Design Types

Plan-and-Document

- Very detailed plan
- Update documentation before changing code

Waterfall approach

- Steps:
  1. Requirements analysis and specification
  2. Architectural design
  3. Implementation and integration
  4. Verification
  5. Operations and maintenance
- Month long phases
- Brook's Law: adding manpower to a late project can make it later
  - $O(N^2)$ vs $O(N)$ b/c coordination

Agile

- Peres' Law: Software change is a fact of life, so don't try to solve it, but work with it
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan
- Continuous improvement
  - Iterative design (~1-2 weeks)
- Use user stories over detailed requirements elicitation
- Not always the best

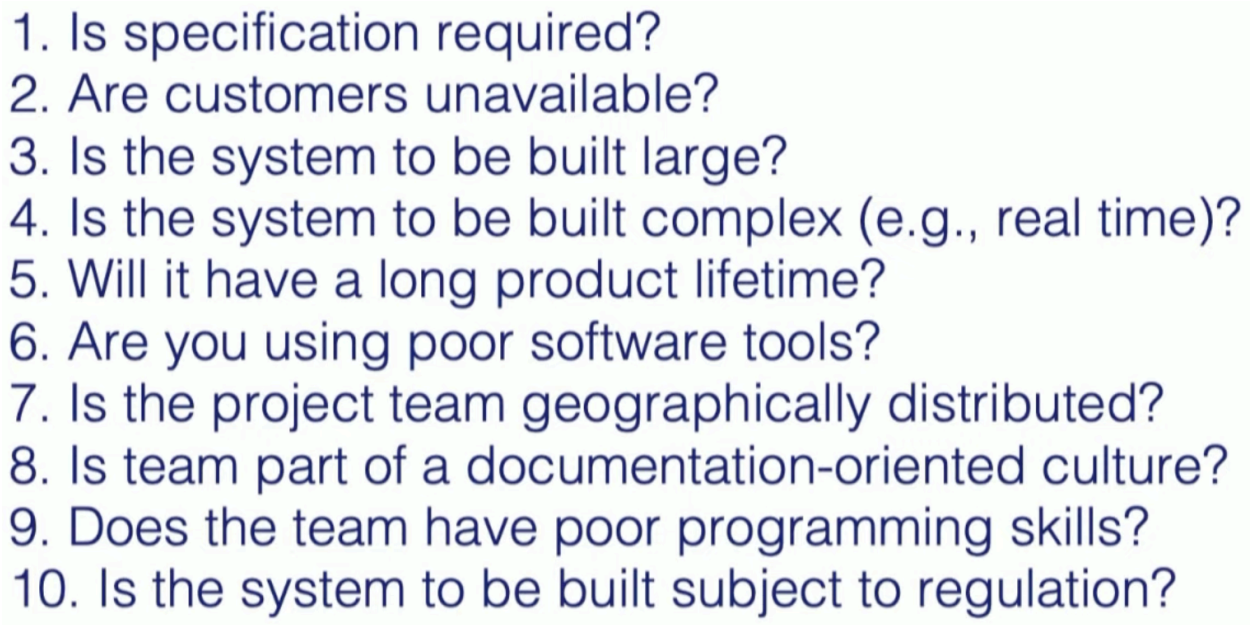

Figure 1: Screenshot_2023-08-24_at_8.49.25_PM.png

–

Extreme Programming

- Take agile to the extreme
- Make iterations as short as possible
- Make things as simple as possible
- Test all of the time (test driven development: write tests before code)
- Review code continuously
  - GitHub PR

Testing and Quality

- Product quality / fitness for use: business value for customer and manufacturer
- Software quality: Satisfies customer needs and is easy for developer to debug and enhance
- Software QA: ensure quality and improve processes in SW organization

Verification and Validation

- Verification: Did you build the thing right?
  - Did you meet the spec?
- Validation: Did you build the right thing?
  - Is it what the customer wants?
  - Was the spec correct
  - More important for SW

Testing

- Test in levels
- Coverage: % of system exercises by test suite
- Unit test: Single method does what was expected
- Module / functional test: Test across individual units
- Integration test: Interfaces between units have consistent assumptions, communicate correctly
- System / acceptance test: Integrated program meets its specifications

Productivity

- Moore's law applied to SW complexity

1. Clarity via conciseness
2. Synthesis: have code that writes code
3. Reuse: reuse stuff when possible
4. Automation and Tools

Clarity via Conciseness

- Raise the level of abstraction
  - More functionality, fewer words
- Exploit high level languages for conciseness

Synthesis

- Common uses are templating and code generators

Reuse (DRY it out)

- Identify commonalities and abstract it

1. Procedures and functions
2. Standardized libraries
3. Object oriented programming: reuse and manage collections of tasks
4. Design patterns: reuse a general strategy even if implementation varies
5. LLMs and prompt reuse

Automation vs Manual Task

- Saves time and improves accuracy
  - eg: make
- New tool desiderata: learning curve justifies productivity gained
- Learn new tools!

Deploying SaaS SOA and Utility Computing

SaaS > SWS (shrink-wrapped software)

- No install / upgrade / compatibility issues
- Data stored safely, persistently on servers
- Easy group access to data for collaboration
- Simpler to keep 1 copy rather than multiple
- Upgrades for all users immediately
- Can beta test on subset of users

Service Oriented Architecture

- Subsystems independent as if in separate datacenters
- Can recombine to make new services
- Must be interacted with through individual API

3 Demands

- Communication
- Scalability
- Dependability
- Done with warehouse-scale computers
  - Plugability and dependency

Deploying SaaS: Browser and Mobile

- ~50% smartphone penetration
  - Apps should be either mobile-friendly or mobile-first
- Should have responsible deign
- Think about interactions (how someone uses it physically)

Web Techs

- HTML: Markup only
- CSS: Styling only
  - Bootstrap: mobile-first framework (integrated into rails)
- Progressive web app (PWA): packages app to be downloaded or function offline

  - Can do progressive updates which Wrapped, and Native cant
- Wrapped apps: apps in containers, no progressive updates
- Native apps: built for a specific platform (better access to HW)