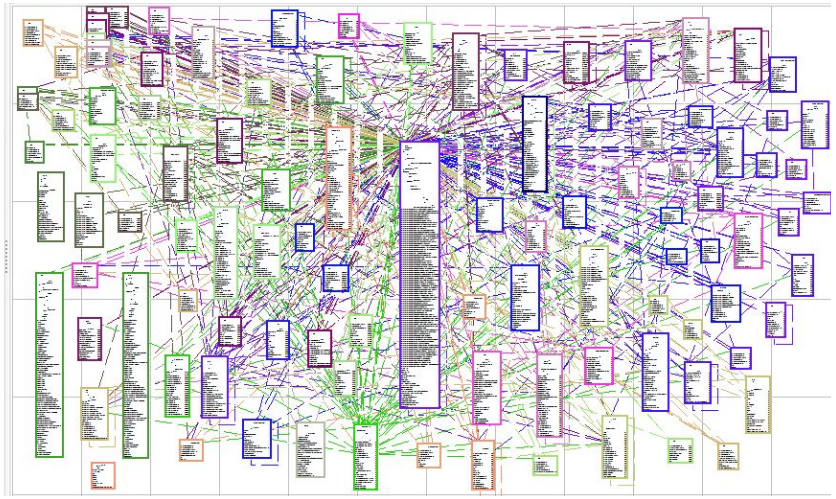
This anti-pattern can occur when:

- Sharing rules are created ad-hoc without a clear strategy or understanding of the overall sharing architecture.
- There is an over-reliance on manual sharing or "sharing rules" as opposed to using "role hierarchy" or "organization-wide default" settings to control access.
- The sharing model is continually modified over time without reviewing or cleaning up old and possibly obsolete rules.

The Spaghetti Sharing Model can lead to issues such as:

Difficulty in managing and understanding who has access to what data.
Performance issues, as a large number of sharing rules can slow down data access.
Potential security risks, if the complex sharing setup leads to data being inadvertently exposed to the wrong users.

The best way to avoid this anti-pattern is to plan the sharing model carefully from the start, leveraging Salesforce's inbuilt tools like role hierarchy, profiles, and permission sets to keep sharing as simple and clear as possible. Regular reviews and clean-ups of sharing settings can also help to prevent this situation.



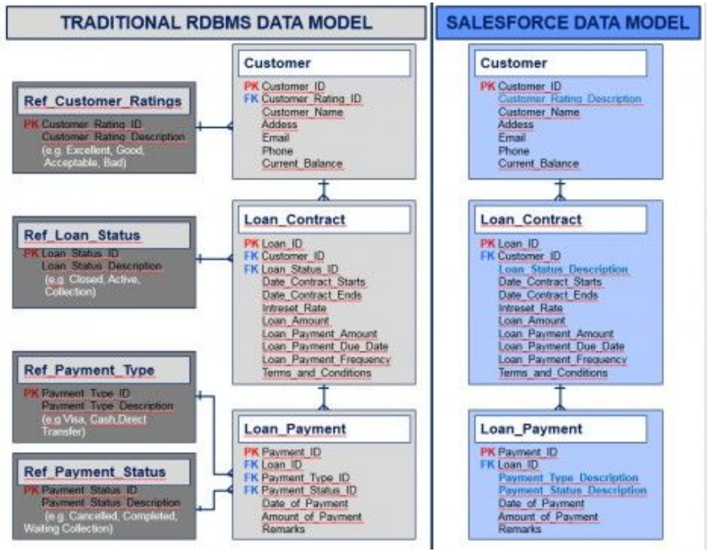
Salesforce is <u>not</u> a relational Database

Salesforce uses a multi-tenant architecture where all users and applications share a single, common infrastructure and code base. At the heart of this architecture is a proprietary database management system (DBMS) designed by Salesforce.

This database is not directly accessible in the way traditional relational databases like MySQL or PostgreSQL are. Instead, interactions with the database are done through Salesforce's APIs, or through the use of SOQL (Salesforce Object Query Language) and SOSL (Salesforce Object Search Language) for querying data.

This architecture allows Salesforce to efficiently manage and scale resources across many users and to seamlessly deliver updates and new features without disrupting service. It also means that each customer does not need to maintain their own individual database, reducing overhead and complexity.

Please note that while Salesforce's underlying database is not directly accessible, the data model is similar to that of a relational database, with tables represented as objects and rows as records.



Salesforce Unique Differentiators

While Salesforce's data model has similarities with a relational database – in that it includes objects (like tables), fields (like columns), and records (like rows) – there are several reasons why thinking of Salesforce strictly as a traditional relational database could lead to misunderstandings and issues:

- **Data Access:** In a traditional relational database, developers have direct SQL access to the database and can perform any CRUD operation. However, in Salesforce, interaction with the data is controlled via Salesforce's user interface, APIs, or through its own query languages (SOQL and SOSL), which have limitations and differences compared to SQL.
- **Multi-Tenancy:** Salesforce uses a multi-tenant architecture, which means that all customers are using the same infrastructure. This has implications for data storage, isolation, and application performance. In contrast, with traditional relational databases, you usually have control over your own dedicated infrastructure.
- **Governor Limits:** Salesforce imposes certain usage limits, known as Governor Limits, to ensure that system resources are used efficiently and prevent monopolization of shared resources. These limits can affect how much data you can read, write, or process. Such restrictions typically don't exist with traditional databases.
- **Customization and Configuration:** Salesforce is essentially a platform that is highly customizable via configuration. This is unlike a traditional database where a lot of functionality would need to be built via programming.
- **Database Maintenance:** Salesforce handles all of the backend infrastructure, including the database management. This means that tasks like scaling, performance tuning, and backups are taken care of by Salesforce, whereas with a traditional relational database, these tasks are typically handled by the organization's database administrator or devops team.

So while it's fine to think of Salesforce's data structure in relational terms to understand its schema and how its data is organized, it's crucial to remember these key differences in how the data is accessed, the limitations that are in place, and how the platform is managed.

Poor Data Management

Poor data management in Salesforce refers to practices that result in inaccurate, inconsistent, or incomplete data. This can occur due to a lack of proper data governance policies or procedures, inadequate training, or insufficient use of Salesforce's data management tools.

Here are some examples of poor data management practices in Salesforce:

- **Lack of Duplicate Management:** Without proper duplicate management rules in place, it's easy to create duplicate records in Salesforce. For example, the same contact or account might be entered twice with slight differences in the name, leading to confusion and inaccurate data.
- **Inconsistent Data Entry:** This can occur when there's no standard procedure for entering data. For example, different users might use different formats for phone numbers or addresses, leading to inconsistent data.
- **Failure to Use Validation Rules:** Validation rules in Salesforce help ensure the accuracy and consistency of data by enforcing certain conditions on the data input. If these are not used, users can enter incorrect or inconsistent data.
- **Lack of Regular Data Cleanup:** Over time, old, inaccurate, or irrelevant data can accumulate in Salesforce. If regular data cleanup activities aren't scheduled, this can lead to large amounts of unnecessary data, which can affect system performance and reporting accuracy.
- **Improper Use of Import Tools:** Salesforce provides tools for importing data from external sources. If these tools are not used properly, for example, if data is not properly mapped to the correct fields, this can lead to inaccurate data.
- **Not Defining Ownership of Records:** If it's not clear who owns certain records, those records might not get updated properly, leading to stale or inaccurate data.

Each of these examples can have serious impacts on an organization, affecting everything from productivity and efficiency to the accuracy of reporting and decision-making.

Poor Data Management - Consequences

Drawbacks of poor data management in Salesforce can be detrimental for an organization. These can include:

1. **Inaccurate Reporting:** Poor data management can lead to inaccurate reporting, as the reports and dashboards in Salesforce are only as good as the data they are based on. This can lead to misleading insights and poor decision making.
2. **Decreased User Adoption:** Users may become frustrated if they constantly encounter incorrect or duplicate data, which can lead to decreased user adoption of the Salesforce platform.
3. **Decreased Productivity:** When data is not managed properly, users may need to spend extra time cleaning data, finding the correct records, or dealing with duplicates, which decreases productivity.
4. **Compromised Customer Relationships:** With poor data, communication with customers can suffer. For example, duplicate records could result in customers receiving the same communication multiple times, damaging the customer relationship.
5. **Regulatory Compliance Issues:** Poor data management can also lead to compliance issues, especially in regulated industries, as it could result in the inability to accurately track interactions or report on activities as required by law.

Case Study or Examples:

- **Duplicate Data Example:** An organization didn't set up duplicate rules in Salesforce, leading to several duplicate accounts. This resulted in confusion and inefficiency as sales reps would often reach out to the same account, not realizing someone else was already working with them.
- **Inconsistent Data Example:** A company didn't have standard procedures for data entry. Some users entered phone numbers with the country code, while others didn't. When a global marketing campaign was launched, the messages failed to send to the numbers without country codes, impacting the campaign's effectiveness.
- **Lack of Data Cleanup Example:** An organization didn't schedule regular data cleanup activities. Over time, old and inaccurate data built up in the system, slowing down performance and causing issues with data storage limits.

Addressing these data management issues is crucial to maintaining an effective and efficient Salesforce environment.

Excessive Use of Code

Excessive use of code in Salesforce refers to using Apex, Visualforce, or Lightning Component code to build solutions that could have been achieved using Salesforce's declarative, "clicks-not-code" tools. While there's a place for code in complex scenarios or where standard functionality falls short, excessive use can add unnecessary complexity, making the system harder to maintain and upgrade.

Here are some examples of excessive use of code:

1. **Trigger for Field Update:** Writing an Apex trigger to update a field when it could have been achieved through declarative automation.
2. **Visualforce for Custom Pages:** Using Visualforce to create custom pages when the same result could have been achieved using standard page layouts or Lightning App Builder.
3. **Apex for Record Creation:** Using Apex to create new records triggered by certain conditions, when a flow or process builder could have done the same job.
4. Apex for Validation: Creating complex validation through Apex code, when the same could have been managed through Validation Rules.
5. **Custom Security Code:** Writing custom Apex code to manage record access instead of leveraging built-in security features like roles, profiles, sharing rules, and permission sets.
6. **Apex for Simple Calculations:** Utilizing Apex for simple field calculations that could be managed with Formula Fields.

In each of these cases, an over-reliance on code can make the Salesforce environment more complex and harder to maintain, and it might require more specialized knowledge to troubleshoot or modify. As a best practice, Salesforce recommends utilizing its powerful declarative tools wherever possible, and saving code for complex situations where the built-in tools aren't sufficient.

Excessive Use of Code - Consequences

The excessive use of code in Salesforce can lead to several drawbacks:

1. **Increased Maintenance:** Code requires ongoing maintenance and testing. With every Salesforce update (3 times a year), all custom code should be tested to ensure compatibility, which adds to the maintenance overhead.
2. **Decreased Agility:** With excessive code, modifications become more complex and time-consuming. This reduces the organization's agility to make changes or implement new business processes quickly.
3. **Increased Costs:** More code means more developers and more time to build, test, and maintain, which can increase costs.
4. **Decreased User Adoption:** Overly complex customizations may not be intuitive to end users, leading to decreased user adoption.
5. **Limited Scalability:** Excessive code can affect the system's scalability. As business needs grow, complex code could be harder to adapt.

Case Study or Examples:

- **Complex Triggers Example:** A company built multiple complex Apex triggers for record updates and validations. With each Salesforce update, these triggers caused issues and required extensive testing and debugging. The maintenance overhead increased considerably over time, leading to higher costs and slower response to business needs.
- **Custom UI Example:** An organization built a custom user interface with Visualforce. While initially meeting their unique needs, over time the UI became hard to update and failed to leverage Salesforce's UI improvements. This lead to decreased user satisfaction and adoption.
- **Custom Security Code Example:** A company used custom Apex code to manage record access instead of using standard Salesforce security features. This not only made it difficult to manage access controls, but also put the system at risk of potential security breaches due to coding errors.

To mitigate these risks, it's best to use Salesforce's declarative, "clicks-not-code" tools whenever possible, and only resort to code when necessary for complex requirements that cannot be met by the standard functionality.

Not Using Roles and Profiles Appropriately

Roles and profiles are key components of Salesforce's security model that help control user access to data.

1. **Roles:** In Salesforce, a role controls a user's visibility into the organization's data. Roles are arranged in a hierarchy, meaning that users higher up in the hierarchy have access to records owned by users lower down in the hierarchy.
2. **Profiles:** A profile in Salesforce controls the permissions a user has to perform different functions within the organization's Salesforce environment. This includes access to specific objects, fields, and record types, as well as system permissions like the ability to export data.

Improper use of roles and profiles can lead to a number of issues. Examples include:

- **Overly Broad Profiles:** Creating profiles that give users access to more data or functions than they need to perform their jobs can lead to issues. For example, if all users are given a profile that allows them to export data, this could lead to unauthorized data exports.
- **Ignoring Role Hierarchy:** Not utilizing or incorrectly setting up the role hierarchy could lead to data visibility issues. For instance, a manager might not have visibility into the records owned by their direct reports if the role hierarchy isn't properly configured.
- **Using Profiles Instead of Roles:** Using profiles to control record visibility instead of roles can create unnecessary complexity and might not properly restrict data access. For example, creating different profiles for users in different regions instead of using roles to control record visibility.
- **Not Regularly Updating Roles and Profiles:** If roles and profiles are not updated as user responsibilities change, this could lead to users having incorrect data access. For instance, if a user changes roles within the company but their Salesforce role and profile aren't updated, they might have access to data they shouldn't, or they might not have access to data they now need.

Properly managing roles and profiles is critical for maintaining data security and ensuring users can perform their jobs effectively.

Not Using Roles and Profiles Appropriately - Consequences

Improper use of roles and profiles in Salesforce can have several serious consequences, including:

1. **Data Leakage:** If users are given more access than necessary, sensitive data could be viewed or exported by unauthorized users.
2. **Operational Inefficiency:** If users don't have access to the data they need, they may have to request access frequently, leading to operational inefficiency.
3. **Lack of Accountability:** When roles and profiles are not well defined, it can be hard to track who did what, leading to a lack of accountability.
4. **Increased Risk of Errors:** If users have access to functions or data they are not familiar with, it can lead to increased mistakes and data integrity issues.

Case Study or Examples:

- **Data Leakage Example:** A company gave all its users the 'System Administrator' profile in Salesforce to avoid having to manage different profiles. This led to a situation where a user inadvertently modified a critical setting, leading to a disruption in service.
- **Operational Inefficiency Example:** In another company, the role hierarchy was not set up correctly, leading to managers not being able to see their direct reports' records. This resulted in frequent requests to the system administrator to manually share records, leading to a lot of wasted time.
- **Lack of Accountability Example:** A company had a single profile for all users and did not utilize roles. As a result, when a record was modified inappropriately, it was challenging to identify who made the change, as everyone had the same level of access.
- **Increased Risk of Errors Example:** A user was given a profile that allowed them to modify object layouts even though this was not part of their job. The user inadvertently removed a critical field from the layout, causing confusion and errors until the issue was identified and corrected.

These examples highlight the importance of properly defining and managing roles and profiles in Salesforce. It's critical to balance providing users with the access they need to do their jobs effectively while minimizing the risk of unauthorized data access or accidental changes to system configurations.

Ignoring Ongoing Salesforce Training

Ongoing Salesforce training is crucial because Salesforce is a complex and powerful tool that is constantly being updated and enhanced. Training ensures that users are able to effectively use the platform, understand new features and updates, and adhere to best practices for data management and security.

Lack of ongoing Salesforce training can lead to several problems:

1. **Lowered Productivity:** Without proper training, users may not understand how to use Salesforce efficiently, leading to reduced productivity.
2. **Decreased User Adoption:** If users find Salesforce confusing or difficult to use due to lack of training, they may be less likely to use it, leading to decreased user adoption.
3. **Increased Errors:** Users who are not properly trained may make more mistakes, such as creating duplicate records or entering data inconsistently, leading to data integrity issues.
4. **Underutilization of Features:** Without ongoing training, users may not be aware of or understand how to use new features, leading to underutilization of Salesforce's capabilities.
5. **Security Risks:** If users aren't trained on Salesforce's security features and best practices, they may inadvertently put data at risk, for instance, by sharing sensitive data with the wrong users or groups.

Examples of problems caused by lack of training:

- **Lowered Productivity Example:** Without training, users at a company were unsure of how to efficiently search for records in Salesforce. As a result, they spent unnecessary time scrolling through records, leading to lowered productivity.
- **Decreased User Adoption Example:** In another organization, users found Salesforce confusing due to lack of training. This resulted in decreased usage of the platform, causing the company to not fully benefit from their investment in Salesforce.
- **Increased Errors Example:** Without proper training on how to enter and update data, users at a company entered data inconsistently. This led to confusion and errors when analyzing or reporting on the data.
- **Underutilization of Features Example:** A company did not provide ongoing training to its users. As a result, when Salesforce rolled out a new feature that could have improved their processes, users were not aware of it and did not know how to use it, leading to underutilization of the platform's capabilities.

Ongoing Salesforce training helps ensure that users can effectively and efficiently use the platform, leading to higher productivity, better data integrity, and a better return on the company's Salesforce investment.

Ignoring Ongoing Salesforce Training - Consequences

Ignoring ongoing Salesforce training can lead to a variety of drawbacks including:

1. **Decreased Efficiency:** Without proper understanding of Salesforce features and functionalities, users can take more time to accomplish tasks, reducing overall efficiency.
2. **Lower User Adoption:** If users find Salesforce challenging to use or don't understand how it can benefit their work, they might resist adopting it, causing a decline in user adoption rates.
3. **Poor Data Quality:** Lack of training can lead to inconsistent data entry, resulting in poor data quality, which affects reporting and decision making.
4. **Underutilization of Salesforce Features:** Salesforce regularly releases updates with new features and enhancements. Without ongoing training, users may not be aware of or understand how to use these features, leading to underutilization.
5. **Increased Support Costs:** Users who are not well-trained may require more support, which can increase the cost of maintaining the Salesforce platform.

Case Study or Examples:

- **Decreased Efficiency Example:** An organization did not provide sufficient training on using Salesforce's report and dashboard features. As a result, users spent a significant amount of time manually compiling data for reports, reducing overall efficiency.
- **Lower User Adoption Example:** At another company, users were not trained on how to use Salesforce's task and event management features. As a result, they continued to use their old systems for managing tasks and events, leading to lower Salesforce user adoption.
- **Poor Data Quality Example:** Due to a lack of training on data entry standards, users at a company entered data inconsistently. This led to poor data quality, making it challenging to generate accurate reports and insights.
- **Increased Support Costs Example:** Without proper Salesforce training, users at an organization frequently needed help from the IT department to perform basic tasks, leading to increased support costs.

By investing in ongoing Salesforce training, organizations can maximize their return on investment, increase user adoption, improve data quality, and decrease support costs.

Neglecting User Experience

The user experience in Salesforce is crucial as it directly impacts how effectively and efficiently users can perform their tasks within the platform. A good user experience can lead to higher productivity, better data quality, and increased user adoption, while a poor user experience can lead to the opposite.

Examples of poor user experience in Salesforce could include:

1. **Complex User Interface:** If the Salesforce interface is overly complicated with too many fields, buttons, or unrelated items, it can confuse users and reduce productivity.
2. **Inadequate Customization:** If Salesforce is not customized to match the users' specific workflows and needs, it can lead to inefficiencies and frustration.
3. **Poorly Defined Processes:** Without well-defined and automated business processes, users may need to perform unnecessary manual tasks, leading to wasted time and potential errors.
4. **Lack of Mobile Optimization:** If Salesforce isn't optimized for mobile use, it can frustrate mobile users and limit their ability to work effectively when away from their desks.
5. **Slow Performance:** If Salesforce pages load slowly, it can interrupt users' workflows and decrease productivity.
6. **Inconsistent Design:** If different parts of the Salesforce interface have inconsistent designs or workflows, it can confuse users and lead to mistakes.
7. **Lack of Training:** Without sufficient training, users might find Salesforce difficult to use, leading to decreased adoption and productivity.

Examples in bullet points:

- A company didn't remove unnecessary fields from their Salesforce page layouts, leading to a complex and confusing interface that reduced user productivity.
- An organization did not customize Salesforce to match its specific business processes, causing users to waste time navigating through irrelevant steps.
- A company did not automate a recurring manual task in Salesforce, leading to wasted time and potential errors.
- Another organization did not optimize its Salesforce interface for mobile use, leading to frustration among its field sales reps who primarily used mobile devices.
- A company experienced slow Salesforce performance due to an excess of complex customizations, causing interruptions and inefficiencies in users' workflows.

Improving the user experience in Salesforce typically involves simplifying and customizing the interface, automating business processes, optimizing for mobile use, ensuring good performance, providing consistent design, and offering comprehensive user training.

Neglecting User Experience - Consequences

Neglecting user experience in Salesforce can lead to several significant drawbacks:

1. **Decreased User Adoption:** If users find Salesforce difficult to use, they may resist using it, leading to lower user adoption rates.
2. **Lowered Productivity:** Poor user experience can make tasks more difficult and time-consuming, which can reduce productivity.
3. **Higher Error Rates:** If the user interface is confusing or complex, users may make more mistakes, leading to data integrity issues.
4. **Increased Training Costs:** If the user interface is not intuitive, more time and resources may be needed for training.
5. **Reduced ROI:** Overall, a poor user experience can lead to a lower return on investment (ROI) for Salesforce, as it is not being used to its full potential.

Case Study or Examples:

- **Decreased User Adoption Example:** A company did not put enough effort into customizing Salesforce to their specific business processes, which led to a difficult and confusing user experience. As a result, many users resisted adopting the platform, sticking to their old systems and methods.
- **Lowered Productivity Example:** In another company, the Salesforce interface was cluttered and complex, making it difficult for users to quickly find the information they needed. This led to a significant decrease in productivity as users spent unnecessary time navigating the system.
- **Higher Error Rates Example:** Due to a confusing and non-intuitive interface, users in a certain organization made frequent mistakes in data entry, leading to data integrity issues that took significant time and effort to correct.
- **Increased Training Costs Example:** A company neglected the user experience in Salesforce, resulting in a steep learning curve for users. This led to higher training costs as users required more in-depth and continuous training.

Focusing on user experience during the design and customization of Salesforce, as well as providing ongoing user experience improvements, can help to avoid these issues and ensure that the organization gets the maximum benefit from the platform.

Skipping Salesforce Updates

Salesforce updates are crucial because they introduce new features, improvements, and security enhancements. Staying current with these updates helps ensure your Salesforce instance is secure, efficient, and optimized with the latest functionalities.

Skipping Salesforce updates can lead to several problems:

1. **Missed New Features:** Each Salesforce update introduces new features and improvements. If updates are skipped, users may miss out on these enhancements, limiting their ability to take full advantage of the Salesforce platform.
2. **Security Vulnerabilities:** Updates often include patches for security vulnerabilities. Ignoring updates can leave your Salesforce instance exposed to these vulnerabilities.
3. **Software Incompatibility:** Over time, other software that integrates with Salesforce may become incompatible if Salesforce is not kept up-to-date.
4. **Limited Support:** Salesforce only provides full support for current versions. If updates are skipped, you may find that you have limited access to Salesforce support.

Examples of problems caused by skipping updates:

- **Missed New Features Example:** A company skipped several Salesforce updates and as a result, they missed out on new features that could have improved their sales processes and reporting capabilities.
- **Security Vulnerabilities Example:** Another company ignored a security update that patched a known vulnerability. Unfortunately, their Salesforce instance was later compromised using this vulnerability.
- **Software Incompatibility Example:** A third company neglected to update their Salesforce instance, which eventually led to compatibility issues with their marketing automation software. This resulted in significant disruption to their marketing campaigns.
- **Limited Support Example:** Another organization, after skipping several updates, encountered a technical issue with their Salesforce instance. When they contacted Salesforce support, they were told that their version was no longer fully supported, and they needed to update to receive full assistance.

Keeping your Salesforce instance up-to-date helps ensure you're benefiting from the latest features and improvements, maintaining a secure environment, ensuring compatibility with other software, and retaining access to full support from Salesforce.

Skipping Salesforce Updates - Consequences

Skipping Salesforce updates can have several negative consequences:

1. **Missing Out on New Features and Improvements:** Salesforce updates often introduce new features and enhancements that can improve user productivity and platform capabilities. Skipping these updates means missing out on these improvements.
2. **Security Risks:** Updates often include important security patches. Skipping these updates can leave your Salesforce instance vulnerable to potential security threats.
3. **Integration Issues:** Salesforce integrates with many different systems, and updates can affect these integrations. If you're not staying current with updates, you may experience integration issues.
4. **Limited Access to Support:** Salesforce provides support for its latest versions. If you're running an outdated version, you may have limited access to support.

Case Study or Examples:

- **Missing Out on New Features Example:** A company skipped several updates and missed out on new AI-powered features that could have greatly improved their sales forecasting.
- **Security Risks Example:** Another company didn't update their Salesforce instance for a long period, and it was later compromised due to a known vulnerability that had been patched in a newer update.
- **Integration Issues Example:** A company had integrated Salesforce with their ERP system. When they didn't upgrade Salesforce to the latest version, the integration began to experience issues, impacting the company's sales and inventory management processes.
- Limited Access to Support Example: **An organization that was running an old version of Salesforce encountered an issue. When they contacted Salesforce** for support, they were informed that their version was no longer supported, and they would need to update to receive assistance.

To avoid these issues, it's important to keep your Salesforce instance up-to-date with the latest updates and security patches. This will help ensure your organization can take full advantage of the latest features and improvements, maintain a secure environment, and receive the necessary support when required.

Best Practices to Avoid Anti-Patterns

Key best practices for Salesforce development:

1. **Leverage Built-in Features:** Before resorting to code, explore Salesforce's built-in features and configurations to meet your requirements. This adheres to Salesforce's "clicks before code" principle.
2. **Use Sandbox for Development:** Always use a Sandbox for development and testing changes before deploying to the production environment.
3. **Implement Version Control:** Use a version control system to track code changes and support collaboration between team members.
4. **Follow Naming Conventions:** Use clear and consistent naming conventions for all custom objects, fields, classes, and other components to make it easy to understand their purpose.
5. **Write Unit Tests:** Ensure that all Apex code is covered by unit tests, and aim for 100% code coverage.
6. **Optimize for Performance:** Keep performance in mind when developing, especially with regards to governor limits in Salesforce.
7. **Adhere to Security Best Practices:** Ensure that your customizations adhere to Salesforce's security best practices to protect your data.
8. **Document Code and Configurations:** Maintain proper documentation for code and configurations to make it easier for others (or future you) to understand.

Role of Continuous Learning and Training:

1. **Keep Up with Updates:** Salesforce has three major releases a year. Continuous learning is necessary to stay updated with new features and improvements.
2. **Improve Skills and Knowledge:** Continuous learning helps developers to improve their skills and knowledge, enabling them to build more efficient and effective solutions.
3. **Promote Best Practices:** Regular training sessions can be used to promote best practices and mitigate anti-patterns among the development team.
4. **Increase Efficiency:** The more you know about Salesforce, the more efficiently you can use it. Regular training can lead to increased efficiency.
5. **Support Career Growth:** For individuals, continuous learning and Salesforce certifications can support career growth and opportunities.
6. **Boost User Adoption:** Training for end-users is critical to boost user adoption and ensure they are using the platform effectively.

Continual learning and training are integral to a successful Salesforce implementation and are crucial for both developers and end-users.

The Role in Governance in Avoiding Anti-Patterns

The importance of governance in Salesforce:

1. **Ensures Consistency:** A strong governance framework ensures consistency across various Salesforce implementations, leading to a better user experience and data integrity.
2. **Maintains Quality:** Governance helps to maintain high quality of customizations, data, and overall system performance.
3. **Mitigates Risks:** Governance helps to mitigate risks associated with system changes by enforcing a process for review and approval.
4. **Supports Scalability:** With proper governance, Salesforce implementations can be scaled smoothly as the organization grows.
5. **Ensures Alignment with Business Goals:** Governance ensures that all Salesforce initiatives align with the organization's strategic goals and objectives.

Governance strategies to mitigate anti-patterns in Salesforce:

1. **Establish a Clear Decision-Making Process:** Implement a governance committee with representatives from key departments to make decisions about Salesforce usage and customizations.
2. **Set Standards and Guidelines:** Develop standards and guidelines for Salesforce usage, data entry, customizations, and updates to avoid anti-patterns like over-customization and poor data management.
3. **Regular Reviews and Audits:** Conduct regular reviews and audits to identify and correct anti-patterns.
4. **Enforce Change Management:** Implement a strong change management process to ensure all changes are reviewed, tested, and approved before being pushed to production.
5. **Training and Education:** Regularly train and educate users about best practices to avoid anti-patterns and to improve their efficiency and effectiveness with Salesforce.
6. **Regular Updates:** Keep Salesforce regularly updated to take advantage of new features and improvements, and to ensure security.
7. **User Experience Focus:** Prioritize user experience in all Salesforce initiatives to ensure high user adoption and satisfaction.
8. **Data Governance:** Implement strong data governance practices to ensure data quality and integrity.

By implementing a strong governance framework and following these strategies, you can effectively mitigate Salesforce anti-patterns, improve user adoption and satisfaction, and achieve your organization's goals with Salesforce.

Conclusion: The Impact of Anti-Patterns on Businesses

Recap of Anti-Patterns and their Business Impact in Salesforce:

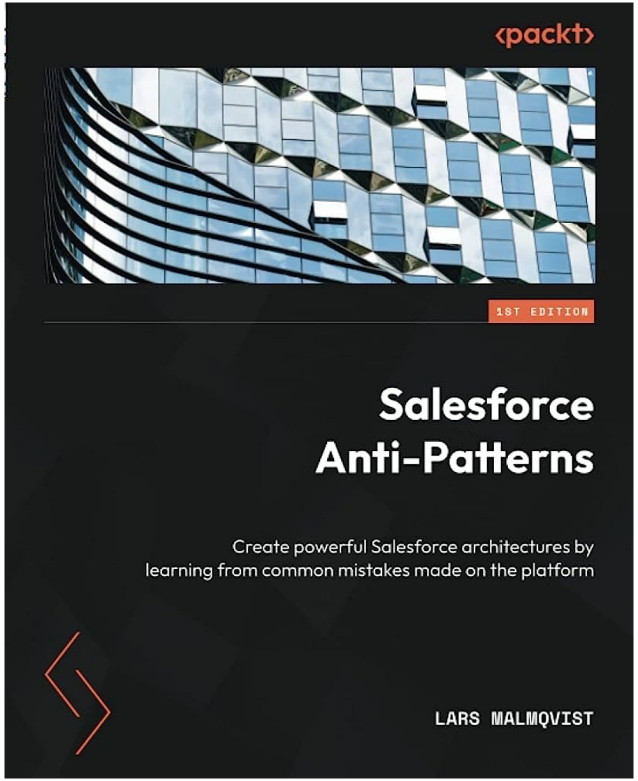
1. **Over-customization:** Excessive customizations can make the system complex, difficult to maintain, and can lead to performance issues. This can impact user adoption and productivity.
2. **Poor Data Management:** Bad data management practices can lead to data inconsistencies, duplicates, and inaccuracies, affecting business decisions and customer relations.
3. **Excessive Use of Code:** Over-reliance on code (over Salesforce's built-in declarative options) can make the system more difficult to maintain and update, potentially increasing costs and decreasing system stability.
4. **Improper Use of Roles and Profiles:** Incorrectly defined roles and profiles can lead to data access issues, impacting user productivity and potentially causing security issues.
5. **Ignoring Ongoing Salesforce Training:** Lack of ongoing training can lead to misuse of the system, lower user adoption, and inefficiency, hindering the return on investment from Salesforce.
6. **Neglecting User Experience:** A poor user experience can result in low user adoption, productivity losses, and higher training costs.
7. **Skipping Salesforce Updates:** Ignoring updates can lead to missed new features, security risks, potential software incompatibility, and limited support from Salesforce.

Role of Best Practices in Avoiding Anti-Patterns:

1. **Adhere to Salesforce's "Clicks Before Code" Philosophy:** Always look for declarative, out-of-the-box solutions before resorting to code.
2. **Implement a Strong Data Management Strategy:** Regular data audits, cleanup activities, and enforcing data entry standards can help maintain data quality.
3. **Use Roles and Profiles Correctly:** Properly design and assign roles and profiles to ensure the right data access and maintain security.
4. **Provide Continuous Training**: Regular training helps users adapt to changes, learn new features, and use the system more efficiently.
5. **Focus on User Experience**: Design the system with the end-user in mind to improve user adoption and satisfaction.
6. **Stay Up-to-date with Salesforce Updates**: Regularly update the system to take advantage of new features, improvements, and security patches.
7. **Establish Governance:** Implement a Salesforce governance framework to make informed decisions, maintain system quality, and align Salesforce initiatives with business objectives.

By adhering to these best practices, businesses can avoid most common Salesforce anti-patterns, ensuring a secure, efficient, and user-friendly Salesforce environment that effectively supports their business needs.

Continued Learning...



‹packt›

1ST EDITION

Salesforce Anti-Patterns

Create powerful Salesforce architectures by learning from common mistakes made on the platform

LARS MALMQVIST

Developers' Blog

Salesforce Anti-Patterns: A Cautionary Tale

By Steve Bobrowski