**Single case results**

*Case 1*

In stage 1, Case-1 focused on developing the major business services for the ERP product such as accounting, inventory, and sales modules. One technical founder and three contracted developers worked using an ad-hoc development process. After two years, Case-1 released its initial product (MVP) to market in 2014. Case-1 made three TD decisions that accumulated TD. The first decision was **reusing an existing open source ERP software (ERPNext[[1]](#footnote-1))**, and making a few customizations to satisfy the needs of the target market. The second decision was to **skip the upgrade of the development framework (WebNotes**[[2]](#footnote-2)**)** which was embedded within the open source ERPNext software. At that time, the ERPNext open source community switched to a new framework called Frappe[[3]](#footnote-3). However, Case-1 decided to continue using WebNotes. The third decision was to **skip performing systematic testing**, in favor of quick and ad-hoc testing.

In stage 2, the development team increased to five and adopted an agile Kanban-based methodology. After about one year into stage 2, the bug rate surged to a high level. According to the release note history, which were publicly available on the Case-1’s website, the releases within that timeframe included fixes for about 150 bugs. Case-1 searched for ways to mitigate the high bug rate issue. Given the resource limitations, the team determined the simplest and cheapest solution which was to **improve the test process by adopting code reviews** instead of systematic testing. One senior developer was assigned to review the code before releasing any change. Case-1 found its product-market fit in 2018.

In stage 3, the development team doubled to 11 people. The team was organized into two groups: development and technical support. They adopted a Scrum development process. The negative impacts of the previously incurred TD were amplified. The infrastructure cost increased significantly as a result of the growing number of customers. Using the WebNotes framework (which adopts a single tenancy architecture), forced the team to create independent host infrastructures for each customer. The issue was further exacerbated when the python foundation announced[[4]](#footnote-4) the sunsetting of Python version 2, making it impossible to implement some new desired features becuase the WebNotes framework was built using Python version 2.7. This led Case-1 to urgently discuss possible alternatives to the legacy framework and ways to improve the software quality. However, it was difficult and expensive for Case-1 to address the legacy framework. Instead, the team **decided to adopt a modern infrastructure and architecture for the new features only, while keeping the existing legacy code as is**. In this approach, new features were added on the new development infrastructure while keeping the legacy WebNotes framework for the old features. While implementing a new feature, the team leveraged the new development infrastructure to **add some unit tests for the newly implemented features**. So testing was incorporated as part of new feature development.

*Case 2*

In stage 1, the four-person development team followed an ad-hoc development process. Three decisions were made that accumulated TD. First, the MVP was developed on top of a previously developed prototype, which was **implemented as one monolithic solution with a lack of modularized code**. The second decision was **selecting a sub-optimal development framework (CakePHP)**, because of familiarity with it on the team even though they acknowledged that CodeIgniter or Symfony were the better options for Case-2’s application. Finally, due to time constraints, the team decided to **skip addressing the code standard and design violations identified by the framework**. Each developer wrote code using their own coding style.

In stage 2, the development team focused on stabilizing the product in one geographic area. The team increased to 7-8 people, forming one agile scrum team. The team formalized the development process, and attempted to incorporate testing and quality checks prior to releasing a new version. However, the team decided to **skip pre-release testing**, but rather monitored the software in the production environment and fixed issues whenever they were observed. Later in this stage, Case-2 tried to upgrade the CakePHP framework from version 2 to version 3, requiring some manual changes in the code to satisfy the new framework requirements. But it was not clear which parts of the code needed to be changed to successfully perform the upgrade. So the team decided to **skip the upgrade and continue using the existing version of the CakePHP framework (version 2)**.

In stage 3, Case-2 expanded the product globally to over 13 countries. The development team grew to approximately 26 people distributed over four scrum teams. During the first year at stage 3, Case-2 encountered some obstacles to evolving and scaling the software. Some features and enhancements could not be implemented due to the existing system complexity and framework limitations. Also, the outdated CakePHP framework deteriorated the system performance (primarily the latency). So the team decided to **migrate from the monolithic software architecture to a microservice architecture**. The migration process required transforming existing modules to small and independent services. As part of the process of migrating a module to a service, the team took the initiative to **refactor the module code, and adopted an updated infrastructure technology for the new service**. For example, they adopted Kotlin for the Android applications, which enhanced maintainability. The migration to a microservice architecture introduced some short-term issues, including an overwhelming increase in bugs. So the team decided to **include systematic testing process**, which was designed and implemented only for microservices. Case-2 achieved its maturity goal by 2020.

*Case 3*

In stage 1, two founders, who were computer science students, developed the initial version of the mobile app during the summer of 2019. They used pair programming with no structured development process, until eventually a third developer joined the team. During this stage, three TD decisions were made. First, the team decided to **write the entire application using a scripting language without any use of a strongly typed language**. The advantage was that most of the team was experienced in front end development, so scripting allowed the team to quickly develop and experiment with the software. The second and third TD decisions were related to the database infrastructure. First, the team decided to **adopt the MySQL database** since it was easier for them to work with, hence, faster development. A short time later, the team realized that MySQL has some limitations when dealing with geographic data, which restricted the implementation of an important feature. So the team decided to **switch the database to PostgreSQL**.

In stage 2, the development team continued with three members and an ad-hoc process. Although Case-3 designed some unit tests earlier (in stage 1), the team decided to **skip updating and implementing the test** due to time limitations and requirement uncertainty. Case-3 wanted to allocate their time on iterating the product based on users’ feedback before investing in testing. In July 2020 (at the time of this case study), the team was working to add the push notification feature, which was complex to implement, requiring a lot of time searching on the internet. The final design was not optimal, as it was designed with high dependancy on two different external libraries. Thus, Case-3 **designed and implemented the push notification feature in a suboptimal way**.

*Case 4*

In stage 1, , the development team was two technical founders and two freelance developers, geographically distributed, using an agile methodology with online weekly sprint cycles. During this stage, two TD decisions were made. First, Case-4 decided to **skip the integration testing** due to time and budget constraints. The second decision is related to the implementation of the search feature. Initially, Case-4 hosted its infrastructure on Amazon Web Service (AWS) and used DynamoDB, a database fully managed by Amazon. The team realized that DynamoDB restricted the search function to just two search indices. However, they wanted to allow users to search for a doctor based on multiple properties such as specialty, city, accepted insurance, etc. To quickly overcome this limitation, the team **designed the search function in a suboptimal way which requiring an additional database called ElasticSearch**. So each microservice had to access two databases: 1) DynamoDB as the primary database, and 2) ElasticSearch to support data retrieval for the search function.

In stage 2, the development team continued with four members using an online weekly sprint. Five TD decisions were made in this stage. First, Case-4 **upgraded its development framework (Angular framework) from Version 7 to Version 9**. Despite the extra effort needed to perform the upgrade, the Case-4’ team decided to pursue this upgrade before the codebase grew. For the second TD decision, Case-4 decided to **use a hybrid mobile app framework (Ionic) to develop a mobile app version of the application**. This allowed the team to develop the mobile app very quickly using one single codebase that could be applied across multiple mobile platforms. However, the design was suboptimal becuase it did not allow full use of special capabilities on some mobile devices, which could restrict the implementation of future features. For the third TD decision, Case-4 decided to **integrate a third party service (Customer.io), resulting in a suboptimal design of the push notification feature**. The team wanted to leverage the first-year-free deal offered by Cutomer.io for early startups. But a few months after the integration, the team discovered that Customer.io did not support notifications for some complex events that involved multiple stakeholders, such as both patients and clinics. So Case-4 decided to **develop a custom API for the push notification, eliminating the need for the Customer.io service**. Lastly, Case-4 decided to **change the web architecture from Single Page Application (SPA) to Multiple Page Application (MPA)**. The main reason was to better support Search Engine Optimization - SEO. To perform this change, Case-4 replaced the AngularJS (which is a single-page architecture framework) with Angular Universal which provides a server-side rendering of the webpage (which is a multiple page architecture framework).

*4.1.5 Case 5*

In stage 1, a simple version of the web application was developed by one of the founders (P-15), without using any structured process. Three TD decisions were made by P-15 in this stage. First, **all business logic was hard coded into the codebase, rather than isolated in a more maintainable business logic component**. This design violation was intentionally committed mainly due to time pressure. Case-5 was using the Ruby on Rails framework, which was flexible and did not strictly enforce the use of a design pattern. In addition, Case-5 adopted the BackboneJS framework to enhance the frontend and user interface design. However, this choice increased maintenance effort because of the need to maintain the API between the two frameworks, which hindered development progress. Thus, Case-5 decided to **remove the BackboneJS and return to relying only on the Ruby on Rails framework**. After completing the development of the initial product, Case-5 **skipped testing completely** because of the time limitation and team shortage.

In stage 2, the development team increased from one to five developers who worked remotely, still fairly unstructured but using Github to communicate and manage the development. Two TD decisions were made in this stage. The first decision was to **quickly develop a mobile app version using a web architecture rather than native**. It was acknowledged by the Case-5 team that this suboptimal design of the mobile app did not support the implementation of the mobile push notification. A few months later, Case-5 noticed that most of its customers (i.e., restaurants) relied on mobile devices for managing orders. To better support the mobile users with push notifications, the decision was made to **re-architect the mobile app from a web to a native application** **for both Android and IOS**.

In stage 3, the team expanded to 10 members, four of them working remotely. Early in stage 3, Case-5 encountered many API integration bugs. This triggered the team to **add some integration tests**,which helped developers to quickly debug the integration issues. Later, the codebase grew and became very complex, which considerably slowed the development. The team agreed on the need to improve the code quality. So they decided to **refactor and redesign the existing code gradually while making code changes**. While refactoring the code, the team decided to **add and improve the test**.

1. https://github.com/erpnext [↑](#footnote-ref-1)
2. https://github.com/webnotes/wnframework [↑](#footnote-ref-2)
3. https://github.com/frappe/frappe [↑](#footnote-ref-3)
4. https://www.python.org/doc/sunset-python-2/ [↑](#footnote-ref-4)