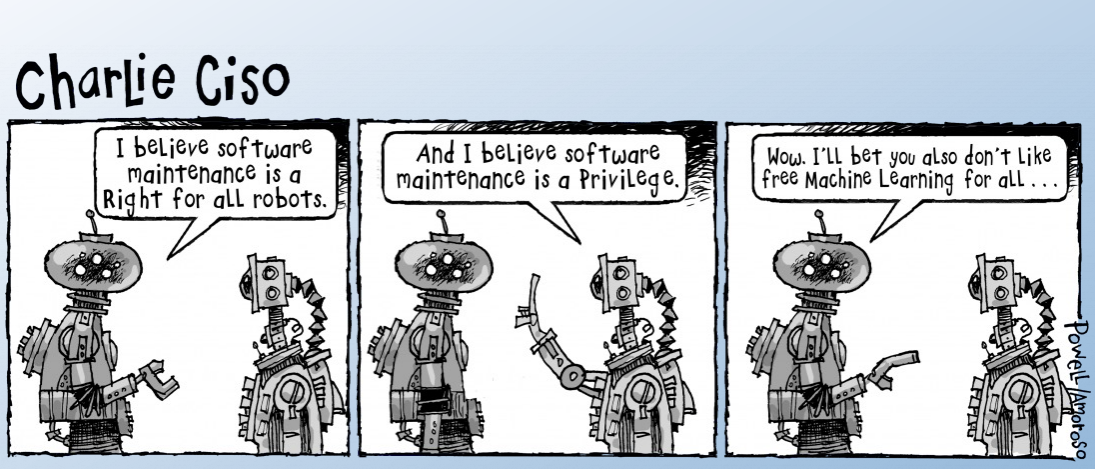
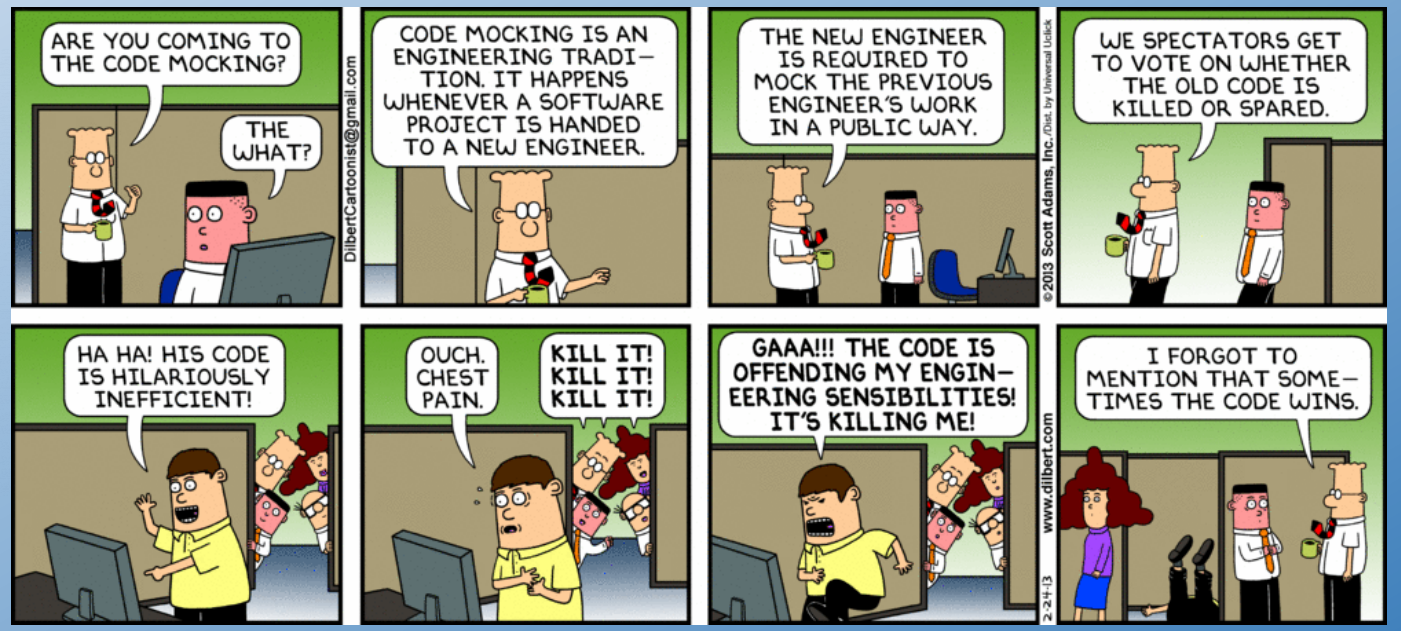
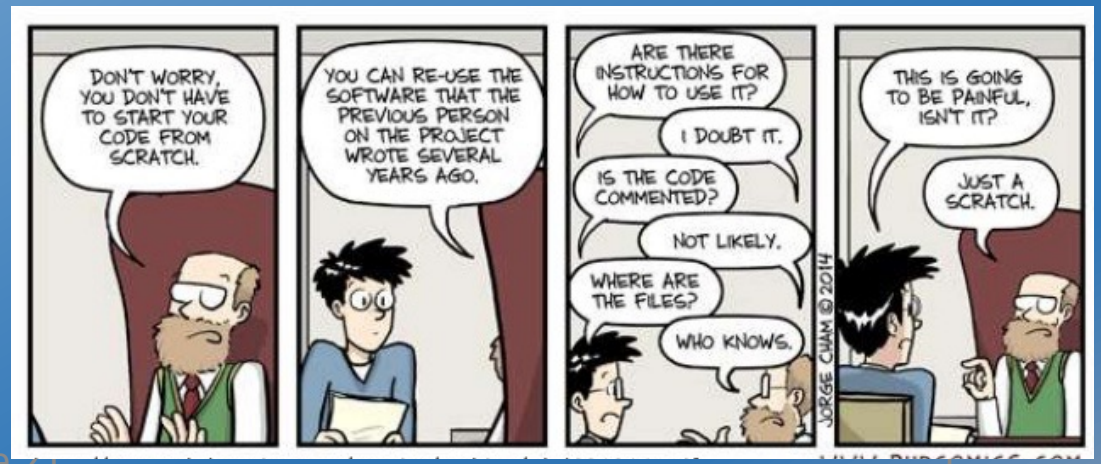




Maintenance I

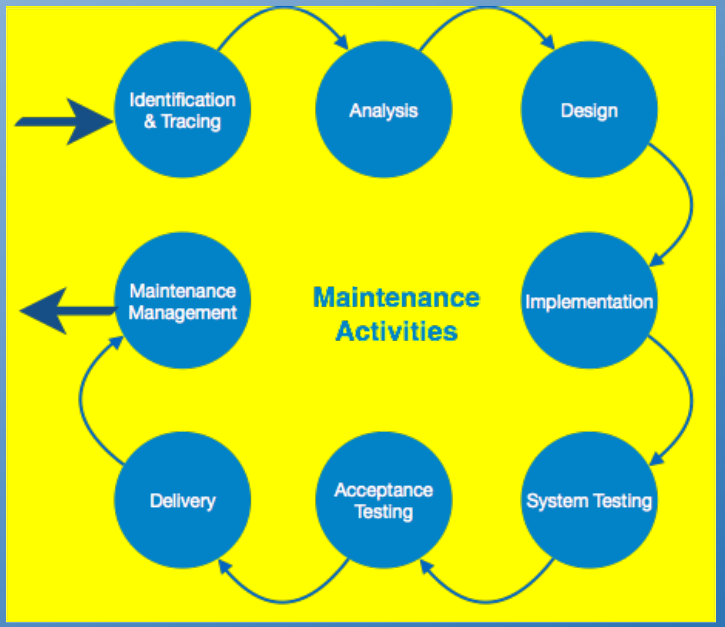
CSCI2340: Software Engineering of Large Systems

Steven P. Reiss

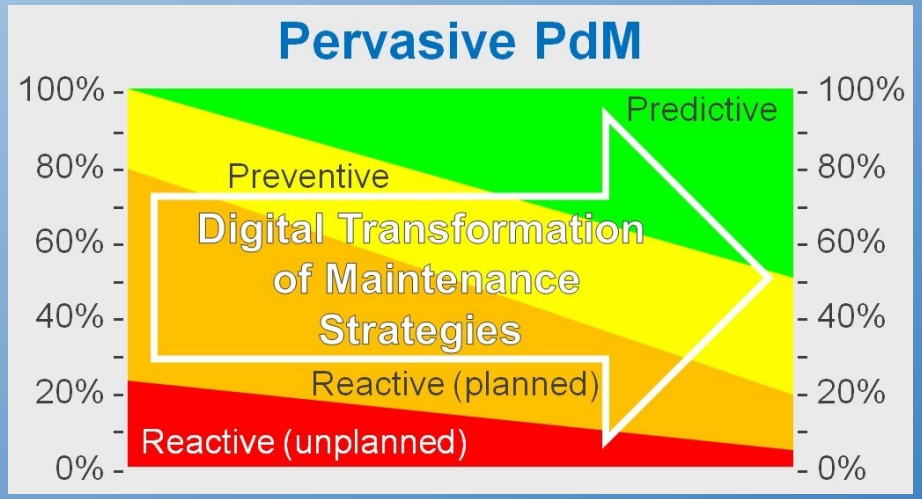


What does Maintenance Involve

- Changes to the software
  - New features, capabilities, uses
  - Fix bugs, scale the system, …
  - Adapting and staying ahead of the competition
  - Attracting and keeping users
  - Making the software self-supporting
- Changes to the environment
  - New versions of the language, libraries, OS
  - New architectures
  - New hardware to interact with
- Changes to the user base
- Changes to personnel



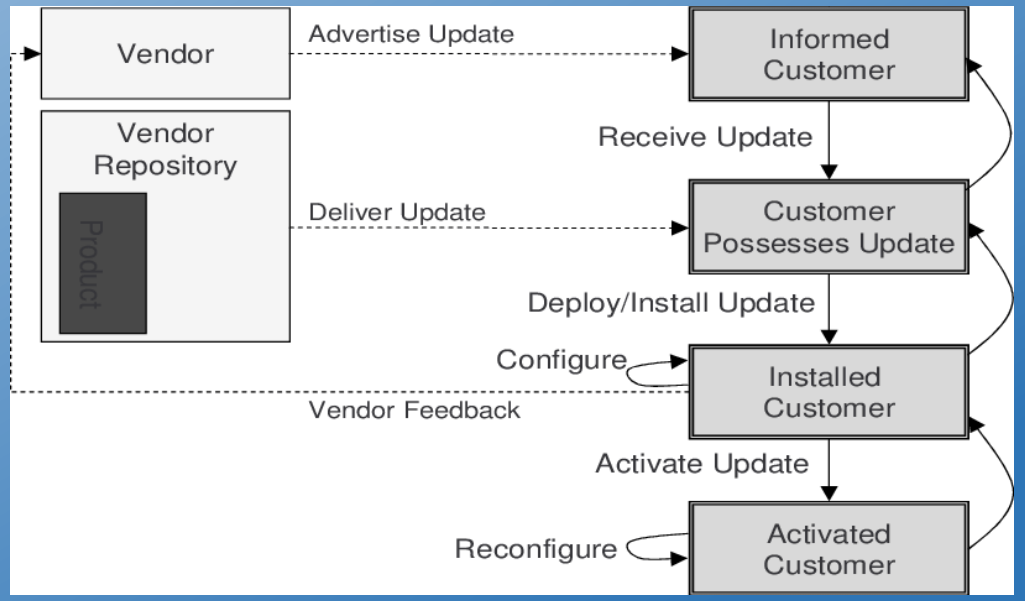
Maintenance is Pervasive



- Throughout software development
  - In requirements, specifications
  - Design for maintenance
  - Coding for maintenance
  - **All software development is maintenance**
- Much of maintenance involves coding and debugging
  - As well as requirements, specifications, and design
  - But we have covered these
- Maintenance also involves some new issues
  - That you and your system must deal with
  - For the user, web-based applications, the backend, & in general

Maintenance Issues: User Facing

- How to have the user install the software
  - App stores
  - Direct downloads
  - Purchases
  - On different systems
  - No need: it's a web page
- How to have the user upgrade the software
  - Automatic upgrades
  - Manual upgrades
  - No need: it's a web page
- How to get the user to understand the software
  - How to use it effectively
  - How to do what they want to do right now
  - Documentation, help, tutorials
- How users interact with the software
  - Privacy and security policies
  - Usage policies

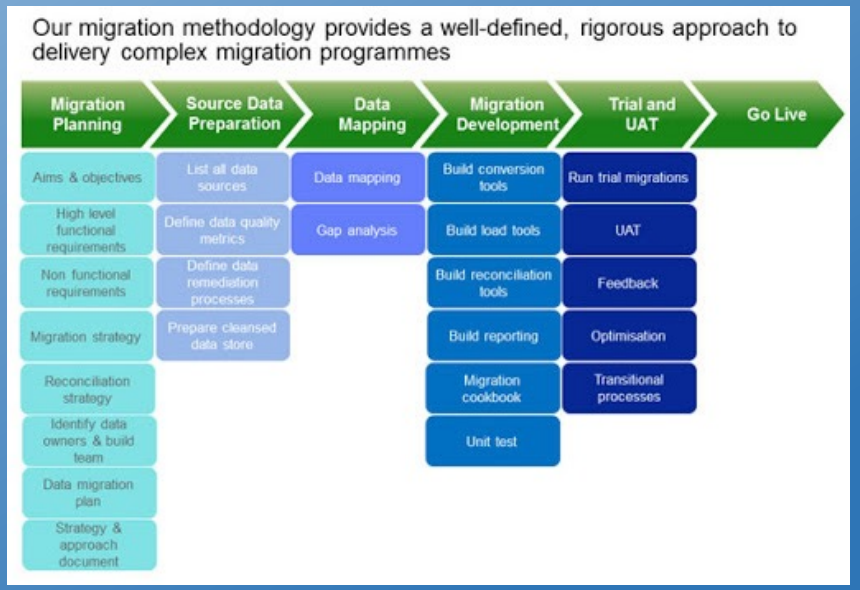


Maintenance Issues: Web-Based

- How will it work with different browsers
  - Older browsers, unusual browsers, new browsers
  - Optional browser features (e.g., location, camera, audio, …)
- How to have it work under different user conditions
  - Lots of different hardware & software options
  - Enable/disable cookies, ad blockers, …
  - Operating system permissions
  - Different networking capabilities
  - Cached pages
- How to have the user understand the software
- How users interact with the software
  - Privacy and security policies
  - Usage policies

Maintenance Issues: Back End

- How to upgrade the server as the software evolves
  - Minimizing down time
  - Without affecting current users
- How to enable porting to a new platform
  - New OS, language, libraries
  - Changing back end
- How to ensure stability of the platform
  - You must upgrade (security, EOL)
  - What happens when things changes
  - What happens as needs changes
  - What happens as resource usage changes
  - Keeping it and the data secure
  - Monitoring what is happening



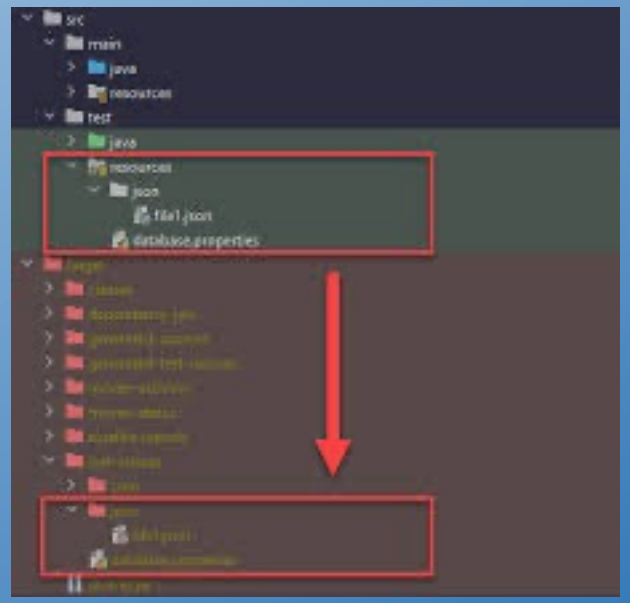
Maintenance Issues: General

- How to bring new developers on board
- How to maintain other programmer's code
- Ensuring you don't break things
  - When maintaining the code
  - Adding new features
  - Adapting to new environments

User-Facing Software Issues

- User environment differs from debugging environment
  - Jar file rather than class directories
    - Binary rather than source
    - Not running in the debugger
  - Binary files need to be architecture specific
  - Different compilers and shared libraries
  - Different versions of JRE, JavaScript, python, Node, npm, …
  - Different shells and system commands
  - Different installation directory and paths
  - Different file system (paths might not work) (/ vs \)
  - Access to resource files can differ
- You want the software to work in the user's environment
  - Running at arbitrary path, with different permissions
  - For arbitrary user environments

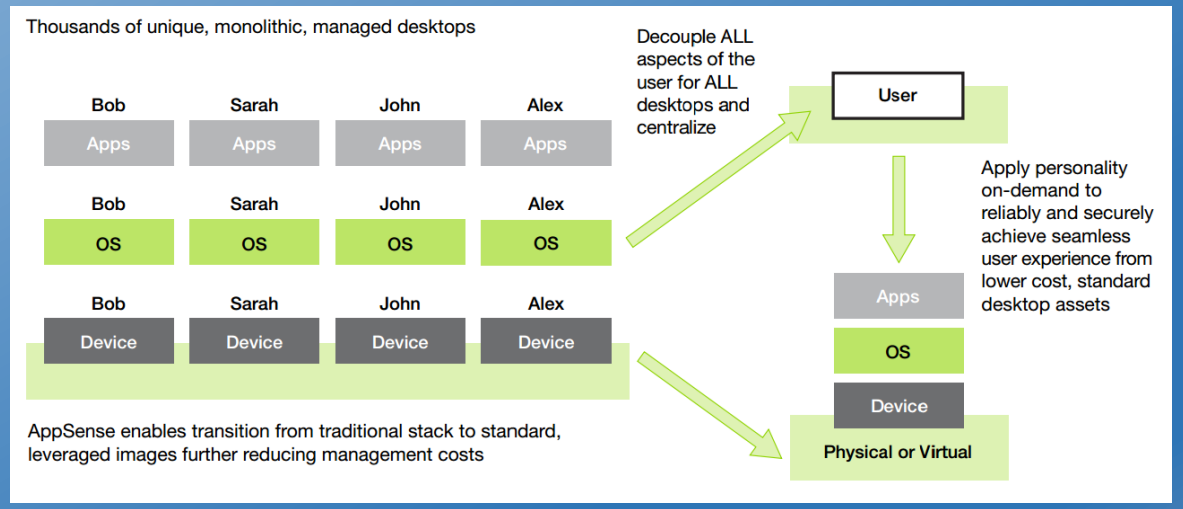
Resources on the User System



- Resource files
  - Images (icons), text (i18n), user-settings, libraries, …
  - Options and user settings
  - Program needs to read these
  - Both while developing and in production
- Accessing resource files in the user's environment
  - Self-discovery (location of binary can be found)
    - If resources are kept in a jar
    - Need to handle resources in development as well
  - Java: Based on class path (using class loader) (works with jar as well)
  - Setting an environment variable (and ensuring user sets it)
    - Updating the user's shell startup files
  - Patching the binary on installation (or compiling from scratch)
  - Part of the system (Mac applications are directories)
  - Fixed location (/System/Library/<application>/…, Windows Registry)
  - As part of installation (android, ios)

Handling the User Environment at Runtime

- Need to run on what the user has
  - Different versions of the OS, language, libraries, tools
  - Different installation directories, permissions, packages
  - Different environment variables, shells, …
  - User might not have sudo or root permission
  - User might not be a sophisticated user
- Can check prior to installation
  - Separate program to check if usable
  - As part of build/initialization (linux configure)
  - As part of app-store
  - As part of preinstallation or installation
  - But user can change this underneath you
- Code for the least common denominator
  - Don't use new features
  - Or use them only if available
  - Or isolate in your own modules

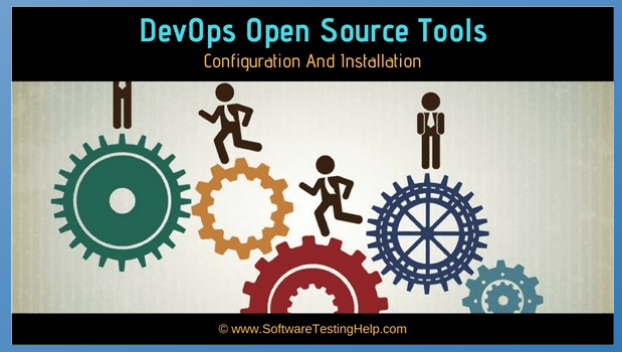


Installation & Upgrading

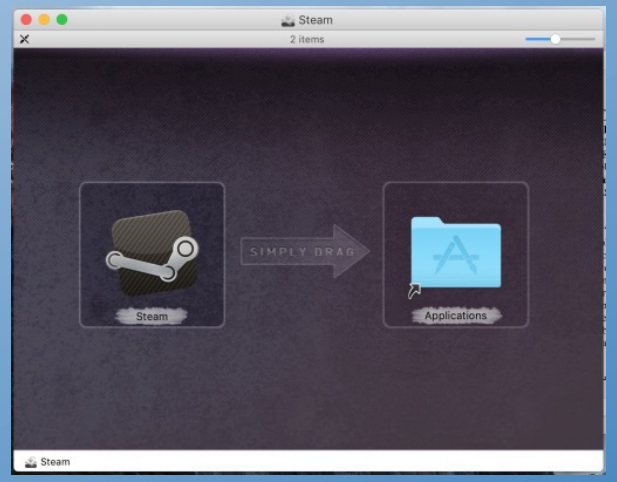
- What do you need to do to facilitate installation?
  - Need to have the user install the software on their machine
  - Need to have the user install the software in their environment
- Potential problems
  - Detecting the environment and adapting to different ones
  - Checking if the environment is suitable
  - Removing inconsistent or older versions of the software
  - Maintaining the user's settings and work from older versions
- Potential solutions
  - Think about all the ways you have installed software

Installation Solutions

- Download a package that includes and runs installation script
  - app-get on Linux (fixed location install; needs sudo)
  - auto-run on CDs
- Download source, configure, compile, install
  - homebrew on the Mac
- Download and run a dedicated installer program
  - Typically, a single disk image or file containing installer and data
    - Can just be installer and do the installation from the network (Eclipse)
  - Commercial installers exists
    - You customize them by writing “scripts”
    - These provide access to common operations
  - Open-source installers exist
    - Not as robust as the commercial ones; scripts are much more limited

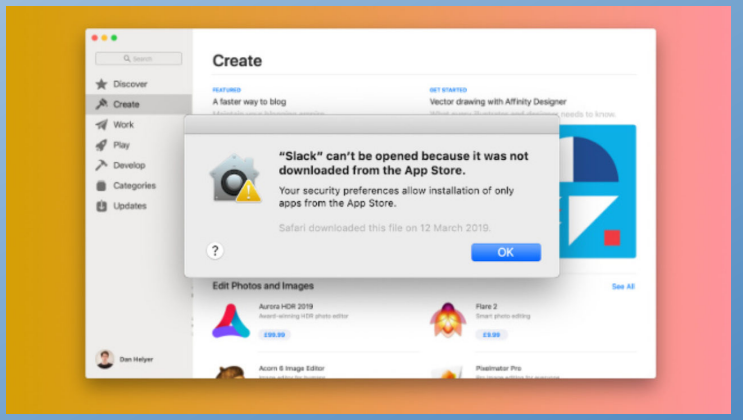


Installation Solutions



- Mac OS/X
  - Application is already bundled in a directory (*.app)
    - User just moves it to the applications folder
    - Directory includes binary(s), libraries, resources
    - Single directory can work with different architectures
  - Disk image contains app and link to applications and hint to move
- Self-Installing
  - Check and install as part of system initialization
    - Can be done in conjunction with any of the above
  - Code bubbles: download jar. First run does the installation
    - Prompts user for any needed information
    - Sets up install directory contents
    - Restarts the application with proper libraries

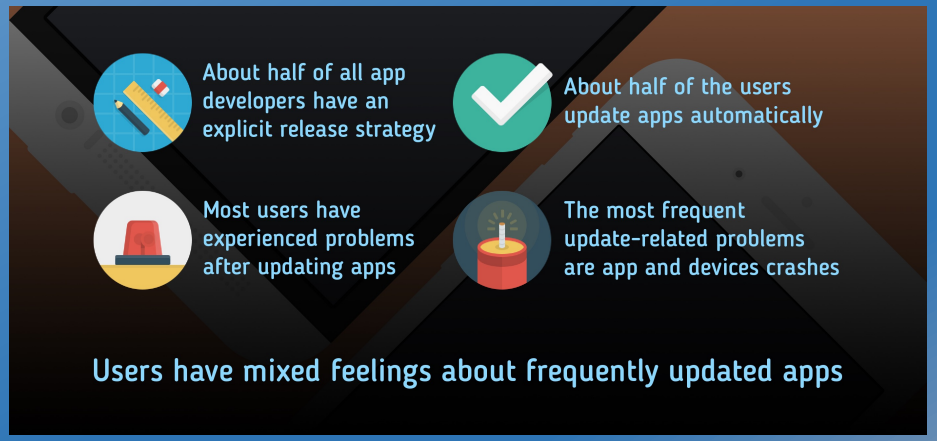
Installation Solutions



- App Stores
  - Provide a standard way of installing mobile apps
    - You must adhere to the protocols, licenses, inspections, etc.
    - Not convenient in early stages of development
    - Include resources, compatibility, capabilities needed, …
  - Being extended to desktops (mac, windows, chrome-OS)
    - But here they typically use normal installers
- Marketplaces
  - Marketplaces provide the equivalent for single apps (e.g., Eclipse, Code)
  - Useful where 3rd-party plug-ins are a way of extension
  - Require considerable work to build, maintain on your own

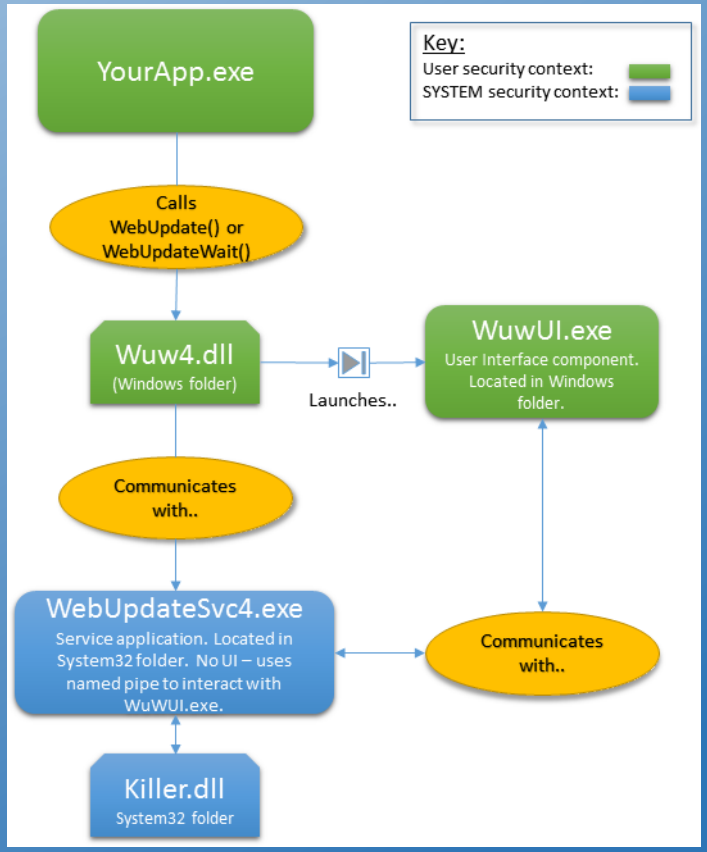
The Updating Problem

- Software updates are going to be required
  - Fixing bugs, adding new features, accommodating new OSs, etc.
  - What do you require from the user
- Detecting when updates are available
  - Knowing when to update, integrating this into the application
  - Not asking the user all the time
  - Doing it automatically
  - Detecting if update is appropriate



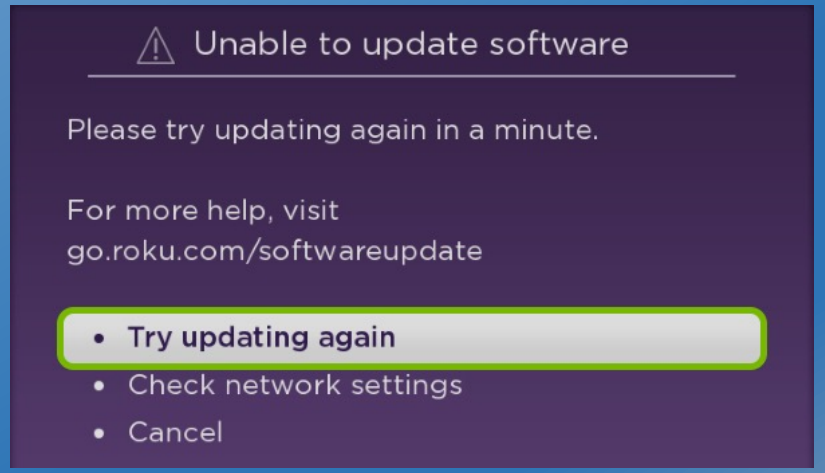
How to Update

- Separate program that does the update
  - Save state that needs to be saved
  - Update the software (and saved state)
  - Remove previous version
  - Restart the software
  - Need to ensure the update can be aborted
    - Download; **atomic** replace; restart
    - Separate file partitions used in IoT devices
- Downloading and running the installer again
  - Rather than a separate program
    - But this is visible to user, not hidden
  - Using user's current settings
    - Requires a smarter installer
  - Installer probably needs to do this anyway



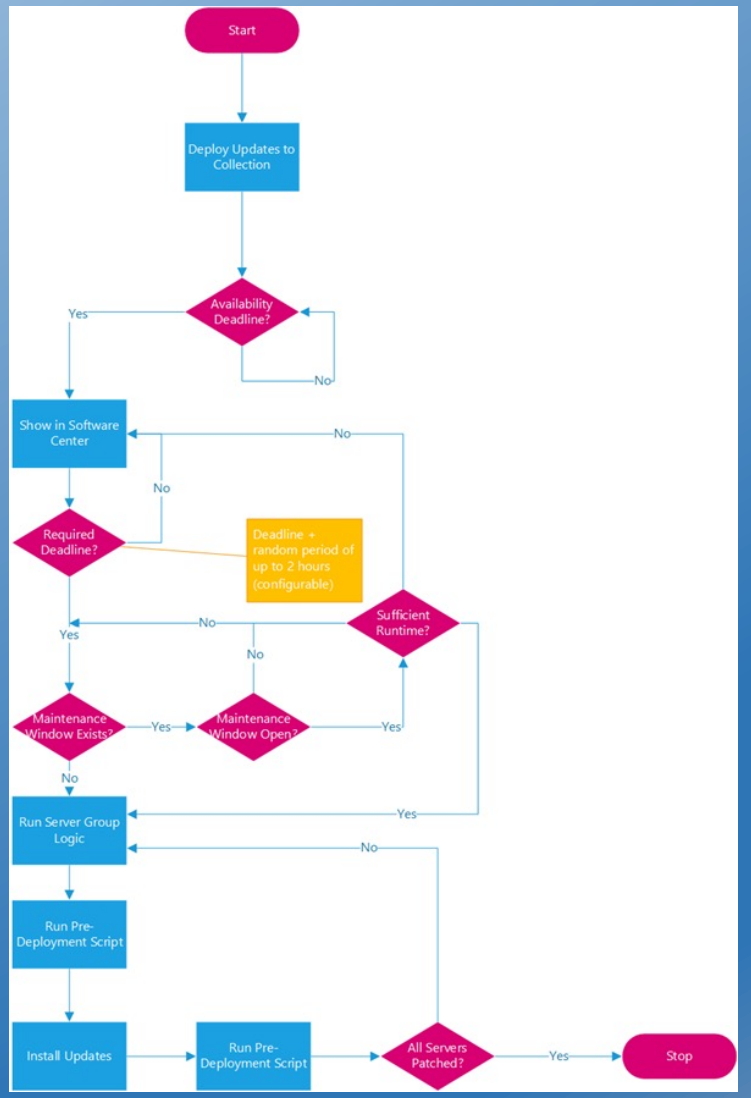
Update Issues to Consider

- Matching the installation with the update
  - Anything the user had to set up before
- Updating personal resource files if they have changed
- Permissions required
- Checking for other required libraries, packages, systems, …
- Updating libraries, resources, registry
- Removing outdated resources
- Handling errors
  - Permissions, disk space, network issues, …
  - Bad updates



Back-End Systems

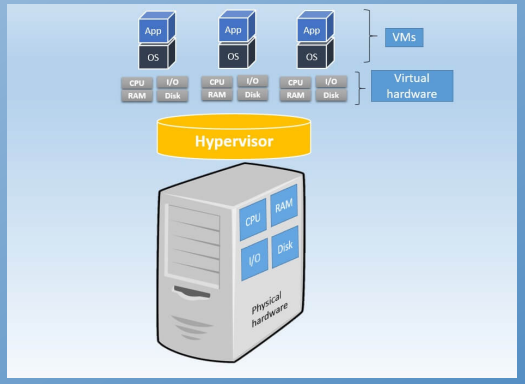
- Back-end systems are a bit easier to maintain
  - You have control of the environment
  - You have control of when and how to update
- Problems still exist
  - Isolating users from potential changes
  - Isolating from other applications and their needs
  - Handling hardware/software errors
    - Don’t crash or do an auto restart
  - Updating
    - With little or no downtime
    - Without losing user connections/sessions
      - Keep these in permanent storage
    - Running multiple versions simultaneously



EXERCISE

- What are your projects installation and update plans?
  - Have you investigated this at all?
  - Have you built it into your system?
- For web applications, what are your compatibility plans?

Virtual Machines



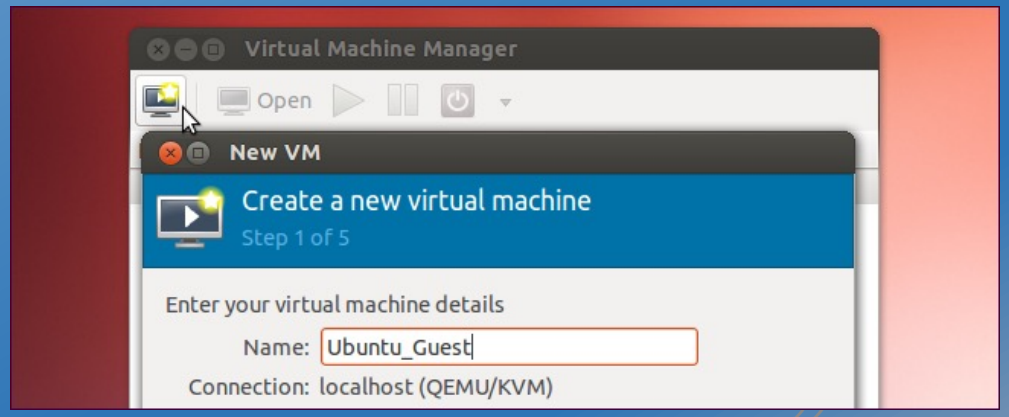
- Today most server code is run on virtual machines
  - Some of you are already doing this
  - You can't run in production from your local machine
- Cheaper than owning specific hardware for each application
  - Your application seems to have its own dedicated hardware
  - But the physical hardware can be shared
  - Someone else worries about hardware maintenance, etc.
  - Can specialize the VM to the application
  - Easy to migrate to a new VM
    - If the hardware goes down
    - As your requirements change
- You have control of the environment
  - Libraries, installed software, versions, etc.
  - You control when things change

Virtual Machines

- Easier to upgrade as needed
  - To a larger server
  - To multiple servers
  - Can migrate from one VM to another
    - Save and restore state
    - Recovery
  - Can just change IP address to point to a new VM to upgrade
    - But then everything needs to be saved in a separate database
- Virtual Machines provide a layer of security
  - Isolate the software to a particular VM
  - Compromise the VM, not the development platform
    - Setting up a new VM should be relatively easy
  - Can migrate on hardware failures

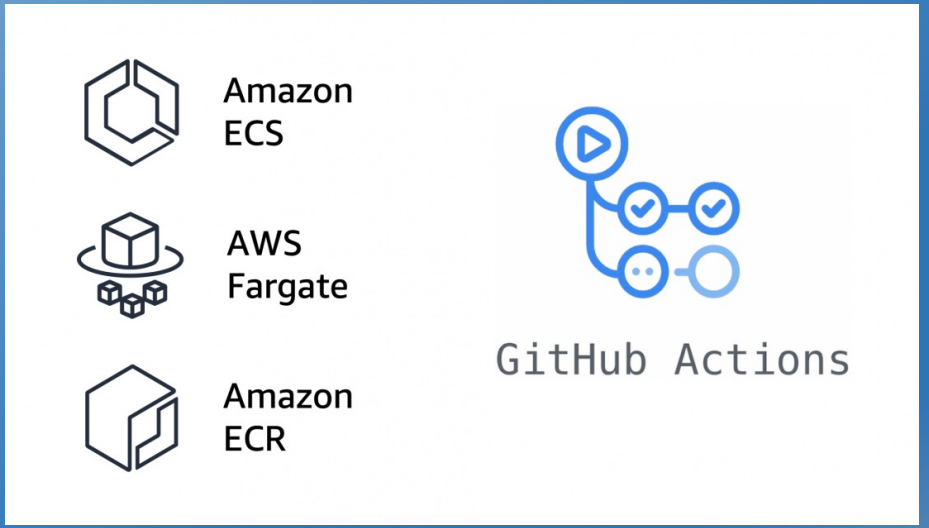
Most VMs run Linux

- Standard configurations for most standard servers
  - For most execution stacks
  - Apache, MySQL, Php; Node, Mongo; Wordpress; dart/flutter; flask …
  - Or instructions for setting up such a server for your stack
- Different ecosystems provide different options
  - Amazon (AWS): servers, disk space, SQL & NoSQL databases, firewall, credentials, …
  - Brown CS: servers, disk space, PostgreSQL, firewall (staff maintained)
  - Google cloud, Microsoft Azure, IBM …
- You should learn how to manage the system
  - Install and upgrade software (apt-get)
  - Manage the database system
  - Manage credentials, users, …
  - Manage the firewall



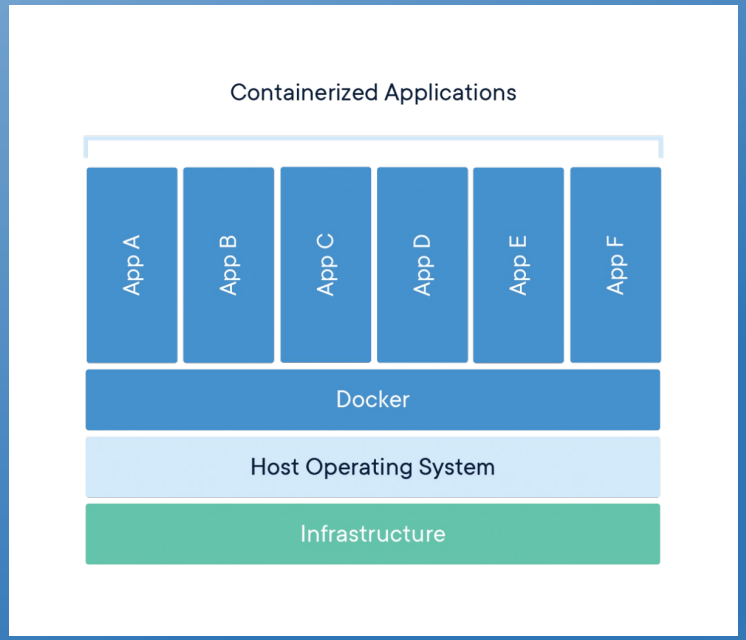
GIT and the VM Server

- GIT can be useful for maintaining the virtual machine
  - Ideally you have a development version of the software
  - Can have a production branch as well
- Installing on the server should be as easy as
  - git pull (to get the proper version)
  - Run installation or update script
    - Set up anything that needs setting up
    - Stop old server
    - Start new server (pm2)
  - Can be automatic with git/GitHub actions
    - Actions also available from cloud service
  - Update script for database
    - If the database schema has changed

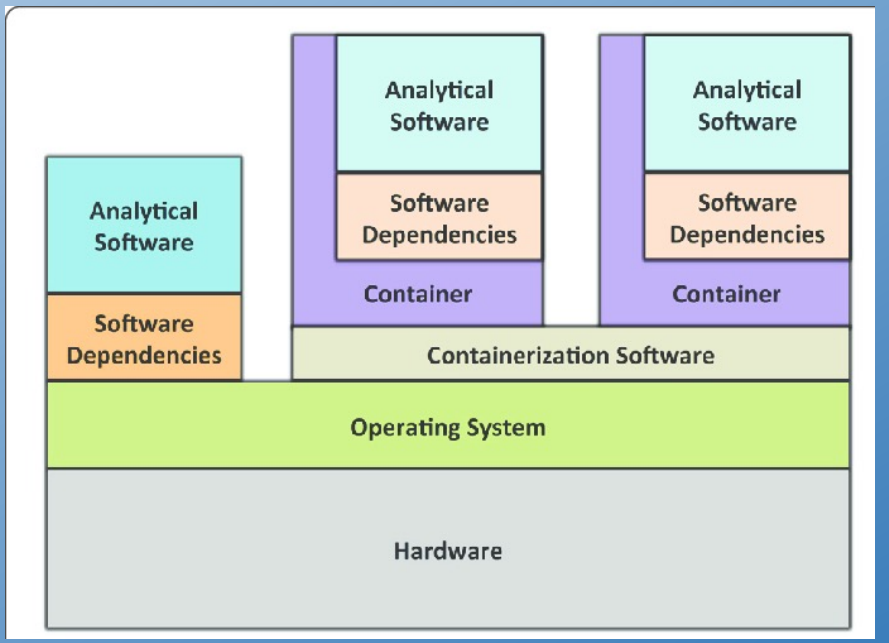


Virtual Machine Issues

- Virtual machines have considerable overhead
  - Pre-allocate memory, disk, …
  - Startup, even from sleep, takes time
  - Extra runtime overhead of a hypervisor
  - Migration can take time (copying large files)
  - Out of your control
- But they provide a high degree of isolation
  - One VM is unlikely to affect another
  - Corrupting a VM doesn't corrupt everything
  - And they offer a lot of flexability
- Would like the benefits without the costs



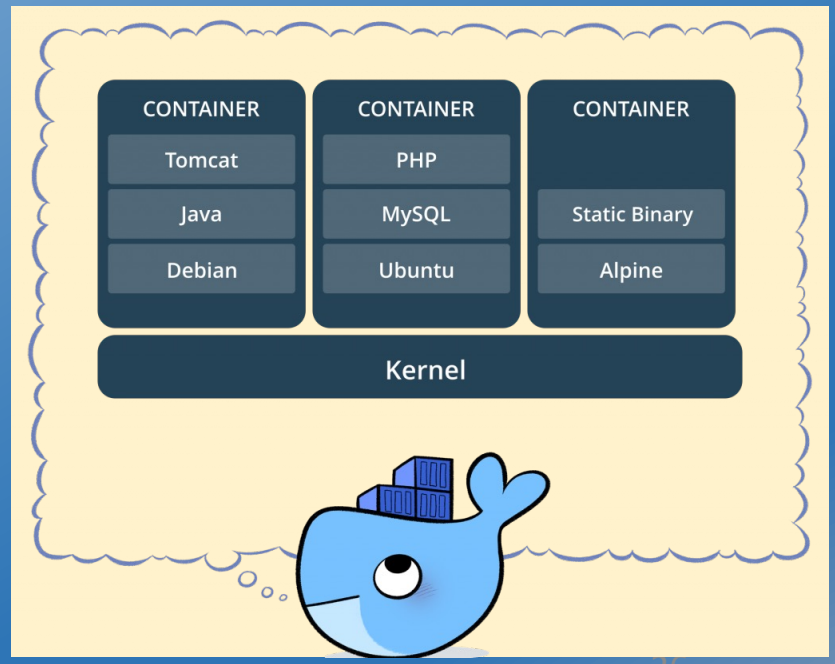
Containers



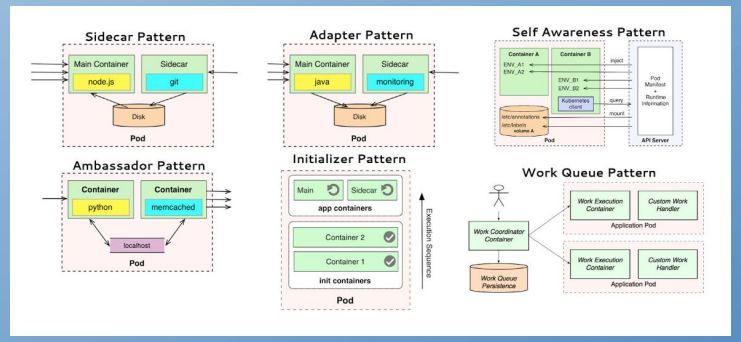
- Containers are a lightweight solution
  - Equivalent functionality to VMs
  - But with much less overhead
- Operating-system level virtualization
  - Multiple isolated systems running using a single kernel
  - Isolation
    - Independent process trees
    - Independent networking
    - Independent user ids
    - Independent file systems
    - Container can crash or be corrupted without corrupting the system

Container Pros and Cons

- Isolation of file system, processes, network, users
- Rollback
  - Save and restore images, restart from image
  - Fast to start a new container from image
- Rapid deployment
- Migrate from one container to another
- Save and restore state
- Difficult to use with user facing apps
  - Need access to the shared display
- Depend on host for networking



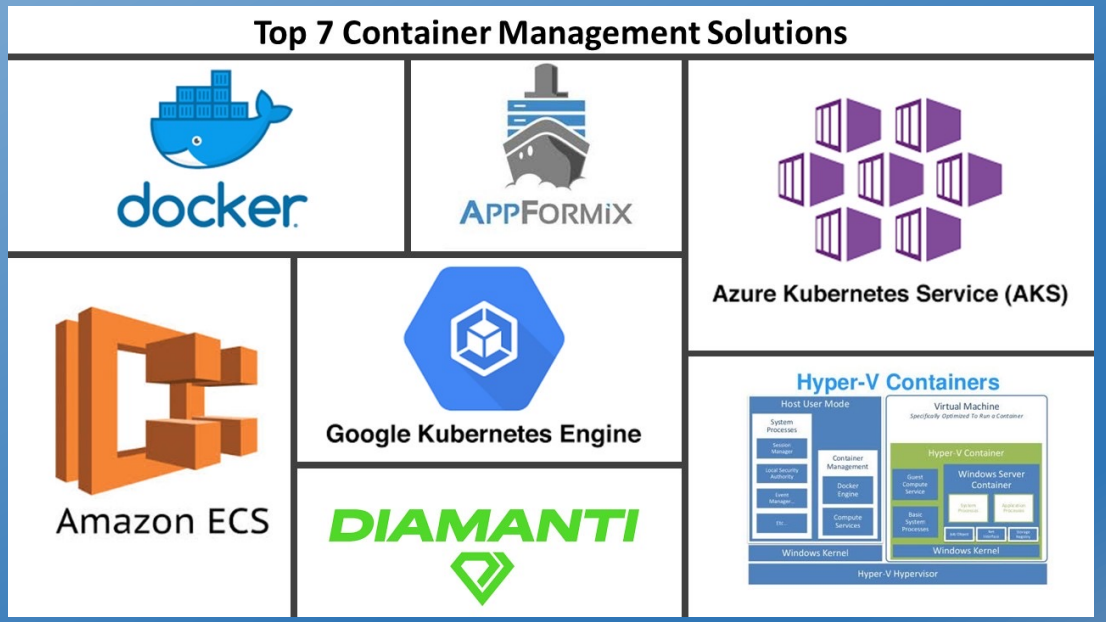
Containers and Microservices



- Microservice architecture
  - Divide the system in small, composable pieces
  - Services can invoke other services
  - Front end (web/mobile?) can invoke services as needed
  - Services can be updated or replaced independently
  - Good fit for some applications
- Containers are a good match for microservices
  - Each microservice can run in its own container
  - Database can run in its own container or VM

Managing Containers

- Managing a lot of containers can get tricky
- DOCKER provides facilities for this
  - Based on kubernates



HOMEWORK

- Sign up at DOCKER if you don’t already use it
- Download and install DOCKER desktop
- Do the tutorial that is there
- No hand in

Privacy and Security Policies

- Privacy and security were considered since requirements
  - Pretty much in the abstract
  - Once you have a user-facing system these become concrete
- You need a privacy policy
  - What data you are storing and why
  - What permissions users retain on their data
  - How to meet various legal requirements
    - California and GDR (Europe) have specific requirements
  - Might want to verify compliance with your policy
- You need a security policy
  - How to handle personally identifiable information
  - Your responsibility for that information
  - Keep track of what you are doing – security czar

Usage Policies and other Legal Issues

- You probably want a usage policy
  - Can someone scrape your site
    - To grab your data or functionality
  - Can someone scrape your site for search
  - Can someone scrape your site for building an AI model
  - How much can one person use the site
    - Can your site be accessed from a bot or application or must it be a real user
- Other legal issues
  - HIPAA (health), FERPA (students), COPPA (children)
  - ADA compliance
  - IP for code and data (copyrights, trade secrets, patents)
  - Liability for accuracy of information presented

PROJECT

- Presentation Schedule
  - 11/26: User Interface Generation
  - 12/03: Speech
  - 12/03: Sense IoT
  - 12/05: LLAMA
  - 12/05: DJ Mix
  - 12/10: Agentic
  - 12/10: Accessibility
  - 12/12: Final demos for all groups (optional)