Case Study: Reviving an Old Flash App

Here at Software Planet Group, we deal with all sorts of software projects. Some are brand new, while others are significantly outdated. Recently, we received a request from a customer to work on a project that was created 13 years ago. This was a web-based game developed in 2009 with legacy technologies such as Adobe Flash and the ASP.NET MVC Framework. The actual game mechanics were implemented as Flash content that was embedded in a simple webpage. In other words, the technology behind the project's most essential module *had long been dead* and was all but a zombie.

Unfortunately, such hopelessness is a common thread in a number of legacy projects, and in this particular case, it hardly came as a surprise. After all, Adobe had been keeping Flash player on life support for a number of years now, despite the challenges to maintain its codebase, which was already incomprehensible in places. The platform teemed with temporary solutions and changes imposed by impromptu security patches.

At some point, however, the company realised that the cost of supporting Flash would far outweigh any lingering benefits, so they finally made the rational decision to stop supporting Adobe Flash player for good. This was just as well. At the time, there were already a number of alternatives to Flash features, and due to the sheer amount of security issues that it was constantly introducing into their products, every major browser developer had also abandoned their support for Flash.

The Customer's Objective

Like many businesses in need of a software company for system migration, above all, our customer, *a CEO of a sports advertising company,* wanted to re-evaluate the business potential of their legacy application, in addition to the feasibility of presenting it to potential investors. So with that in mind, they requested that we launch the project from its original codebase, restore the functioning of the parts run by Flash, and provide them with a reasonable time frame of how long it'd take to create a beta version with minimal changes.

The Challenges of Legacy Projects

When dealing with these types of migrations, it is not uncommon at all to have to contend with no software documentation and to reconstruct the ways to deploy the application, both locally and in public environments. This process may be compared to attempting to put together an ancient puzzle — but that is only part of the problem. The *most challenging* part of the equation is making sure the outdated components will work. In this particular project, we had to jump through a number of hoops to make the outdated back-end functional again — as it had been written with .Net 4.5. In the end, we were able to obtain a working solution by migrating it to .NET 4.8 (2019).

How to Launch a Flash Application

Now, if you are wondering how to renovate a legacy system in such unfavourable, horrific circumstances, traditionally, the way to get away from using Flash is to replace it with a newer technology. The game element, in such an event, could be re-developed using HTML5. But before examining this approach, we first needed to launch the application and discover exactly what our team was dealing with. On top of that, *it all had to be done at minimal cost* (as close to £0 as possible). So with that in mind, after familiarising ourselves with the codebase and the artefacts at our disposal, we identified three possible courses of action:

- use a Flash emulator — e.g. https://ruffle.rs — to run Flash-based components in a modern browser;
- install an older browser version which supports Adobe Flash player and can launch the application from there;
- use Flash Browser — a special browser with a built-in flash player.

→ Use a Flash emulator

The emulator option, if successful, would provide far wider applicability, so we decided to make it our first attempt. Unfortunately, however, as we soon found out, Ruffle is unable to play third-party flv video files. At the time of writing this article, this unfortunately still remains the case. In practice, we were blocked by the following exception:

```
ruffle_web.js:1235 Error running definition tag: DefineVideoStream, got Invalid data: Invalid video codec.
```

There are also certain limitations on supported video file formats, so these must sometimes be converted prior to use. What all of this essentially meant was that the approach was not a suitable option for *this particular* project. The approach itself, however, is still a valid one and should be considered in similar scenarios, as you might well be able to embed an flv into your existing swf files. It is important to also note that Ruffle might eventually add support for external video files in future.

→ Use older browser versions

Moving on, we also had to fix a number of compatibility code defects (both at the frontend and backend of the project). This would enable us to launch the application using older versions of Chrome and Firefox, where support for Flash Player was still intact. At last this yielded some positive results.

→ Use a Flash-playing browser

Nonetheless, because we were not entirely happy with the idea of using outdated browser versions, we decided to give the chromium-based Flash Browser a try. This is an open source product which significantly reduces security worries.

Bingo. Not only was the game successfully launched here, but we finally had a working prototype which demonstrated the game’s UI, mechanics and gameplay. From here on out, we were finally able to plan the game’s entire redevelopment with a brand new tech stack to boot.

Typical Challenges when Renovating Legacy Projects

Our developers were extremely lucky to have fla files on hand as well (animation project files created by the Adobe Animate program), in addition to swf ones. Otherwise, we would have had to do some serious reverse engineering by “disassembling” the swf. This process does not often lead to the desired outcome and makes it practically impossible to introduce code changes *without breaking the app’s ability to launch*.

In the absence of the fla files your swf files depend on, your chances of quickly launching your project become next to zero. In such a scenario, your best bet would be to attempt to rewrite the Flash functionality completely, by utilising an HTML5 canvas, for instance.

Another common concern is that the version of ActionScript that was used in the old fla files may also be severely outdated. In many cases, to introduce changes, developers have no choice but to rewrite the existing code with the latest ActionScript version, or re-develop the same functionality with the help of more modern technologies.

System Migration: When Is the Time Right?

Many of the problems described above may be mitigated or avoided entirely when a migration is expected and planned for. This is actually easily done, as the vendors of most libraries, frameworks and tools will seek to inform users about the end of life or deprecation of their software products. However, it is important not to leave migration tasks until the very last minute. After all, any problems encountered in production will already signal the urgency of the matter.

On the other hand, if left unresolved, your software solution may eventually be unable to be migrated, and restoring it to working order will call for a lot of time, effort and money.

Thankfully, if you are looking for a system or data migration software company, at SPG, we have over twenty years of experience handling numerous types of migration projects. We can assist you with the following requirements:

1. Upgrading whole technology stacks
2. Replacing obsolete solutions with available alternatives
3. Examining the overall feasibility of bringing your company’s dream products to life