## List of Philosophy Priorities

* Reduce technical debt
* Mobile First
* (RWD) Responsive Web Design
* Offline First
* Build First

## Battling of Technical Debt

The philosophy 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 by as the number platforms. Each platform requires a dedicated, highly skilled, resource or broader skilled resources that understand multiple programming echo systems.
* **Increased Risk** – Multiple code bases require increased testing, bug fixing, and introduces the risk 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 echo system, 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 echo system 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 unexpected re-writes.

### Abandoned working code

Code for an application that is abandoned because it works. Over time it’s forgotten and unattended to. 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’s always worked. Estimates cannot take into account processes it does not know about.
* **Unplanned Re-write** – Once discovered, support for abandoned code often requires a complete re-write. Only then can new features can be added or the application can become considered supported. Re-writes 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 updated even in unattended code. The application remains updated, visible, and not forgotten.

### Device dependence

Forcing 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 an opportunity lost because of the iOS hardware dependency.

**Cost**

* The cost of smartphones and tablets vary as competition increase. 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 first to its commitment to supporting businesses who created applications based on the Internet Explorer platform.
* Overtime 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 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 to many technologies exist to support. As a result team knowledge becomes shallow and developers become paralyzed. Without leadership a team cannot 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 help 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.
* ***“Lack of building loosely coupled components****, where functions are not* [*modular*](http://en.wikipedia.org/wiki/Modular_programming)*, the software is not flexible enough to adapt to changes in business needs.”*
* ***“Lack of test suite****, which encourages quick and risky* [*band-aids*](http://en.wikipedia.org/wiki/Band-aid_%28Computing%29) *to fix bugs.”*
* ***“Lack of documentation****, where code is created without necessary supporting documentation. That work to create the supporting documentation represents a debt that must be paid.”*
* ***“Lack of collaboration****, where knowledge isn't shared around the organization and business efficiency suffers, or junior developers are not properly mentored”*
* ***“Parallel development*** *at the same time on two or more branches can cause the buildup of technical debt because of the work that will eventually be required to merge the changes into a single source base. The more changes that are done in isolation, the more debt that is piled up.”*
* ***“Delayed refactoring****– As the requirements for a project evolve, it may become clear that parts of the code have become unwieldy and must be refactored in order to support future requirements. The longer that refactoring is delayed, and the more code is written to use the current form, the more debt that piles up that must be paid at the time the refactoring is finally done.”*
* ***“Lack of alignment to standards****, where industry standard features, frameworks, technologies are ignored. Eventually, integration with standards will come, doing sooner will cost less (similar to 'delayed refactoring').”*
* ***“Lack of knowledge****, when the developer simply doesn't know how to write elegant code.”*