International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 10 | Oct - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Building Vanilla Layer Code in Customer Communication Management Tools

Renuka Kulkarni

Independent researcher, USA

Renukak12@gmail.com

Abstract

Customer Communication Management (CCM) software tools help businesses create, manage, and deliver personalized, multichannel communications to their customers. These communications can include various formats, such as invoices, statements, marketing materials, notifications, emails, and customer service correspondence. Building a Vanilla Code Base is an effective strategy in large organizations where different departments or regions need similar but slightly different versions of customer communications. The idea is to create a foundational set of templates, workflows, and components that can be used universally across different regions or departments. From this base, customizations can be layered on top for specific regional or departmental needs.

Keywords

Customer Communication Management, code base, reusability, scalability, Maintainability

Introduction

When we talk about big organizations that are spread across multiple regions, i.e., supporting the business across the globe or providing business to various domains, These organizations, when communicating to the outside world, ensure that key communication structures (templates, workflows, etc.) are consistent across all regions or departments. This reduces errors and inconsistencies. In such scenarios, A Vanilla Code Base is essential in modern software development, particularly in large organizations with multiple departments, regions, or teams that require consistent, scalable, and maintainable solutions. The idea behind a Vanilla Code Base is to create a core, standardized foundation that can be easily extended or customized to meet the specific needs of different parts of the organization without reinventing the wheel each time.

The main objective of the Vanilla code base is to ensure that the system adapts to specific local requirements and avoids code duplication. The Vanilla Code Base concept revolves around creating a foundational architecture that is both portable and flexible.

Breakdown of architecture

Below is a breakdown of key principles and steps to build such a codebase

Designing the systems

While designing the system, each requirement needs to be broken down into independent, reusable components. Each component should serve a specific function, such as user data processing, masking sensitive data, and business logic. This allows developers to focus on one part of the system at a time, making it easier to maintain. Another vital point when designing a common code base is separable utilizing that changes in one module shouldn't heavily impact others, facilitating easier updates and modifications in the future. Another important aspect is that code should be moldable, allowing for easy integration of new features without adding extra burden to the core system. Utilize methodologies such that custom features can be added or modified as needed.

Reusability

Reusable templates such as bills, letters, emails, and postcards should be generic enough to be applicable across different use cases but flexible enough to be customized as per the specific requirements of the region or department. A typical data structure needs to be maintained across the design to ensure no structural changes in integration between the systems. At the same time, data structures should allow adding new sections or data items that might be needed while customizing the vanilla code for customization. System design and processes should be configurable and flexible. A few processes can act as base processes but should be easily updated by adding or modifying logic and triggers.

Customizing the Vanilla code

It is critical that the base system remains unchanged and customizations are implemented on top of it. This can be achieved by allowing region-specific or department-specific configurations, such as localized workflows, data formats, or user interfaces while keeping the core code untouched. An important point that should be considered is that instead of going for hardcoded code, more reference files and configuration-based architecture should be adopted to customize the code effortlessly.

Documentation and version controlling

Building version control ensures that the correct code is checked out for regional development, reducing redundancy and confusion while starting the implementation of localization. Robust testing strategies for base and localized code and supporting documentation ensure that core and customizations work correctly.

Generally, core systems are tested with mock data, dummy system loads, and interfaces by stimulating real implementation scenarios. When implementing the vanilla code base for any regional customization, ensuring the system can handle increasing amounts of traffic or data without significant performance degradation is essential.

Vanilla Set up in CCM

For CCM, the specific features that need to be considered when building a vanilla code base are as follows. Firstly, composition templates need to be built so that they contain all the essential parts of communication, e.g., In the case of print output for a bank while coding, it should be built in such a way that the template provides areas for customer address, salutation, account summary, account transactions, cheques should be included and create in such a way that there should be minimal changes when localization it made on the base template. Data files should be ready to reuse as they allow the addition of specific local fields. Standard output facilities such as print and e-drivers should have required features added so that development on print channels is none or negligible. The critical step in building a vanilla code base involves analyzing components that can be reused and are flexible to make modifications.

When designing a template, Break down the template into modular sections or blocks, such as the customer address, salutation, account summary, account transactions, and cheque details. This way, each section can be modified independently without affecting others. This also allows easy localization, where only specific sections (like the salutation or account summary) must be changed for different languages or regions. Localization should ideally only require changes to text content (e.g., for other languages or country-specific legal requirements) rather than structural changes to the template. To ensure consistency, define a standard data file format (such as XML, CSV, or JSON). The data structure should be flexible enough to handle all required fields and extensible for future needs. For generating outputs like print and electronic documents (e.g., PDFs, statements, etc.), the code base should include Print Output Generation and Electronic Output (E-Drivers). As the system evolves, businesses might need to add new channels (e.g., SMS, email). In such cases, it is critical to ensure the codebase is modular so the same template can be rendered for different output channels with minimal development overhead.

Advantages of using Valina code base

Below are the key advantages of using the Vanilla layer code base in CCM.

Globalization and localization

For large organizations with a global presence, localization (adapting the content for a specific region) and internationalization (building flexibility into the system to efficiently support various languages, currencies, etc.) have become important. A Vanilla code base allows for defining core communication content that can be adapted to regional differences, like languages, currencies, and legal requirements.

Cost-Effectiveness

Reusing the same base code for different regions or departments minimizes the need to create new solutions from scratch. Customizations can be applied on top of the Vanilla Code Base, making the development process more efficient and cost-effective. When a bug is found, fixes only need to be applied once. Without a Vanilla Code Base, each version of the code (for each region/department) would require separate maintenance, increasing costs and complexity. As business requirements change or new features are added, updating a centralized code base is more straightforward than dealing with multiple customized versions of the same code. This leads to fewer errors and more efficient updates.

Scalability

When new regions or departments are added, the only item that needs to be analyzed is the localizations, as the Vanilla Code Base provides a foundation that can quickly be adapted. New teams can start working on top of an established structure, reducing the ramp-up time. A Vanilla Code Base is designed to be scalable, allowing organizations to add new regions, departments, or features without significantly reworking the code. This makes it easier for the company to grow or change without disrupting the existing structure.

Organizational collaboration

A standard source code for multiple programs and modules reduces complexity at all application development and operation levels. It makes life easier for all involved teams across the departments. As all departments collaborate and work together to achieve consistency, it reduces effort for development, promotes rapid code development, and results in simple deployment of updates and upgrades. This approach ultimately strengthens the organization's ability to deliver a seamless customer experience across multiple channels.

Consistent branding

As the same code is reused across various departments/regional developments, reusability is achieved, leading to centralized branding logic, making it flexible, improving efficiency, ensuring brand consistency, and, most importantly, creating a base for all future branding needs. Therefore, Vanilla code helps maintain brand consistency while reducing complexity and maintenance costs.

It creates and maintains a unified brand identity and message across all marketing channels by generating a consistent customer experience, which helps to gain customer trust and clarity is established with customers, enabling them to recognize a brand quickly.

Disadvantages of Vanilla code in CCM

Performance issues

Sometimes, even the Vanilla code cannot provide the expected performance. CCM systems are typically focused on creating and delivering customer communications (e.g., statements, invoices, marketing emails); some components may involve graphics rendering (e.g., dynamic templates, digital signatures, custom branding) or the use of specific hardware capabilities (e.g., integrating with a customer's mobile device camera or location services). Apps built from a common codebase may not always perform on all apps, particularly in graphics-intensive applications or those requiring considerable device hardware interaction.

Longer Learning curves

While a vanilla codebase provides a common foundation and a unified structure for Customer Communication Management (CCM) applications, it does introduce specific challenges, especially for regional teams who need to implement and maintain the codebase in a way that aligns with their local requirements. The complexity of understanding the core code and the coding methodologies used by the central team can increase the learning curve for regional teams.

Over Complex code

To achieve reusability, sometimes vanilla code becomes too complex. As the code is complex, in a few cases, making regional changes gets too complicated to fit in existing designed methodologies and conventions. This results in complex workflows, higher deployment costs, and increased errors due to the difficulty of fitting regional needs into the standard design.

Lack of innovation

Sometimes, regional team developers can design a more efficient workflow or logic than common code, but regional teams are bound to use the base code, which adds a barrier to use own logic. With the limitation of using designated code, the team lacks innovation. Also, the team usually gets occupied with understanding the existing code and loses interest in implementing new mythologies. As the standard Vanilla gets older, it sometimes becomes obsolete with the latest trends, making it even more costly for organizations as the cost of adapting to new technologies in existing frameworks is enormous.

Conclusion:

Using a vanilla codebase in Customer Communication Management (CCM) brings numerous benefits, including reusability, standardization, and cost efficiency across various regions and teams. Organizations can maintain consistent communication strategies, streamline workflows, and ensure alignment on coding conventions and methodologies by establishing a standard foundation. This reduces development time, makes it easier to maintain, and minimizes errors across multiple communication channels.

However, relying exclusively on a vanilla codebase presents challenges. Over time, as the codebase evolves to accommodate regional needs, it can become increasingly complex, making regional customizations, innovation, and adaptation to new technologies more difficult. Regional teams may find it challenging to implement more efficient or innovative workflows, which can result in bypassing the core system. Additionally, adapting it to new trends and technologies can become costly as the vanilla codebase ages.

Organizations should prioritize strategies such as modular design, flexible customization, and incremental updates to make the most of a vanilla codebase while addressing these challenges. Promoting cross-functional collaboration and regularly updating the codebase will help avoid stagnation and ensure the CCM system remains adaptable, scalable, and innovative.

In the end, a balanced approach preserving a strong core system while allowing for regional flexibility and continuous adaptation to new trends can help organizations maximize the potential of their CCM strategy, ensuring both efficiency and long-term success.

References:

[1]Nikhil. Bhargav.” Vanilla Software and Programming”.baeldung.https://www.baeldung.com/cs/vanilla-meaning.(accessed Jul. 5, 2024)

[2]Avinash. Shivankar. ”Implementing Vanilla ERP Systems: Factors to consider in strategy, business alignment & customization.” .batchmaster. https://www.batchmaster.co.in/wp-content/themes/engitech/download/implementing-vanilla-erp-systems-ebook.pdf.(accessed Jul. 6, 2024)

[3]"Vanilla Implementation of Oracle PDF"scribd . https://www.scribd.com/document/289649081/Vanilla-Implementation-of-Oracle-pdf.(accessed Jul.26, 2024)

[4]"Implementing ERP Systems". ques10. https://www.ques10.com/p/48195/implementing-erp-systems-1/.(accessed Jun. 16, 2024)

[5]"What does Vanilla mean in software development?". aerion. https://aerion.com.au/wiki/what-does-vanilla-mean-in-software-development/.(accessed Aug. 14, 2024)

[6]"thatsoftwaredude". Thatsoftwaredude.https://www.thatsoftwaredude.com/content/6343/the-benefits-of-coding-in-vanilla-javascript. (accessed Jul. 6, 2024)

[7]"Common Codebase vs. Separate Codebase". Insights.https://insights.daffodilsw.com/blog/common-codebase-vs.-separate-codebase.(accessed Aug. 10, 2024)

[8]"Advantages and Disadvantages of using Shared Code from the Developers Perspective: A Qualitative Study ". Researchgate.https://www.researchgate.net/publication/304375879_Advantages_and_Disadvantages_of_using_Shared_code_from_the_Developers_Perspective_A_qualitative_study/link/5772e31608aeeec38954163b/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19.(accessed Aug. 16, 2024)