Before moving forward with full-scale development, software developers utilize both Architectural Spike and Proof of Concept (PoC) methodologies to evaluate ideas, investigate technologies, and reduce risks. Even though they are both exploratory in character, they have different goals and are carried out at various phases of the development process.

Architectural Spike

An Architectural Spike is a focused investigation meant to address a particular technical risk or uncertainty in a project. In order to verify viability and provide guidance for ensuing development decisions, it is frequently used to assess various architectural approaches, technologies, or solutions. Time-boxed Architectural Spikes produce in-depth analysis and suggestions rather than code that is suitable for deployment.

GeekUp Scenario Example:

Scenario: GeekUp hopes to apply a machine learning system to precisely detect counterfeit goods through item scanning.

The architectural spike method

Determine the Technical Uncertainty:

The technical team determines the unknowns around the machine learning algorithm's implementation, including the need for computational resources, the choice of machine learning framework, difficulties with data pretreatment, and model accuracy.

Conduct Spike: Give the group a fixed amount of time to investigate various machine learning frameworks, test out data preprocessing methods, train preliminary models, and assess how well they perform in comparison to a dataset of real and fake products.

Result: The spike provides light on the performance, viability, and potential difficulties of putting the Machine Learning algorithm into practice. A report summarizing the results, suggestions for the optimal machine learning framework based on scalability and performance indicators, and an estimate of the resources required for full-scale development are possible components.

Making Decisions: GeekUp can confidently move forward into the development phase, allocate resources suitably, and decide on the Machine Learning method based on the spike findings.

Proof of Concept (PoC)

A more complex prototype known as a Proof of Concept (PoC) shows the viability and potential of a certain feature, functionality, or product concept. A Proof of Concept, in contrast to an Architectural Spike, aims to produce concrete outcomes, often including a practical prototype exhibiting essential features. PoCs assist stakeholders in putting concepts into perspective, getting early user feedback, and validating assumptions before making significant development investments.

GeekUp Scenario Example:

Scenario: In accordance with user preferences, GeekUp wishes to introduce a service that lets users remember searches and get notifications.

Method for Proof of Concept:

Describe the scope: Define the Proof of Concept 's scope accurately, taking into account important functions like search storing, setting up notifications, and managing user preferences.

Prototype Development: Create a simplified version of the application with the search saving and notification functionalities as its only priorities. To start, use fake information or streamlined algorithms to show off fundamental capabilities.

User Testing: To acquire input on the usability, efficacy, and appeal of the search storing and notification capabilities, do user testing sessions with a small sample of target users (collectors).

Reactions and Rework: Repeat on the proof-of-concept (PoC) prototype in response to user feedback in order to improve feature interactions, user interface and user experience aspects, and any encountered usability or technical problems.

Validation and Decision Making

Verify that the proof of concept satisfies user requirements, is consistent with organizational objectives, and can be workaday incorporated into the entire application. Make decisions about the development plan, feature prioritization, and budget allocation for the main app development based on the insights gathered from the PoC.

Key Differences

Objective: While Proof of Concept focuses on obtaining stakeholder feedback and validating user-facing features, Architectural Spike aims to resolve technical uncertainties and make well-informed technology decisions.

Scope and Output: Proof of Concept has a wider scope and produces a workable prototype or demo exhibiting specific features, whereas Architectural Spike is more focused and generates insights and recommendations.

Timing: Proof of Concept takes place throughout the design/prototyping phase to confirm user needs and viability, while Architectural Spike usually happens early in the project to inform architectural decisions.

Task 7

A

A timebox is a set amount of time that is assigned to particular tasks or activities in project management. Enhancing focus, controlling scope, and ensuring on-time project milestone delivery are the objectives. By establishing boundaries, timeboxing facilitates effective task prioritization and productivity among teams.

Timeboxing is a project management technique that involves allocating a fixed amount of time (a timebox) to complete a specific task or phase of a project. Timeboxing helps teams stay focused, prioritize tasks, and manage expectations regarding project timelines and deliverables.

In the context of the GeekUp project, timeboxing can be applied to various aspects of the development process to ensure efficient use of resources and timely completion. Let's illustrate a timebox for the project based on the given scenario and requirements:

Timebox: Development Phase

App Development (Android):3 months

Justification: Developing the app for Android will require substantial effort, including UI/UX design, backend integration, payment gateway setup, and testing. Given that it's the initial platform, it will require significant attention to detail and testing to ensure a smooth user experience.

App Development (iOS): 2 months

Justification: Following the Android development phase, porting the app to iOS will require less time since the core functionalities and design elements would already be established. However, iOS-specific optimizations, testing, and App Store compliance checks will still need to be performed.

Machine Learning Algorithm Integration: 1 month

Justification: Integrating the machine learning algorithm for item scanning and counterfeit detection will require dedicated time for training the model, fine-tuning algorithms, and integrating the feature seamlessly into the app's workflow. This phase may involve collaboration with data scientists or AI experts.

User Account and Authentication System: 1 month

Justification: Implementing user account creation, login/authentication via Google, Facebook, and other options, along with account management features, will require backend development, security considerations, and integration with third-party APIs.

Search and Notification Features: 1 month

Justification: Developing search functionalities, saving searches, and setting up notifications (custom reminders and new item alerts) will involve frontend and backend development, database management, and ensuring real-time updates without compromising app performance.

Payment Gateway Integration: 2 weeks

Justification: Integrating payment options like PayPal, Visa, etc., requires secure payment gateway integration, testing transaction flows, and ensuring compliance with payment processing standards.

UI/UX Optimization and Performance Testing: Ongoing throughout development phases

Justification: Continuous optimization of the user interface (using images/icons over text where possible), ensuring responsive design across devices, and performance testing on devices up to five years old are crucial for user satisfaction and app success.

Justification for Estimates:

- The estimates are based on the complexity and criticality of each development phase.

- Android development is allocated more time due to initial platform setup and comprehensive testing requirements.

- iOS development takes less time as it leverages existing designs and functionalities from the Android version.

- Machine learning integration requires time for model training, testing, and integration complexities.

- Other features like user authentication, search/notification systems, and payment integration are estimated based on typical development timelines for such functionalities.

By timeboxing the project, GeekUp can manage resources effectively, prioritize tasks, and ensure timely delivery of key features while maintaining quality and user satisfaction. Regular progress reviews and adjustments can be made during each timebox to stay aligned with the overall project timeline and goals.

|  |  |  |  |
| --- | --- | --- | --- |
| **Time box Name** | | Initial development phase | |
| **Duration** | | 3 month | |
| **Task** | | **Duration** | |
| App Design and Planning | |  | |
| Basic App Functionality | |  | |
| Machine Learning Integration | |  | |
| Performance Optimization | |  | |
| **Task** | **Duration** | 17.1.2024 | 17.1.2024 |
| Functional Requirement | 1 day | 19.1.2024 | 19.1.2024 |
| Use Case Diagram | 1 day | 20.1.2024 | 20.1.2024 |
| Class Diagram | 1 days |  |  |
| Sequence Diagram | 1 days |  |  |
| High Level & Low-Level Prototype | 1 days |  |  |
| Coding | 10 days |  |  |
| Functional Testing | 1 day |  |  |
| Usability Testing | 1 day |  |  |
| Time Box Summary | 1 day |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Time box Name** | | Manage Purchase Process Time-box | |
| **Start Date** | | 1.1.2024 | |
| **End Date** | | 20.1.2024 | |
| **Task** | **Duration** | **Start Date** | **End Date** |
| Functional Requirement | 1 day | 1.1.2024 | 1.1.2024 |
| Use Case Diagram | 1 day | 2.1.2024 | 2.1.2024 |
| Class Diagram | 1 days | 3.1.2024 | 3.1.2024 |
| Sequence Diagram | 1 days | 4.1.2024 | 4.1.2024 |
| High Level & Low-Level Prototype | 1 days | 5.1.2024 | 5.1.2024 |
| Coding | 10 days | 6.1.2024 | 16.1.2024 |
| Functional Testing | 1 day | 17.1.2024 | 17.1.2024 |
| Usability Testing | 1 day | 19.1.2024 | 19.1.2024 |
| Time Box Summary | 1 day | 20.1.2024 | 20.1.2024 |