ACCSCIENCE PUBLISHING

Design+

ARTICLE

Extending the Essence framework to adopt user story and microservice practices for automated real estate pre-sales customer relationship management

Parthasarathi Ray[1]* and Pinakpani Pal[2]

[1]Applied Statistics Unit, Indian Statistical Institute, Kolkata, West Bengal, India

[2]Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, West Bengal, India

Abstract

As a framework, Essence can operate completely on its own or in tandem with existing frameworks, helping organizations engaged in greenfield development. Although it is an established framework, there remains space for improvement. We leverage Essence's ability to serve as a common ground, allowing different practices to be composed in our methodology. An industry use case for automating the pre-sales business processes in real estate customer-relationship management is demonstrated by charting the journey of a real estate organization undertaking a development project. We demonstrate how to essentialize practices such as user story and microservices on top of Scrum, creating a method tailored to the development project that leverages Essence. The widespread use of the selected practices imbues the resulting method with versatility that could make it suitable for different industry/domain settings. Additionally, this exercise to compose practices into a method guided by Essence can be utilized in a much wider context, both from a practice-composition perspective and an industry/domain applicability perspective. This approach can be generalized as a blueprint for essentializing combinations of practices suited to a given context (for example, by selecting practices that address context-specific requirements) and composing them under the guidance of the Essence framework. This enables the development of a method that is sufficiently detailed and prescriptive to achieve the intended objectives across diverse industry and domain settings.

Keywords: Agile; Scrum; Essence; User story; Microservices; Customer relationship management

***Corresponding author:**
Parthasarathi Ray
(Parthasarathi.ray.86@gmail.com)

Citation: Ray P, Pal P. Extending the Essence framework to adopt user story and microservice practices for automated real estate pre-sales customer relationship management. *Design+*. 2026;3(2):025070012. doi: 10.36922/DP025070012

Received: February 10, 2025

Revised: March 2, 2026

Accepted: March 2, 2026

Published online: May 5, 2026

1. Introduction

With client-centricity at its core, focusing on business relationship improvement, customer retention, and customer experience augmentation, customer relationship management (CRM) continues to dominate the market as the largest and fastest-growing enterprise application software.[1–4]

Real estate, as an industry, exerts considerable influence on global economies,

attracting significant investor interest, and is deemed a strategic resource.[5–8] While numerous factors drive real estate businesses to adopt technology, business process automation has become a crucial driver for real estate companies to achieve excellence.[9,10]

The information technology (IT) organization structure of a real estate company has a strong bearing on its technology adoption. Typically, a real estate company tends to fall into one of the following three categories that are applicable to all non-IT companies in general:

(i) C-I: While there is a clear IT strategy present for the organization, it may not have a support IT structure, or even if it does, that IT structure is not equipped to implement the IT strategy on its own. Therefore, that organization must hire external consultants and/or IT vendors to implement that IT strategy. If a support IT structure exists for such an organization, it may not be capable of executing projects on its own and would likely find it difficult to embrace the latest technology stack/programming paradigms. However, its personnel may have the skills to engage external consultants/IT vendors to ensure successful project implementation and may be able to take over support and maintenance after deployment. Joint application development might also be a possible approach.

(ii) C-II: This organization is characterized by the presence of a mature in-house IT division capable of executing project(s) (which can involve greenfield development or package solution adoption) on its own. The IT division also has the expertise and maturity to embrace the latest technology stack/programming paradigms.

(iii) C-III: IT support is almost non-existent in this case, with the organization just being able to stay afloat with no significant investment in IT resources forthcoming.

The current study discusses the exercise conducted for a real-estate organization that falls into the first category above to automate the pre-sales function of its CRM. It has an in-house IT team that needs prescriptive guidance to translate the IT vision into reality, necessitating the involvement of external IT consultants to provide the necessary expertise and facilitate the adoption of methodologies. We refer to that organization as Nirmanik in this paper.

The CRM systems compile data from various communication channels, including a company's website, phone/mail/chat, campaigns (digital/non-digital), and social media.[11] A number of software are commercially available in the CRM space today, addressing a host of functionalities and business scenarios. While the pros and cons of the software need to be carefully considered in the given context, there is also the option of developing bespoke solutions. To make the right decisions to achieve the desired benefits, we should leverage software engineering and apply its proven practices and design methodologies to consistently manage the ever-increasing complexity.

Agile transformation seems ubiquitous, and most of those are Scrum. However, numerous Scrum implementations face challenges, where Essence, an object management group (OMG) industry standard[12], provides value by improving the probability of success.[13] A key distinction of Essence is that it is a perfect partner to any method or framework an organization uses, allowing us to describe our evolving practices in a method-agnostic yet structured way. As a framework, it also allows us to create new practices if needed and shows us how those will work alongside what we already have. This is a crucial reason we decided to leverage Essence, as it helped us make the Scrum implementation strategy more effective by facilitating the accurate identification of new practices to supplement the Scrum framework. Additionally, the selection of Essence as the underlying framework is bolstered by its acclaim from industry and academia alike as eminently suitable for integrating software engineering practices. We have decided to leverage Essence's established advantage rather than resort to alternative frameworks, which may not have such a unifying capability.

As we made Essence central to Nirmanik's endeavor as a framework while leveraging Scrum, it became clear that we would need to induct additional practices to address the four consecutive kernel solution activity spaces: understand the requirements, shape the system, implement the system, and test the system, the keys to the software development lifecycle spanning requirement analysis, design, development and testing. This paper aims to articulate our accomplishment in leveraging Essence to help Nirmanik address the said kernel solution activity spaces by adopting the user story and microservice practices, and to detail those out, along with the associated alphas and work products, as part of their essentialization.

While the user story and the microservice practices are both well-established in their own capacities, the improvement here would be to weave them into a Scrum-based method after essentializing them, as guided by the Essence framework. The resulting method can also provide guidance for similar project implementation endeavors in a variety of industry/domain settings where these practices are deemed to be good choices, assuming the organization in question either falls in the aforementioned category C-I, requiring help from external IT consultants/IT service providers, or belongs to category C-II by virtue of having a robust internal IT organization. Organizations belonging

to category C-III would not qualify for the reasons already stated.

There is an even wider context to which our work might be extended, where the exercise carried out here can serve as a blueprint for a variety of industry/domain settings in which the practices selected (or deemed suitable) differ from those used here. Leveraging the principles of our work in this paper and keeping Essence as the central framework, those practices can be essentialized and composed into a method to realize the objectives for the industry/domain setting under consideration.

The remainder of this article is organized as follows:

- Key concepts of Essence
- Revisit functionalities in scope, business process modeling, and Essence adoption
- Considerations for composing practices into a method using Essence
- Adoption of the user story practice
- Adoption of microservice practice
- Conclusion and future direction

2. Key concepts of Essence

Essence defines a set of core alphas that are universally applicable across different practices, enabling teams to understand their current position within the overall context of the endeavor. The Essence specification identifies seven alphas commonly applicable for all software engineering endeavors: opportunity, stakeholders, requirements, software system, work, team, and way of working.[14]

We now consider these elements more formally by examining the two constituent components of Essence: the Essence Kernel and language.

The Essence Kernel defines the most basic elements needed by all kinds of development. It also facilitates developers' understanding of the project's state and the next activities to be completed.[15]

Essence starts by organizing the elements of software engineering (e.g., the seven common alphas described earlier) into three areas of concern, each covering a distinct dimension of software development[15]:

Customer: covers everything related to the usage of the software system intended to be delivered. The alphas "opportunity" and "stakeholders" are mapped here.

Solution: contains everything about the specification and construction of the software system. The alphas' "requirements" and "software system" are mapped here.

Endeavor: about the team undertaking development work and their approach to carry it out. The alphas "work," "team," and "way of working" are mapped here.

The three areas of concern, customer, solution, and endeavor, are depicted in the Essence cards using the colors green, yellow, and blue, respectively. However, in our study, we are refraining from using color due to limitations; hence, all cards and related diagrams do not include colors.

Table 1 summarizes the concepts for the elements used in Essence language along with their corresponding icon symbols.[16]

Of the artefacts mentioned, we selected two key artefacts, an alpha card and an activity card, and showed the corresponding cards for illustration purposes. Figure 1A and 1B show the Essence alpha card "user story" and the Essence activity card "prepare a user story," respectively, pointing out their linkage with the constituent Essence elements.

2.1. Alpha state cards for requirements

Figure 2A–2C show the Essence alpha state cards for three requirements—conceived, bounded, and coherent.

2.2. Applicable Essence cards for user story lite

Figure 1 shows the alpha card "user story" and the activity card "prepare a user story." The work product story card, the activity cards—"find user story" and "accept user story"—and the pattern card "splitting user story" are shown in Section 5.2.

2.3. Applicable Essence cards for microservices lite

The "microservice" alpha card is shown in Section 6.1. The "design model" work product card and the "microservice design" work product card are shown in Section 6.2.[16]

3. Revisit functionalities in scope, business process modeling, and Essence adoption

The industry being considered is the real estate business, where technology-driven automation has been making a difference. As part of the automation scope for the pre-sales CRM function of the real-estate organization, Nirmanik, we focus on the following business processes:

- Property management
- Lead generation for properties
- Automated workflows

In our previous study, we discussed how a large conglomerate, as part of its diversification strategy, had ventured into real estate to take advantage of favorable market conditions and to leverage synergies with its other affiliated operations, and had subsequently spun off the Nirmanik organization.[17] The conglomerate had an in-house IT team with a track record of supporting many of the affiliated organizations' IT initiatives and, hence,

Table 1. Elements of Essence language

Element type	Symbol	Description
Alpha		An essential element of the software engineering endeavor that helps assess the progress and health of the endeavor
Alpha state		The alpha states denote the lifecycle of an alpha
Activity		Things that practitioners carry out
Activity space		A placeholder for things to be carried out in the software engineering endeavor, with zero to many activities
Work product		The tangible artifacts produced by practitioners while conducting software engineering activities
Level of details		The extent of detail in a work product. An example would be a story card having its value expressed or additionally having its acceptance criteria listed
Competency level		The set of abilities, capabilities, attainments, knowledge, and skills needed to accomplish a specific kind of work. A number is provided to indicate the necessary expertise level
Pattern		An arrangement of other elements represented in the language
Resource		A source of information or content, such as a website or a book reference
Essentialized method		A team's way of constructing its work in a System of Interest development endeavor
Essentialized practice		A repeatable approach to carry out certain work, with the main areas of concern that a software engineering endeavor has to focus upon
Essence Kernel		A set of elements forming a common ground to describe a systems engineering endeavor
Essence language		The Essence language is a domain-specific language to define methods, practices, and kernels
Essence		The combination of the Essence language and kernel that provides a common foundation for presenting guidelines across all software engineering practices

a solid understanding of the prevailing software stacks. It was decided to leverage this team to custom-build a solution to address the requirements. While the technical expertise of the in-house IT team was acknowledged, there was nonetheless a need to explore whether some practices/frameworks could provide additional agility, which was where the external IT consultants came in to provide the required perspective and facilitate the adoption of the methodology.

Nirmanik had put up a company website with its property listings to solicit inquiries from prospective customers, in addition to traditional advertising channels like billboards and hoardings. The inquiries from these channels, online real estate portals, social media, and call inquiries were examined after being de-duplicated (removing potential repetitions), and the list of these potential customers, i.e., the leads (resorting to the customarily used term), was prepared.

The status of a lead who was involved in the system for the first time (fresh lead) was one of the following:

(i) Guest lead: Any fresh lead who wants to visit but has yet to confirm a visit date. It was expected that the telemarketer would call the lead multiple times to schedule the visit and convert the lead into a prospective lead.
(ii) Prospective lead: Any fresh lead who shows a clear intent to visit right away by providing a visit date.

In response to the request by the prospective lead, an assignment process was carried out to match the request with an available sales executive and agree on a mutually acceptable time slot for the requested site visit date.

The sales executives then met prospective customers at the appointed time to conduct on-site visits. Subsequent interactions led to either a showing of intent by the prospect, eventually culminating in a property sale (booking), or a lost opportunity. The sale of the property

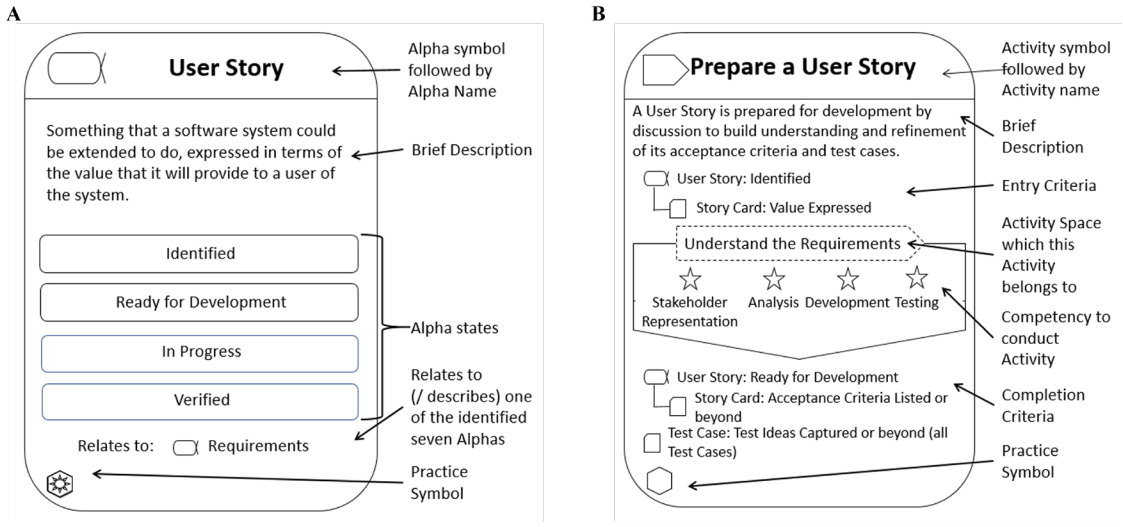

Figure 1. Examples of an alpha card and an activity card, showing the linkage with Essence elements. (A) User story alpha card. (B) Prepare a user story activity card. Both show the linkage with Essence elements.

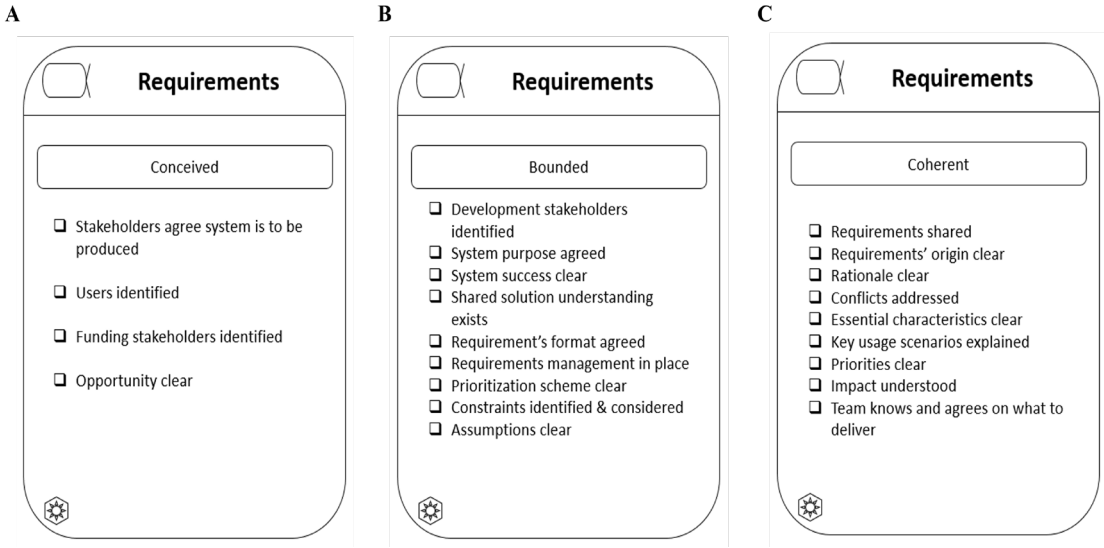

Figure 2. Requirements alpha state card.[16] (A) Conceived. (B) Bounded. (C) Coherent.

would be handled through the sales/post-sales process, which was beyond our scope.[17]

The workflow of real-estate pre-sales CRM functionalities in scope was captured in our previous study[17] using the communication flow diagram[18], as shown in Figure 3.

While the key practices for Nirmanik's development process were yet to be selected, it was decided at the outset to leverage Essence as the framework. The states of the Essence Kernel alphas were mapped to the development endeavor undertaken by Nirmanik, based on certain assumptions about the Essence Kernel alpha states for the project, as shown in Table 2.[17]

The convention of adding *Nirmanik::* prefix to the generic artifact name was adopted, with the resulting name put in italics, to distinguish the Nirmanik artifacts from the generic Essence artifacts. The remainder of this paper follows that convention.

A progression of the Nirmanik development endeavor was reflected in the advancement of the above-mentioned states of the Essence Kernel alphas. It was imperative to determine the best practices to adopt, culminating in the

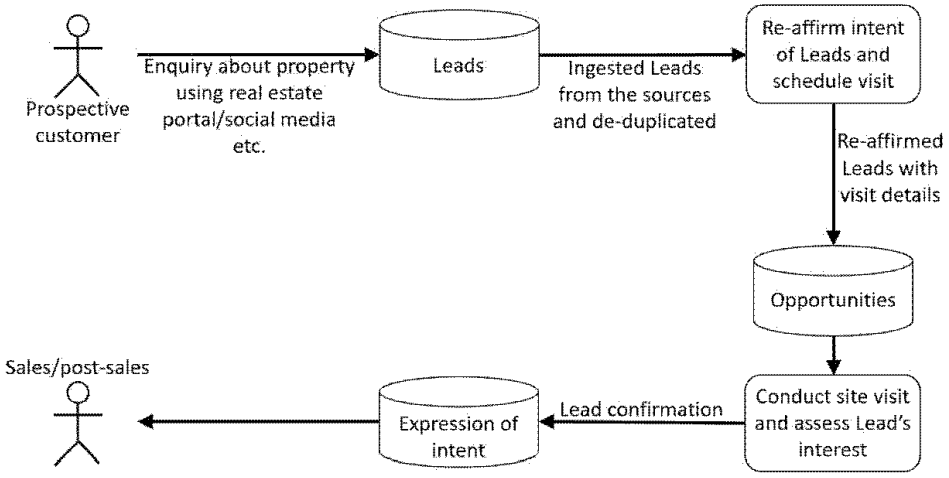

Figure 3. Communication flow of real estate pre-sales customer relationship management.

Table 2. The states of the Essence Kernel alphas mapped with regard to the development endeavor undertaken by Nirmanik[17]

Areas of concern	Alpha	Alpha state
Nirmanik::Customer	*Nirmanik::Opportunity*	Value established
	Nirmanik::Stakeholders	Involved
Nirmanik::Solution	*Nirmanik::Requirements*	Value conceived
	Nirmanik::Software System	Architecture selected
Nirmanik::Endeavor	*Nirmanik::Work*	Initiated
	Nirmanik::Team	Performing
	Nirmanik::Way of working	Foundation established

selection of key practices for the development process, using Essence as the framework and leveraging Scrum. The next section outlines considerations for composing practices to arrive at a method using Essence.

4. Considerations for composing practices into a method using Essence

A method supports software development teams by providing structured guidance and access to the practices needed during development. Methods are compositions of practices, and as the number of practices increases, the number of possible combinations can grow significantly. It thus follows that, while methods and method variants abound in the world, reusable practices are far fewer in number.

A key value of Essence is its ability to serve as common ground and to provide guidelines for all practices, thereby clarifying understanding and enabling appropriate

modification of practices. Essence provides a language and a kernel of software engineering and facilitates the comparison of practices described using the same common ground.[16]

While this allows the practices contributed by the software engineering community to be composed in different ways as necessary to form methods, those practices need to be essentialized first, i.e., they need to be described using Essence (the kernel and the language). As a result, the methods composed by using those practices are essentialized. Essentialization ensures that the descriptions of the method/practice are detailed to the basics. The value addition is realized through libraries of practices created from numerous methods. Therefore, we can mix and match practices selected from a shared practice library, such as one illustrated in Figure 4A, to obtain a method that suits a project context for organizations like Nirmanik, which belong to category C-I, as well as those belonging to category C-II (described in Section 1). While the starting point involved the Essence Kernel as the base, we selected a number of practices that would constitute Nirmanik's way of working. The set of practices thus selected, along with the kernel, would become the method for Nirmanik, as shown in Figure 4B.

If a team's starting point is a full-scale prescriptive "framework" and then it keeps making decisions about which elements to leave out, there's a likelihood of adopting more than is needed. An inexperienced team may become saddled with unnecessary techniques at the outset or burdened with unnecessary process overhead. On the contrary, starting with the absolutely essential ones and then adding each practice as required for a specific area is a better strategy, which aligns with what Essence recommends. This is very much applicable to organizations in category C-I (described in Section 1), where the external IT consultants or vendor IT organizations entrusted with the project implementation should be well-versed in this methodology and can facilitate the composition of the method based on the practices chosen after essentializing them. This can also be extended to organizations in category C-II, as they undertake project implementations on their own; even if they have good experience/knowledge of the technical stack, their understanding of software engineering can still benefit from the stated guidance. The applicability to organizations belonging to category C-III is ruled out for reasons already stated before.

We decided to opt for iterative development in an agile way as the agreed approach for Nirmanik, building the software in increments, leveraging Scrum. However, Scrum primarily addresses the activity spaces in the endeavor area of concern. We discerned gaps in the requirements area, which was part of the solution area of concern, as we considered the next steps. Acknowledging that the team would benefit from explicit guidance on activities in the solution area of concern to achieve greater effectiveness, we adopted the user story practice as detailed in the following section.

5. Adoption of the user story practice

In the previous section, where we mapped the states of the Essence Kernel alphas for the project, the "requirements" alpha[16] was deemed to be at the conceived state, as shown in Figure 2A. To address the requirements and advance the "requirements" alpha through the subsequent states, we used the user story lite practice, a simplified version of the user story practice. User stories enable a team to contemplate, question, and understand the value of their endeavors from a user-centric perspective.

5.1. User story lite practice: An overview

A user story is a depiction of the system functionalities of interest, articulated through an informal discussion between the system user and the developer.[19] User stories might be considered as sub-alphas, like treating "requirement" items as sub-alphas of "requirements."[16] The user story lite practice enabled us to break down "requirements" into user story sub-alphas. The corresponding alpha card is shown in Figure 1A.

The work products in the user story lite practice are the story card and the test case for each user story. A story card captures the narrative of a user story (as shown in Section 5.2), and a test case helps verify it.[16]

Figure 5 represents the essentialized model of the user story lite practice, showing relationships among the elements (alpha, work products, and activities) in the practice, the activity flow, and relationships with kernel elements. These are further elaborated in the subsequent sections.

5.2. Adoption of user story lite practice for Nirmanik

Working with user story lite entailed several activities, such as identifying user stories, preparing each for development, and accepting the implementation of each. The user story (*Nirmanik::Basic user story*) was created:

The top management of the Nirmanik real estate organization wanted to have an automated pre-sales management system in place so that leads obtained from various sources can be acted upon and nurtured to closure quickly using well-defined workflows with the provision of appropriate and timely alerts before handing over to sales/post-sales functions.

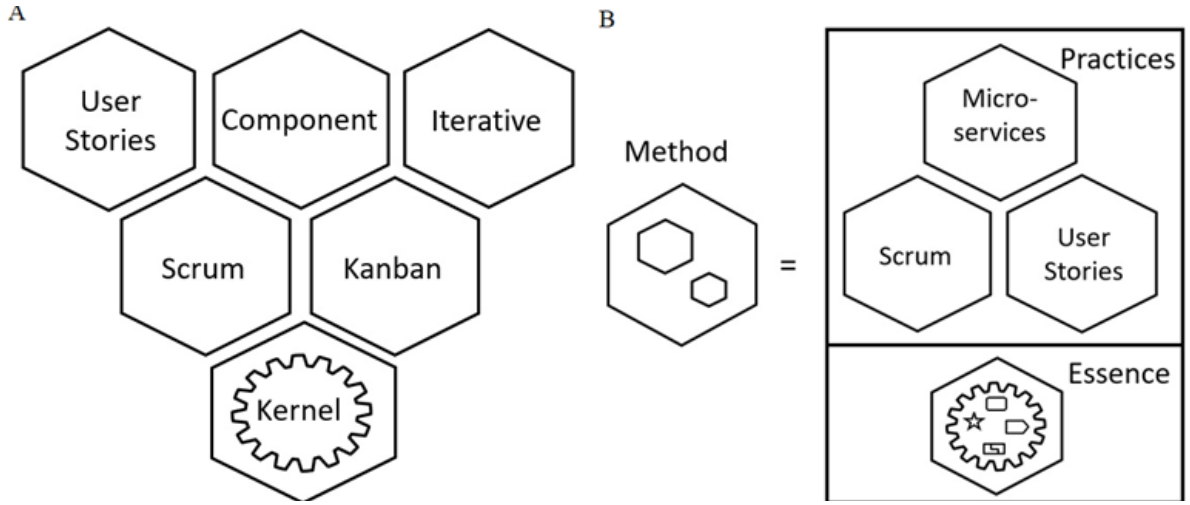

Figure 4. Leveraging practices from a shared library for method composition. (A) An example of a shared practice library. (B) Method: composition of practices on top of the Essence Kernel and language (for Nirmanik).

Once we decided to adopt user story lite for Nirmanik, we conducted the following activities: (i) find user stories, (ii) prepare each user story for development, (iii) apply the splitting user stories pattern, and (iv) accept the implementation of the user story.[16] The elaboration on how these activities were undertaken is as follows:

(i) *Nirmanik::Find user stories*: The "find user story" activity card is shown in Figure 6B.[16] The user stories to be taken up for development were identified, which would be as follows:
- *Nirmanik::Track pre-sales leads*
- *Nirmanik::Assign prospective leads to sales executives*
- *Nirmanik::Facilitate sales executive's activities to nurture the opportunity towards closure*
- *Nirmanik::Build reports/dashboards to facilitate timely action by management*

(ii) *Nirmanik::Prepare a user story*: The "prepare user story" activity card is shown in Figure 1B.[16] While proceeding to prepare the user stories for development, our primary focus was on the first story in Section 5.2, *Nirmanik::Track pre-sales leads*, as it was a key part of the CRM pre-sales activities for Nirmanik and provided the basis for the remaining three user stories as mentioned above. The user story developed for *Nirmanik::Track pre-sales leads* is as follows:
- User story (*Nirmanik::Track pre-sales leads*): The Nirmanik stakeholders would like to identify the pre-sales leads coming from various sources and track them as they progress. There should be a facility to capture leads from various sources. The sources can be online portals, social media, company website, calls, etc. The leads should be de-duplicated. While the same lead may appear multiple times across different sources, only one instance of the same lead should be tracked over the effective lifecycle of the lead. There should be a provision to indicate the progress or maturity of leads. Applicable progress status values include guest, prospective, inactive, and canceled. When a lead indicates a preferred site visit date and time slot, the lead status should be updated to prospective.

  The acceptance criteria are as follows: (i) There should be provisions to ingest leads from multiple sources. (ii) There should not be duplicate instances of leads over a lead's effective lifecycle. (iii) The status field for progress of leads should change appropriately as leads are tracked and followed up for visit confirmation.

(iii) *Nirmanik::Applying the splitting user stories pattern*: This involved splitting the larger user stories into smaller stories in alignment with the INVEST criteria—independent, negotiable, valuable, estimable, small, and testable—especially the small and testable criteria.[16] Figure 7 shows how the first user story, *Nirmanik::Track pre-sales leads*, was split into three smaller ones. The corresponding pattern card, "splitting user story," is shown in Figure 8A.

(iv) *Nirmanik::Accept a story*: This is about clearly depicting the acceptance criteria for the user stories. The frequent communication between the developers and the product owner over delivering the user story ensured better alignment and smoother acceptance. The "accept user story" activity is shown in Figure 8B.[16]

With the adoption of the user story lite practice, the requirements for the proposed software solution to be built were clarified, a shared solution emerged within the team, a requirements format was agreed upon, and the

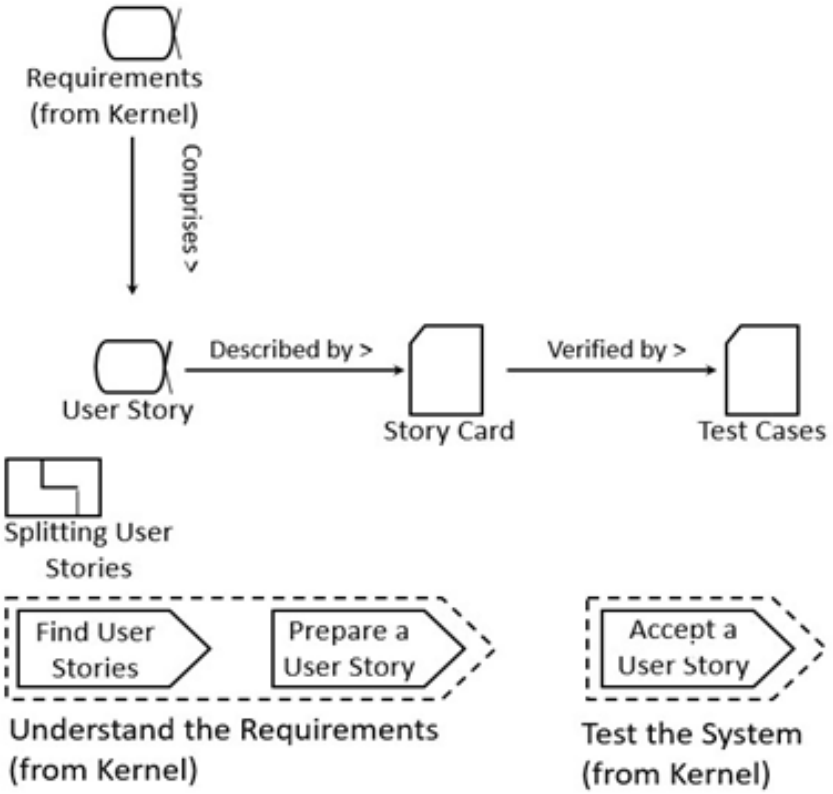

Figure 5. The essentialized model showing the user story lite practice.[16]

internal algorithms were well-articulated. Accordingly, the "requirements" alpha would advance from the conceived state, as shown in Figure 2A, to the bounded and coherent states, as shown in Figure 2B and 2C, respectively.[16]

Additionally, there is now clarity regarding the stakeholders' funding for this work. It was decided to utilize a backlog stemming from the requirement set drawn up for planning purposes, and to keep a regular communication channel open with the stakeholders to ensure relevance of the backlog, thus progressing the "work" alpha to the prepared state.[16]

5.3. The value of the kernel to the user story lite practice

A key accomplishment of essentializing the user story lite practice was to provide the Nirmanik team with clarity regarding which alphas were being progressed. We helped the Nirmanik team understand that the user story lite practice was instrumental in achieving the following Essence Kernel alpha states:

- Requirements alpha: Bounded and coherent state
- Work alpha: Prepared state

The three activities in user story lite[16] cover two kernel solution activity spaces: "understand the requirements" and "test the system," as shown in Figure 9.

However, even with the adoption of the practices thus far, two kernel solution activity spaces, "shape the system" and "implement the system," remained to be addressed, as indicated by their placement between the two aforementioned shaded spaces in Figure 9. The next section describes the adoption of the microservice practice to address design and implementation, covering the two aforementioned kernel solution activity spaces for Nirmanik.

6. Adoption of microservice practice

Microservices architecture involves developing an application as a collection of services that demonstrate properties such as cohesiveness and loose coupling, with each such service having complete ownership of its individual data, logic, and behavior.[20] Related functionalities are combined into a specific business capability (termed a bounded context), with each microservice ideally implementing one such capability.[21] It is important to identify the right partitioning of the system into microservices, given the impact of the architecture on

system performance.[22,23]

In the microservice practice, the microservices can be viewed as sub-alphas of the software system kernel alpha.[16] We decided to use the microservices lite practice, a lighter version of the microservice practice, for this endeavor. Individual components within the *Nirmanik::Lead tracking subsystem* were built as separate microservices rather than as a monolith within Nirmanik. That approach ensured no tight coupling to external systems, thus enabling the exploration and evolution of the new functionality in a de-risked manner.

6.1. Microservices lite practice: An overview

Microservices lite begins by identifying requirements using a suitable practice, such as user stories. Then, the microservices to implement the requirements would need to be identified.

The primary alpha in the microservices lite practice is the microservice alpha. All work products and activities in this practice are related to this alpha. The microservice alpha can be treated as a sub-alpha of the software system alpha.[16] The corresponding alpha card is shown in Figure 10. As a software system consists of microservice sub-alphas, each microservice has a bearing on the progress of the system.[16]

The work products of microservices lite are as follows:

- Design model: Inspection of the software system is required to break it up into microservices, leveraging criteria such as low coupling and high cohesion. This work product focuses on the interfaces between and the collaboration among microservices.
- Microservice design: Work product describing a microservice design from interfaces to behavior.
- Microservice build and deployment script: A work product, an automated script enabling each microservice to be deployed quickly, independent of others.
- Microservice test case: A work product for measuring the behavior of a microservice.

Figure 11 represents the essentialized model showing the microservices lite practice[16], the relationships among the elements (alpha, work products, and activities) in the practice, the activity flow, and the relationships with kernel elements.

6.2. Adoption of microservices lite practice for Nirmanik

Among the activities involved, a key activity is to identify microservices, the alphas in the microservices lite practice.

For Nirmanik, ingestion/consumption of leads from external sources (company website, external property listing sites, social media, etc.) is independent of identifying duplicates among those leads, while lead tracking and follow-up is a separate workflow. These three demonstrate the essential attributes of low coupling and high cohesion, thus making a good case for potential microservices. Accordingly, for *Nirmanik::Lead tracking subsystem*, the microservices include one for ingesting leads from external sources, one for de-duplication, and one for lead tracking and follow-up for visit confirmation.

As the initial efforts focused on getting the design in place for the microservice-oriented architecture, the design model and the microservice design work products were the most relevant artifacts resulting from the said exercise. Accordingly, we considered the design model and the microservice design work products and elaborated on them in the context of Nirmanik, specifically using the *Nirmanik::Lead tracking subsystem* as an example. Figure 12A shows the design model work product card.[16] This describes the software system alpha by listing its elements and showing their interactions.[16]

Figure 13 shows the design model created for the *Nirmanik::Lead tracking subsystem*. The work product in question has progressed to the point where collaborations and interfaces are defined, describing how the *Nirmanik::Lead tracking subsystem* microservices collaborate among themselves and interact with external lead-generation sources (such as real estate portals).

The left side of Figure 13 shows the external sources of leads. Most of them provide application programming interfaces (APIs). For those that do not provide APIs, suitable techniques need to be employed to build interfaces that provide the details of the leads as appropriate. These, collectively called iLeadEventProducer, are instrumental in emitting lead events. When an incident of note occurs (e.g., an interested customer searching for a property listing), it is transmitted via an event to the iLeadEventHandler interface. The *Nirmanik::Lead tracking subsystem* then processes that information and leverages it to trigger the subsequent workflows.

The right side of Figure 13 shows that *Nirmanik::Lead tracking subsystem* is designed using three microservices: *Nirmanik::Ingesting leads from external sources*, *Nirmanik::De-duplication of leads*, and *Nirmanik::Lead tracking and follow-up for visit confirmation*.

Each lead would be collected through the iLeadEventHandler interface. This lead ingestion process is well-suited to being implemented as a microservice, represented as the *Nirmanik::Ingesting leads from external*

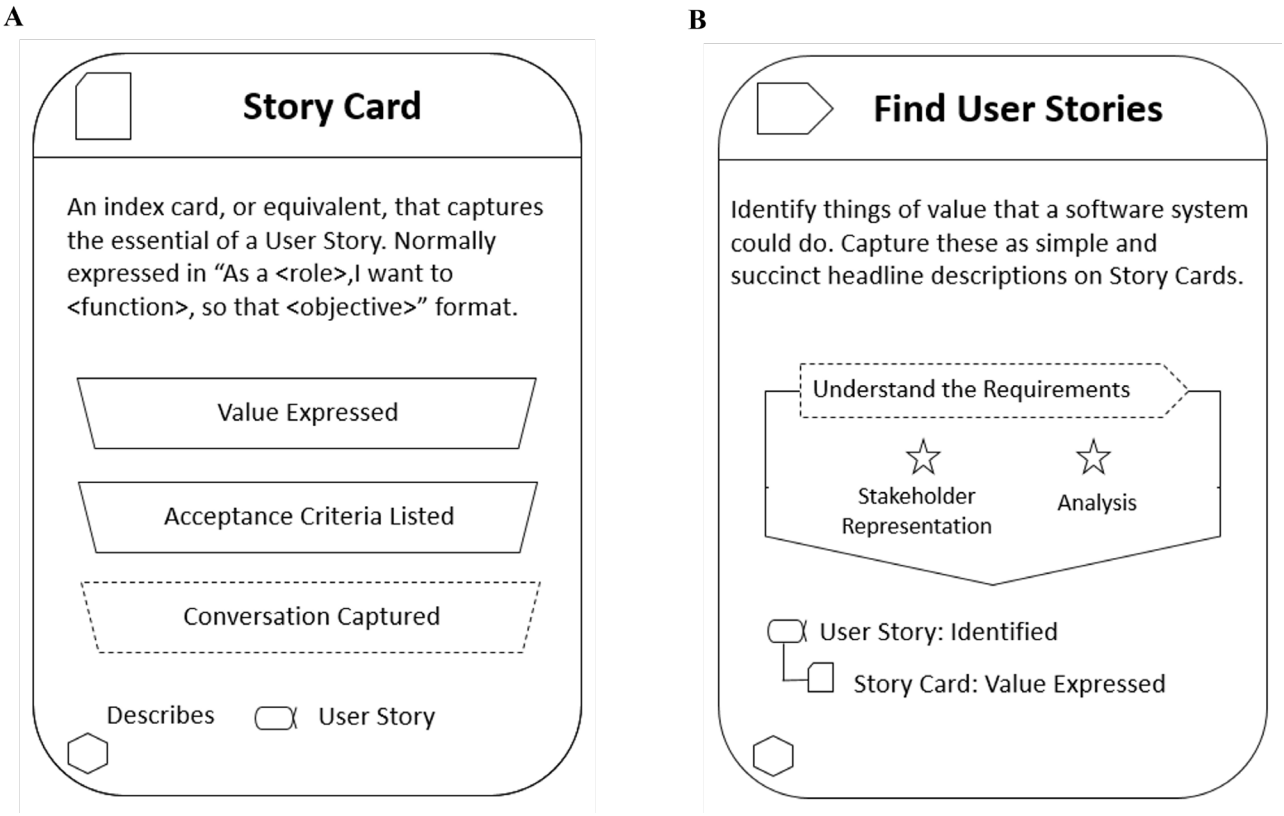

Figure 6. User Story cards. (A) Story card work product. (B) "Find user story" activity card.

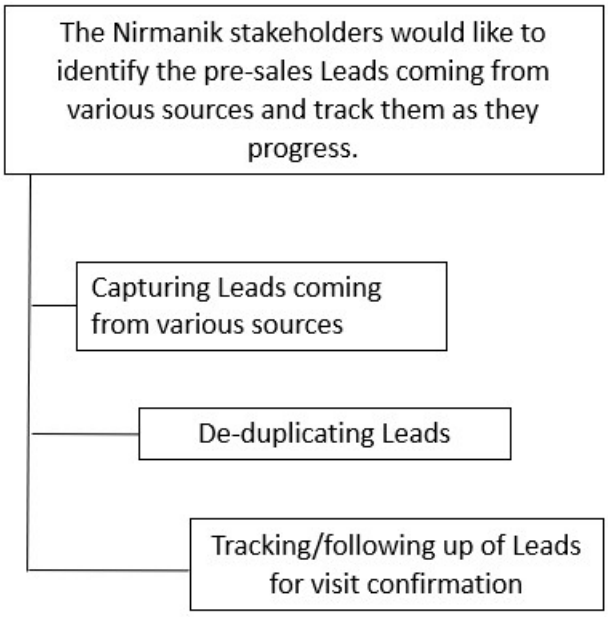

Figure 7. Splitting the first user story: *Nirmanik: :Track pre-sales leads.*

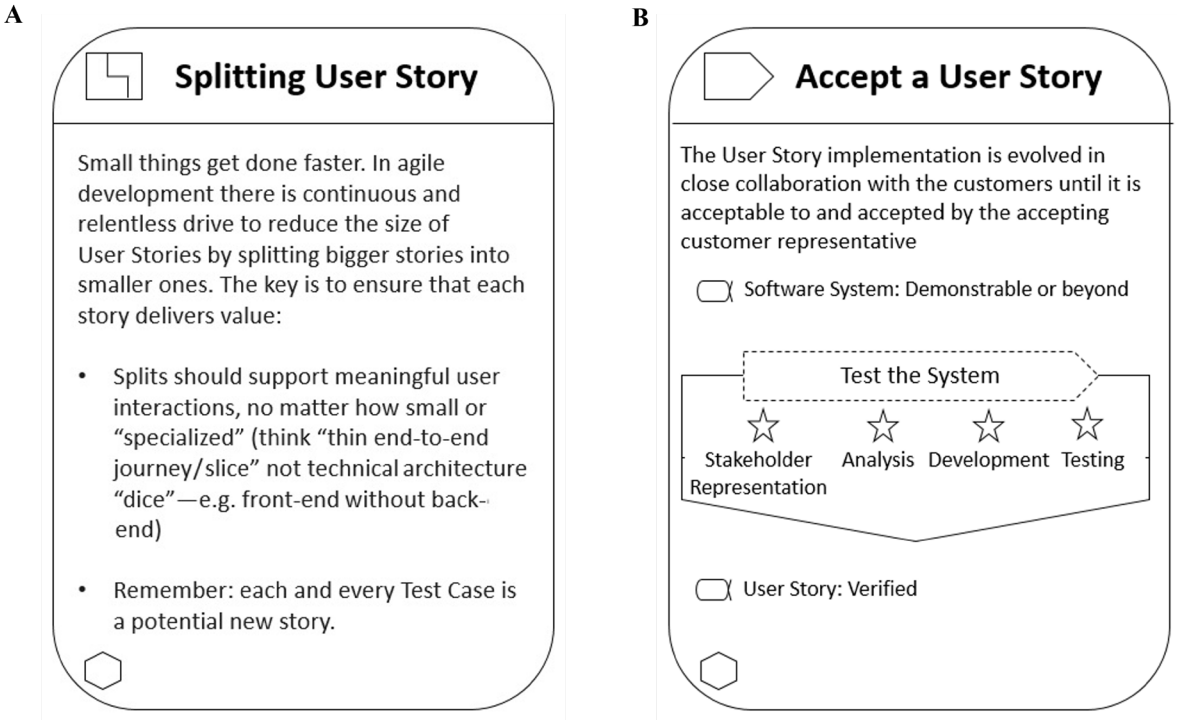

Figure 8. Pattern and activity cards for User Story. (A) The "splitting user story" card.[16] (B) "Accept user story" activity card.

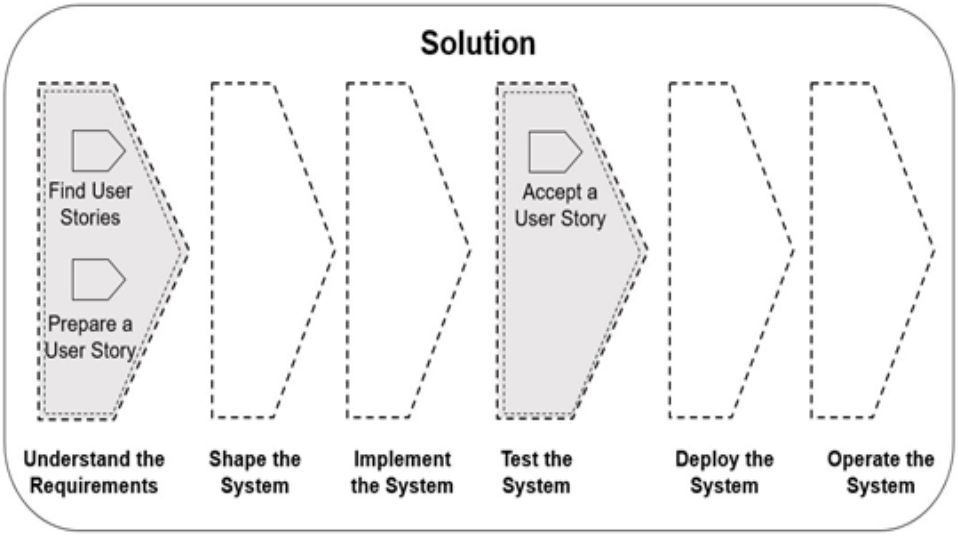

Figure 9. User story lite coverage of kernel solution activity spaces

sources microservice in Figure 13. This ingestion is accomplished via API, which can be either push- or pull-based. Pull-based API implies a batching mechanism. On the other hand, for leads that need to be treated in near real time, the push-based API is preferred.

De-duplication of the leads ingested by the system is an important activity. An individual interested in a given property might approach Nirmanik through multiple channels (multiple third-party property listing sites, social media, or Nirmanik's portal). This might result in duplicate entries for essentially one lead, leading to potential redundancies and potential credit allocation issues later. De-duplication of leads is critical for pre-sales to increase the effectiveness and efficiency of subsequent operations.

Without it, valuable resources might be wasted pursuing the same leads, the same prospect might receive multiple calls, resulting in irritation and possibly lost sales, and reporting would become inaccurate.

A combination of key attributes of the prospect (e.g., mobile number and email) can be used to identify duplicate leads. Ways to handle this might be to reject all duplicate leads outright or to treat the earliest-arriving lead as the parent lead and consider subsequent duplicate leads as its children. The second approach can help build intelligence on channel effectiveness in generating leads for Nirmanik.

The characteristics of this functionality also fulfill the key criteria for adopting a microservices approach. This is shown as the *Nirmanik::De-duplication of leads* microservice in Figure 13.

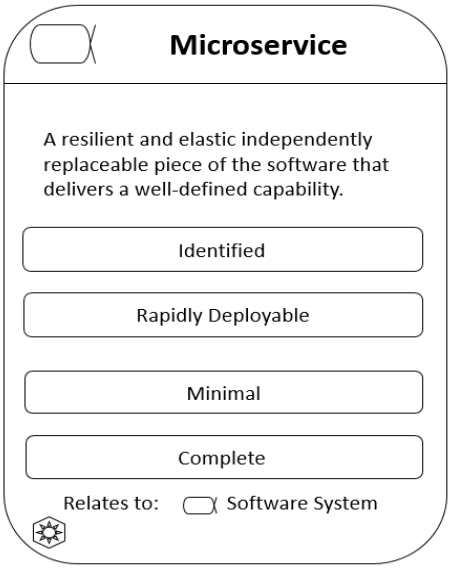

Figure 10. Microservice alpha card

Subsequently, the unique ingested leads are processed through a workflow as they are called up, with changes to the status field reflecting progress on the leads based on the response (or lack thereof), as detailed below.

The status field for lead progress can assume, at this stage, one of the following values as described below.

- Guest lead: Any fresh lead who wants to visit but has yet to confirm a visit date (time slot). It is expected that tele caller would call the lead a certain number of times to confirm the visit date (time slot) and convert the lead to a prospective lead.
- Prospective lead: Any fresh lead (or a guest lead) who shows a clear intent to visit by providing a visit date (time slot).
- Inactive lead: When the lead does not respond to the tele caller. It is expected that the tele caller would call the inactive leads a certain number of times to confirm the visit date (timeslot) to convert them to prospective leads.
- Canceled lead: An inactive lead that remains unresponsive over three attempts by the tele caller would have the status changed to canceled. A canceled lead would no longer be included in the system. If a canceled lead approaches later, the lead would be considered a fresh lead, and no prior history would be taken into account. A lead that responds to the tele caller indicates lack of interest in visiting in the foreseeable future is also categorized as canceled.

For prospective leads, a site visit date (timeslot) needs to be captured. It is later addressed in the next user story. This entire flow of activities has a good degree of cohesion; hence, it is a good candidate for a single microservice, shown as the *Nirmanik::Lead tracking and follow-up for visit confirmation* microservice in Figure 13.

Each of the three microservices exhibits high internal cohesion but low coupling among themselves, satisfying the essential characteristics of microservices. Each of these provides an iLeads interface to leads, with progressively more refinement (or processing) for subsequent stages.

Figure 12B shows the microservice design work product card.[16] This is a work product describing a microservice's design, including the interfaces, behavior, and internal design.[16] A key aspect of a microservice is its ability to communicate with other microservices, which is realized by its interface or API. It needs to be ensured that the APIs (i.e., the component interfaces) are loosely coupled, which would be instrumental for independent deployment of microservices, again a key requirement in this paradigm. While a set of interfaces in the design model is described in the previous sections, there might also be interfaces that a microservice would require to handle execution, as explained below.

Figure 14 shows all the interfaces to the *Nirmanik::Ingesting leads from external sources* microservice, which include: (i) iService (takes care of the execution), (ii) iLeads (handles the processed leads), and (iii) iLeadEventHandler (manages external lead-generating events).

Figure 14 is a work product for the microservices lite practice, drawn in the given context using the *Nirmanik::Ingesting leads from external sources* microservice as an example from the *Nirmanik::Lead tracking subsystem*, which has progressed to the level of specified interfaces.

As mentioned earlier, each of the remaining two

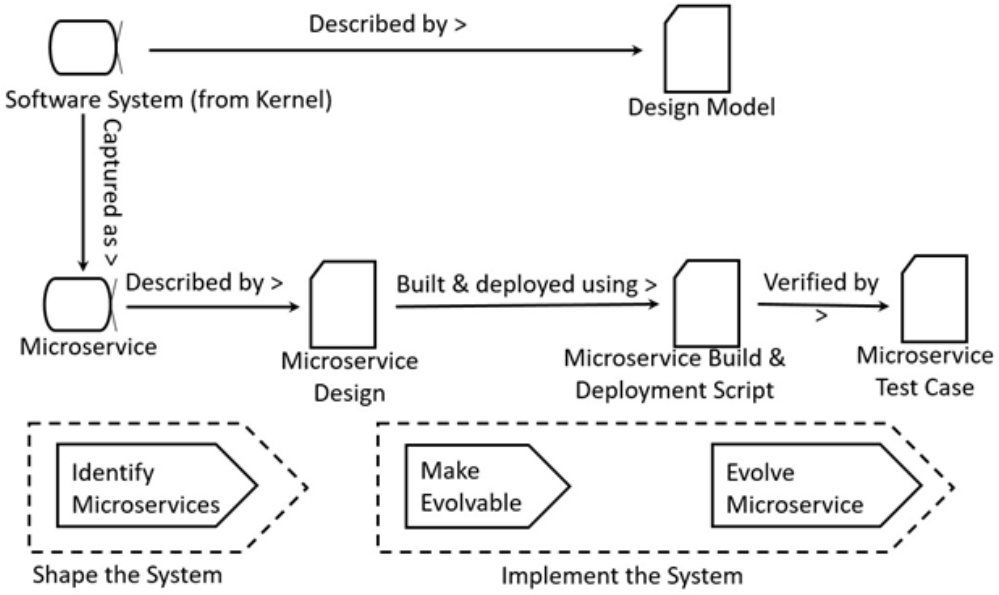

Figure 11. The essentialized model showing the microservices lite practice.[16]

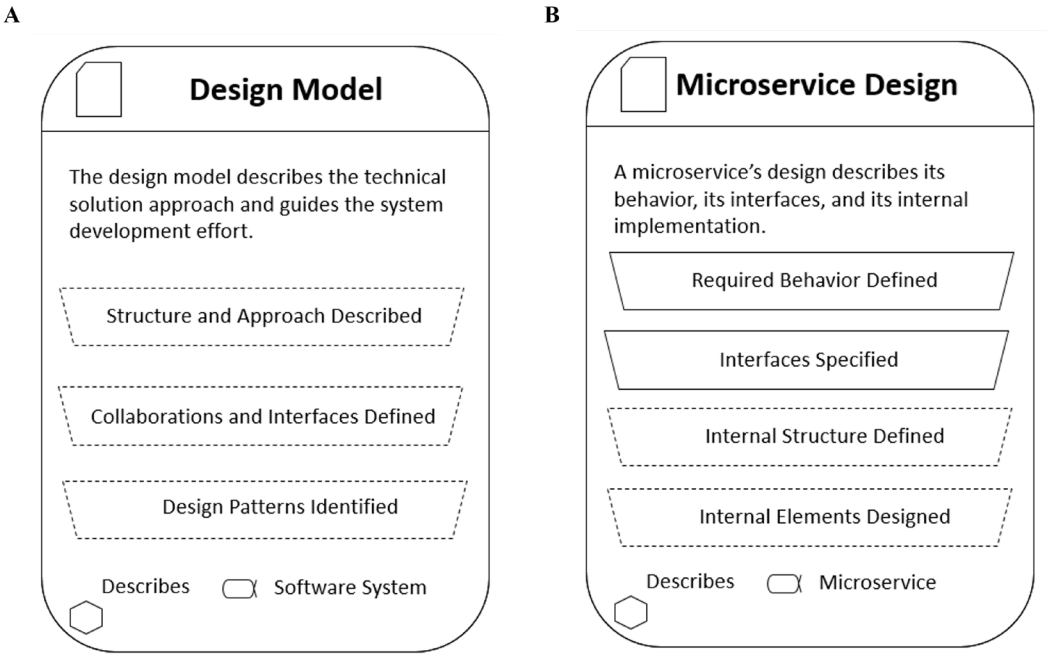

Figure 12. Work product card. (A) Design model. (B) Microservice design.

microservices, *Nirmanik::De-duplication of leads* and *Nirmanik::Lead tracking and follow-up for visit confirmation*, provides an iLeads interface to leads with a progressive amount of refinement (or processing) for subsequent stages. In addition, both microservices have an iService interface to handle execution.

A key tenet of implementing distributed systems is to use business capabilities as the design element, moving away from the data-centric design.[24] We subsequently consider adopting a publish/subscribe workflow, keeping in mind that a message-oriented implementation is an effective way to keep the shared interfaces between components up-to-date (with appropriate refactoring), and asynchronous message-passing would aid loose coupling, a desired trait of a microservice architecture.

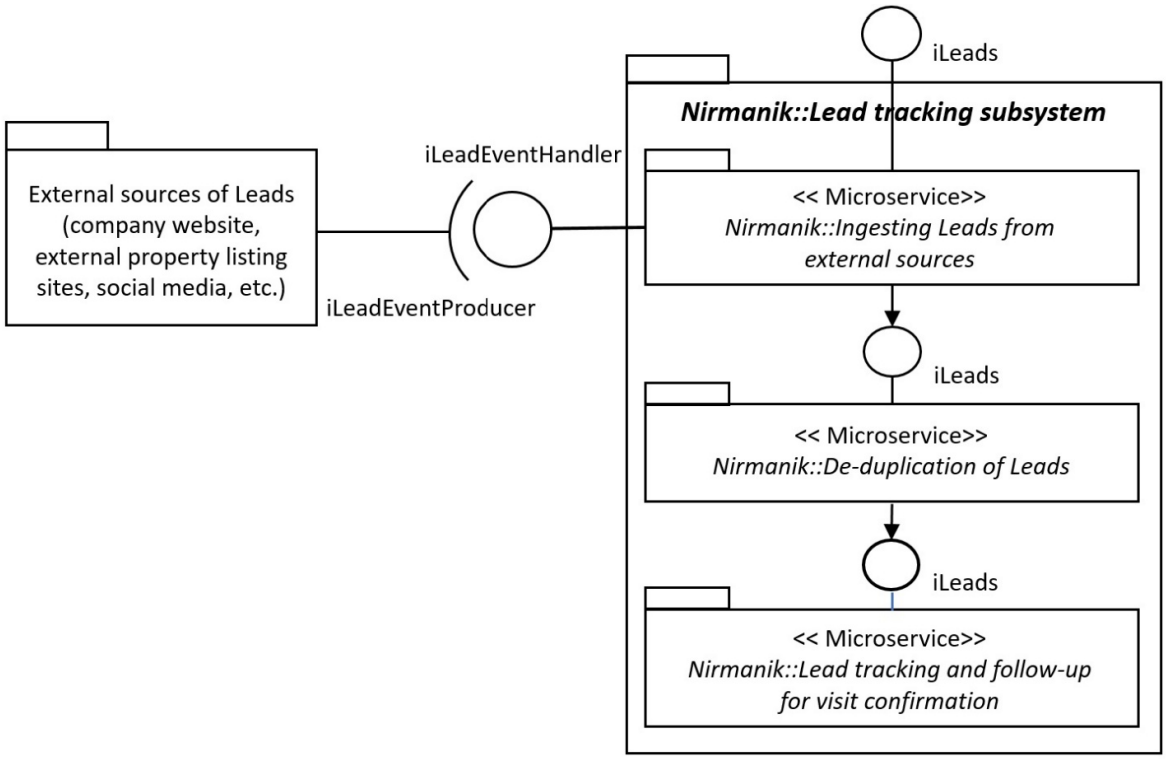

Figure 13. Design model for Nirmanik::Lead tracking subsystem.

6.3. The value of the kernel to the microservices lite practice

Section 5.3 shows that two of the kernel solution activity spaces, "shape the system" and "implement the system," remain unaddressed after adopting the practices at that point.We subsequently adopt the microservices lite practice, which provides implementation guidance for Nirmanik, and is instrumental in covering two kernel solution activity spaces, "shape the system" and "implement the system," as shown in Figure 15.

The adoption of microservices lite practice in addition to user story lite practice already adopted earlier would result in covering all four consecutive kernel solution activity spaces: "understand the requirements," "shape the system," "implement the system," and "test the system," as explained in greater detail in the next section.

7.Conclusion and future direction

While software engineering continues to mature, the success of a software project is still not guaranteed, as both successful and failed outcomes remain prevalent. Agile development is ubiquitous, and Scrum is the most widely used agile framework. Despite its popularity, implementing Scrum poses significant challenges, as mentioned by Scrum's co-founder, Jeff Sutherland.[13] He elaborates how the adoption of Essence, an OMG standard advocated by the Software Engineering Method and Theory community, can help agile development attain its objectives.We have decided to adopt Essence as the unifying framework for its strengths acknowledged by academia and industry alike, and for its ability to describe our evolving practices in a method-agnostic yet structured way. Importantly, Essence also facilitates the creation of new practices and shows how they work alongside existing practices.

We designed a view of the software project execution process in this paper by showcasing how the practices user story lite and microservices lite were adopted on top of Essence Kernel for the CRM automation journey of a real-estate organization, building upon our previous study of adopting Essence as the foundation that leveraged Scrum.[17] In our study, we identified four user stories within the applicable scope, of which the first, *tracking pre-sales leads*, was prioritized and addressed. Then we realized the said user story via microservice practice. Out of the work products associated with the microservice practice, we prioritized the design model and the microservice design work products and developed those two in the context of the user story being considered, *tracking pre-sales leads*, in this paper.

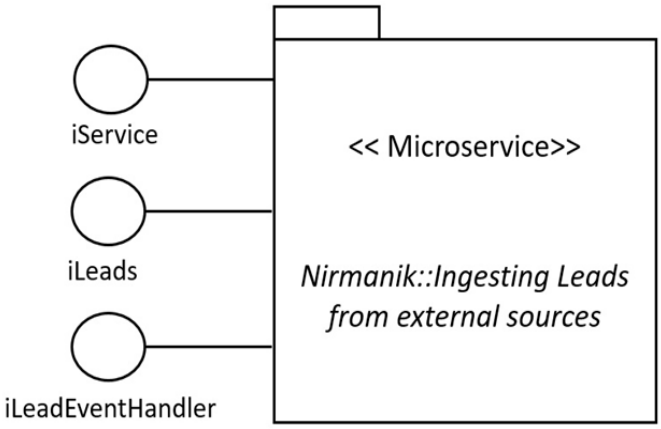

Figure 14. Microservice design of *Nirmanik::Ingesting leads from external sources*.

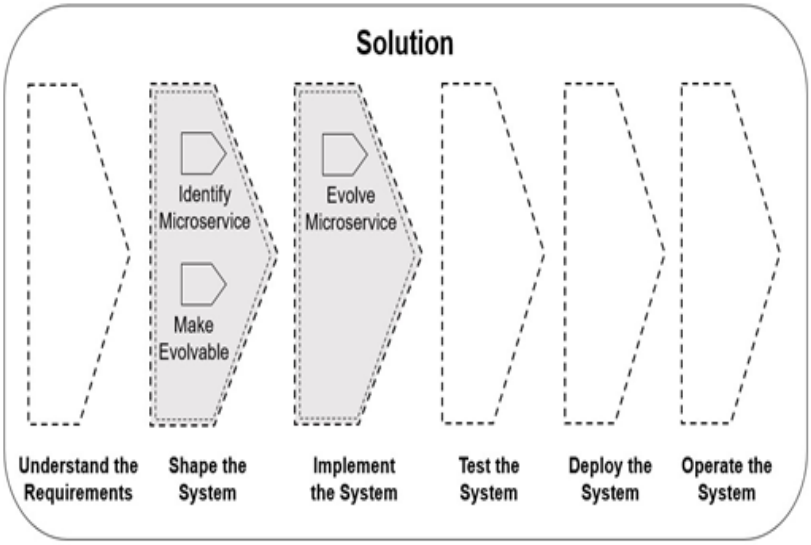

Figure 15. Microservices lite coverage of kernel solution activity spaces.

Figures 9 and 15 show that user story lite and microservices lite practices together cover the four consecutive kernel solution activity spaces: "understand the requirements," "shape the system," "implement the system," and "test the system." This is summarized in Figure 16.

In other words, weaving the user story lite and microservices lite practices into the method composition allows us to address the software development lifecycle spanning requirement analysis, design, development, and testing. We showed how established practices such as Scrum, user story, and microservices can be extended using Essence Kernel as the unifying framework and how those essentialized practices can be composed into a method tailored to a given organization. A key user story was prioritized and addressed by leveraging the user story lite practice. We continued to focus on that user story by developing a set of work products in the microservices lite practice to provide an informed view.

While the exercise described in this paper was conducted for an organization belonging to category C-I, it is also applicable to organizations in category C-II. Although the use of practices like user story and microservices is widespread, it is important to note that simply bringing together such acclaimed practices may not guarantee success. The key success factor is to weave them into a synergistic whole, and that is what we aim to achieve by adopting them into the Essence framework. Our endeavor

is to develop a blueprint that incorporates the said practices, which we have essentialized and composed into a method that can be utilized in scenarios as discussed. To the best of our knowledge, such an exercise has not previously been undertaken in this context and thus represents a unique contribution of this paper.

The usefulness of the resulting method can be extended beyond the industry considered here and the domain, as it would have broad applicability by providing guidance for similar project implementation efforts in other industries and domains. While the user story and the microservice practices are almost ubiquitous in their own right, the exercise conducted in this paper enabled us to essentialize these practices and compose them into a method, as guided by the Essence framework.

This study has the potential to be extended to a broader scope, serving as a blueprint for other industry/domain settings that require selecting practices different from those used here. The principles of our work here can be leveraged, keeping Essence as the central framework, essentializing the new practices, and composing them into a method to help meet the objectives for the industry/domain setting being considered.

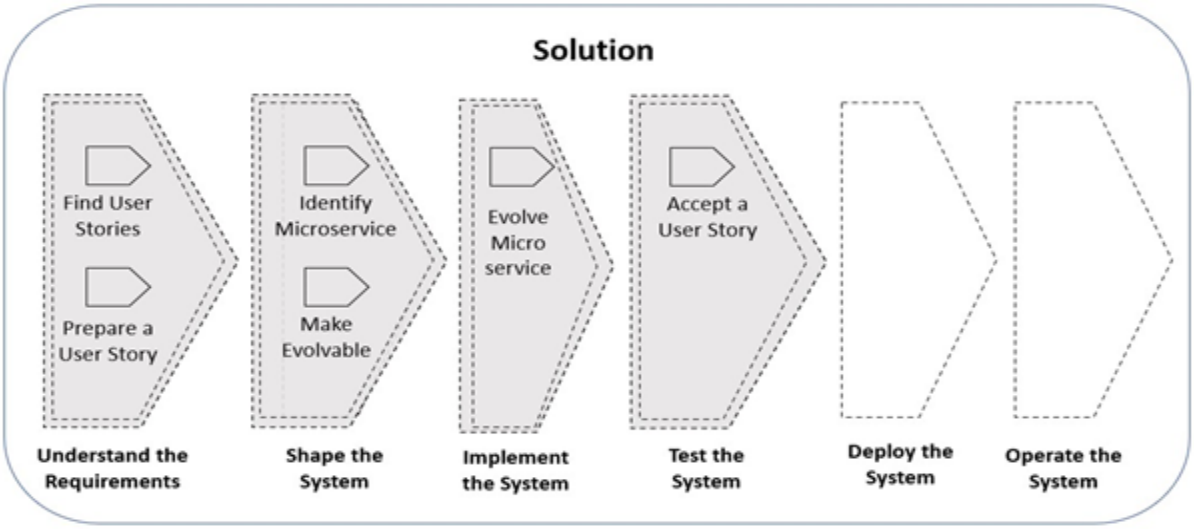

Figure 16. Coverage of the composition of the user story lite and microservices lite of the kernel solution activity spaces.

Moving forward, future studies should take up the remaining user stories identified and develop the work products in the microservices lite practice for each to provide a full-fledged view. This would allow us to progress the "requirements" alpha from the conceived state to the addressed state, and the software system alpha from the "architecture selected" state to the "usable" state. The second user story should detail the process logic for a workflow to assign leads for site visits to suitable sales executives, subject to various constraints and rescheduling or preference changes, which might lend itself well to Petri net modeling.

Acknowledgments

The authors would like to thank the Indian Statistical Institute for providing the support and infrastructure to carry out this study. We would also like to thank the client stakeholders we had interacted with, along with their team members, during our erstwhile CRM consulting engagements, for providing insights and the opportunity to participate in the software endeavors that have proved useful for this paper.

Funding

None.

Conflict of interest

The authors declare they have no competing interests.

Author contributions

Conceptualization: All authors
Formal analysis: All authors
Investigation: All authors
Methodology: All authors
Visualization: All authors
Writing – original draft: All authors
Writing – review & editing: All authors

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data

No datasets were generated or analyzed during the current study.

References

1. CRM (Customer Relationship Management). Salesforce. 2025. Available from: https://www.salesforce.com/in/crm/ [Last access on March 7, 2026].
2. Columbus L. Four interesting insights from Gartner 2020 CRM market share update. Software Strategies Blog. 2021. Availble from: https://softwarestrategiesblog.com/2021/07/09/four-interesting-insights-from-gartner-2020-crm-market-share-update/ [Last access on March 11, 2026].
3. Bain & Company. Customer relationship management.2018. Available from: https://www.bain.com/insights/management-tools-customer-relationship-management [Last access on March 7, 2026].
4. Moore S. Gartner Says Worldwide Customer Experience and Relationship Management Software Market Grew 15.6% in 2018. Gartner. 2019. Available from: https://www.gartner.com/en/newsroom/press-releases/2019-06-17-gartner-says-worldwide-customer-experience-and-relati [Last access on March 7, 2026].
5. Tait. How does real estate affect the economy? Realtyna. Available from: https://realtyna.com/blog/real-estate-affect-economy/ [Last access on March 7, 2026].
6. Nguyen J. Understand 4 key factors that drive the real estate market. Investopedia. 2025. Available from: https://www.investopedia.com/articles/mortages-real-estate/11/factors-affecting-real-estate-market.asp [Last access on March 7, 2026].
7. Apgar M. What every leader should know about real estate. Harvard Business Review. Available from: https://hbr.org/2009/11/what-every-leader-should-know-about-real-estate [Last access on March 7, 2026].
8. Apgar M. Managing real estate to build value. Harvard Business Review. 1995. Available from: https://hbr.org/1995/11/managing-real-estate-to-build-value [Last access on March 7, 2026].
9. PwC; Urban Land Institute. *Emerging trends in real estate®: United States and Canada 2023*. Washington, DC: PwC and Urban Land Institute; 2022. Available from: https://knowledge.uli.org/-/media/files/emerging-trends/2023/emergingtrendsunitedstatesandcanada2023.pdf [Last access on March 7, 2026].
10. Kreyon. 10 ways business process automation is changing real estate. 2016. Available from: https://www.kreyonsystems.com/Blog/10-ways-business-process-automation-is-changing-real-estate/ [Last access on March 7, 2026].
11. Shaw R. *Computer aided marketing and selling: Information asset management (The marketing series)*. Butterworth-Heinemann; 1991.
12. Object Management Group. Essence: kernel and language for software engineering methods version 1.1 with change bars. 2015. Available from https://semat.org/documents/20181/57862/formal-15-12-03.pdf/formal-15-12-03.pdf [Last access on March 7, 2026].
13. Sutherland J, Jacobson I. Better Scrum with Essence [video]. Ivar Jacobson International. Available from: https://www.ivarjacobson.com/videos/better-scrum-essence-jeff-sutherland-and-ivar-jacobson [Last access on March 7, 2026].
14. Ng PW. Integrating software engineering theory and practice using essence: A case study. *Sci Comput Program*. 2015;101:66-78.

    doi:10.1016/j.scico.2014.11.009
15. Jacobson I, Ng PW, McMahon PE, Spence I, Lidman S. *The essence of software engineering: applying the SEMAT kernel*. Addison-Wesley Professional; 2013.

    doi:10.1145/2381996.2389616
16. Jacobson I, Lawson HB, Ng PW, McMahon PE, Goedicke M. Essentializing practices. In: *The essentials of modern software engineering: free the practices from the method prisons*! Association for Computing Machinery; 2019.

    doi:10.1145/3277669.3277694
17. Ray P, Pal P. An agile approach to automate real estate CRM (pre-sales) using Scrum and Essence. In: Proceedings of the Conference on Software Engineering Research & Practice (SERP); 2021. Available from https://www.american-cse.org/static/CSCE21%20book%20abstracts.pdf [Last accessed on 7 March, 2026].
18. Kock N, Verville J, Danesh-Pajou A, DeLuca D. Communication flow orientation in business process modeling and its effect on redesign success: results from a field study. *Decis Support Syst*. 2009;46(2):562-575.

    doi:10.1016/j.dss.2008.10.002
19. Cohn M. *User stories applied for agile software development*. Addison-Wesley; 2009. Available from https://athena.ecs.csus.edu/~buckley/CSc191/User-Stories-Applied-Mike-Cohn.pdf [Last accessed on March 7, 2026].
20. Ray P, Pal P. Extending the SEMAT kernel for the practice of designing and implementing microservice-based applications using domain-driven design. In: Proceedings of the 2020 IEEE 32nd Conference on Software Engineering

Education and Training(CSEE&T). IEEE; 2020:1-4.

doi:10.1109/cseet49119.2020.9206200

21. Lewis J, Martin F. Microservices. MartinFowler.com. Available from https://martinfowler.com/articles/microservices.html [Last accessed on 7 March, 2026].
22. Namiot D, Sneps-Sneppe M. On micro-services architecture. *Int J Open Inf Technol*. 2014;2(9):24-27.
23. Heinrich R, van Hoorn A, Knoche H, *et al*. Performance engineering for microservices. In: Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering Companion (ICPE Companion). ACM; 2017:223-226.

doi:10.1145/3053600.3053653

24. Newman S. *Building microservices*. O'Reilly Media; 2015.