## Battling Technical Debt

The purpose of the MashupJS is leveraging past experience to reduce **technical debt** moving forward.

Technical Debt according to Wikipedia.org:

<http://en.wikipedia.org/wiki/Technical_debt>

“Technical debt (also known as design debt or code debt) is a recent metaphor referring to the eventual consequences of poor system design, software architecture or software development within a codebase. The debt can be thought of as work that needs to be done before a particular job can be considered complete or proper. If the debt is not repaid, then it will keep on accumulating interest, making it hard to implement changes later on. Unaddressed technical debt increases software entropy.”

"As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."

— Meir Manny Lehman, 1980

## Examples of Technical Debt Reduced by the MashupJS

### Multiple Code Bases

Maintaining multiple code bases for the same application.

**Example**

* Mobile application where each hardware platform has its own code implementation.
* Multiple copies of a project because of multiple .NET Frameworks.

**Cost**

* **Resource Cost** – When dealing with multiple hardware platforms, costs are increased with the number of platforms. Each platform requires a dedicated, highly skilled resource or broader skilled resources that understand multiple programming ecosystems.
* **Increased Risk** – Multiple code bases require increased testing, bug fixing, and introduce the risk that a bug fix might not make it into all code bases.

**Solution**

* The power in web technologies is the ability to run on all platforms that support a web viewer. Using a hybrid mobile approach with Cordova, Ionic, and Angular with Responsive Design, it is possible to build an application capable of running from a web browser, desktop, Android, iOS, and Windows 10 devices with a single code base.

### Lacking Depth of Knowledge

The greater the number of technologies in an ecosystem, the more shallow knowledge and experience become.

**Example**

* Over time, applications are developed for the business, but never kept up to date.
* After a decade, in a small shop, an ecosystem might have the following:
  + Two classic ASP applications
  + Three classic ASP.NET forms applications
  + Two ASP.NET MVC applications
  + One Silverlight/EF4 application
  + Two modern web Angular/WebApi applications
  + Web Services, WCF, and WebApi

**Cost**

* **Increased Cost** – Cost increases as skills diminish. With so many technologies, the number of staff required is far higher than it would be if the technologies were in sync.
* **Increased Cost** – With so many technologies, knowledge depth decreases. With decreased knowledge, support costs increase.

**Solution**

* The Mashup Core gives all applications a common set of components and services. When a core component is enhanced, all applications are enhanced.
* Adopting a process of Continuous Improvement, the effort to keep applications updated is far less than with unexpected rewrites.

### Abandoned working code

This is code for an application that is abandoned because it works. Over time it’s forgotten and remains unattended. The day comes when the business needs new features or new OS changes and breaks the application.

**Example**

* A nightly batch process/program that has run well for many years. It copies critical application data to applications that require it. Over time, the IT organization has forgotten about this critical process.

**Cost**

* **Uncertainty** – Unknown dependencies.
* **Poor Estimates** – Your customer/product tables might be updated by a nightly process that is long forgotten. Estimates and support roles forget to take this into account because it has always worked. Estimates cannot take into account processes it does not know about.
* **Unplanned Rewrite** – Once discovered, support for abandoned code often requires a complete rewrite. Only then can new features can be added or the application be considered supported. Rewrites occur when a code base is so out of date it cannot be cost effectively refactored.

**Solution**

* The Mashup attempts to combine applications using a single core. Even in cases where an application requires little attention, its core components are kept up to date. As new techniques are adopted, linters are used to identify code that needs updating, even in unattended code. The application remains updated, visible, and not forgotten.

### Device dependence

Device dependence forces the business into dependency on a device platform as a result of writing hardware platform specific applications.

**Example**

* An IT department writes a series of mobile applications for iOS because, at the time of writing code, all corporate phones were iPhones. At some point the business is offered a significant savings on the Android platform but this becomes a lost opportunity because of the iOS hardware dependency.

**Cost**

* The cost of smartphones and tablets varies as competition increases. A vendor offering can easily cut a company’s smartphone devices in half. This can be a significant lost opportunity because of decisions made by the IT department.

**Solution**

* Don’t lock your company into a hardware platform. Use the Mashup, Angular, Bootstrap, Ionic, and WebApi to create platform independent applications.

### Fear of Breaking Interfaces

Interfaces allow components to be consumed by other components and applications to talk to other applications. Rather than change an existing interface that might be consumed by another component or application, the approach has been to bolt on additional code.

**Example**

* A component, representing Customer, is being used by multiple applications within an organization. After merging with another company, the definition of Customer has changed. Rather than alter the Customer to reflect the new reality, the old reality of Customer is maintained while also supporting a new reality of Customer.
* Another example of the fear of breaking interfaces is the web browser, Internet Explorer. Microsoft held fast to its commitment to supporting businesses that created applications based on the Internet Explorer platform.
* Over time, code that has been bolted onto becomes brittle and easily broken and then it becomes the “Avoided Code.”

**Cost**

* **Complexity** – The increased complexity and, often, harder to read code increases the cost of ownership for the system.
* **Internet Explorer –** The IE6 browser became famous for being slow and not keeping with the times. Nearly all vendors have dropped Internet Explorer through version 9 as a result. Microsoft is even abandoning Internet Explorer in favor of their new browser, “Spartan,” which holds no allegiance to past applications.

**Solution**

* Abandon the philosophy of protecting existing interfaces. Break every interface that does not reflect the current reality. This can only be done softly by adoption of a DevOps build process with Continuous Delivery. The Mashup, because of Angular, is uniquely suited for testability within a build process.

### No technical leadership

Lack of technical leadership will create the scenario where too many technologies exist to support. As a result, team knowledge becomes shallow and developers become paralyzed. Without leadership, a team cannot decide where to spend its most precious resource, time.

**Example**

* Front-end development stack – So many options exist for front end development that organizations without strong technical leadership become paralyzed and learning stops.

**Cost**

* **Lost Talent** – Talented developer(s) desire strong leadership so they have confidence their hard work is not wasted. Talent will also not allow a corporation to handicap them in the job market.
* **Dated teams –** Teams become dated because there are too many choices and no guidance.

**Solution**

* The Mashup helps ease the friction of learning the technologies to build modern web applications. The Mashup Core abstracts much of the complexity of Angular and other micro libraries, making SPA programming easy to learn. As skills increase, developers can contribute in more advanced and complex ways.