## List of Philosophy Priorities

**Reduce technical debt**

All effort must be made to reduce technical debt. We’re building business applications so we must consider cost before all else. The complete cost of ownership including initial development, support, and opportunity costs must be taken into account. Technical debt can be significantly reduced by making sound decisions based on past experiences.

* Reduce the number of technologies in the ecosystem.
* Utilize strong technical leadership that gives direction.
* Develop a strong DevOps component to the development process.
* Adopt Continuous Integration and Continuous Deployment.
* Resist version divergence

**Mobile First**

Whether or not you are designing specifically for mobile, the mobile first approach offers many benefits. When considering the real-estate available for mobile devices, you fall into the lap of success with better imagined UI design and faster running applications. When considering mobile, every aspect of performance from render time to the bits on the network is considered.

Ultimately, it’s far easier to scale the UI up from mobile to desktop than to scale an applications UI down from the desktop.

**(RWD) Responsive Web Design**

Responsive design is critical to delivering applications with a single code base because it takes in all form factors.

**Offline First**

Designing your application to be offline first makes the application more resilient in the face of network instability. Both desktop and web applications benefit from offline first design.

**Build First**

Build continuously and build often. When designing your application, consider how to use Grunt and other tools to continuously build your project. You’ll find errors early.

**Continuous Integration**

Continuous Integration is similar to build first. In a corporate environment, all your shared code is built often. The end result is you no longer have to fear breaking interfaces. Other applications that rely on your shared code will break during the nightly build. Fear of breaking interfaces leads to technical debt and eventually a complete re-write. Internet Explorer is a good case study for what happens when you fear breaking interfaces.

**Continuous Deployment**

DevOps is an emerging approach designed to reduce friction between the developer and operations processes. Continuous deployment to production also reduces risk because many smaller changes are less risky than fewer but larger changes. This approach leads to automation, removing the human element, resulting in further reduced risk.

**Continuous Improvement**

Managing the version of development tools and keeping component versions in sync requires continuous touching of all code. When a new version of a micro library is introduced to the ecosystem, all applications are immediately updated to this new version. When a new technique is discovered, making the effort to implement this technique uniformly lowers technical debt. All of the small changes required to achieve uniformity help avoid unplanned rewrites and forgotten code.