SDLC

2019-05-06 22:55
1a0f14a

Contents

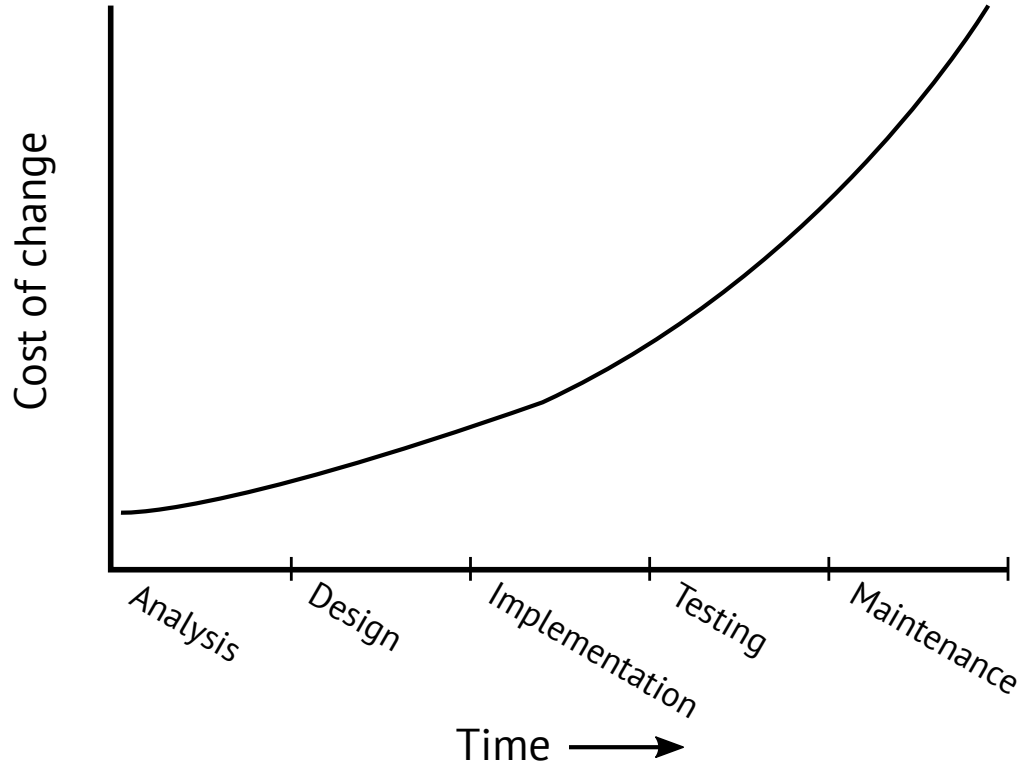
Software Development Lifecycle

1. Analysis = Gathering and document **requirements**
2. Design = System architecture: components/modules
3. Implementation = Code the modules
4. Testing = Verification that they meet the specification
5. Maintenance = Any evolution after initial deployment

Waterfall process for software dev

- Do each phase in order
- Each completes before starting next phase
- Outputs of one phase are inputs to next phase

Cost of change over time



Incremental and iterative processes

- Spiral model [B.Boehm]
- Agile methods [K Beck, W Cunningham, …]

A few agile techniques

Pair programming

- 2 devs, 1 keyboard, writing code together
- The best debugging is never "embugging" in the 1st place

Test-driven development

- Write test **first**, then write code to pass the test.

"Sprints" = defines what the increment is

- "Product owner" decides what features go into the next iteration.
- We have a defined timeline: ~3 weeks.

Analysis

Gathering and documenting requirements.

- Identify the **stakeholders**.
  - Users of system. Also there can be different categories of users:
  - Administrative users, end users, power users
  - Investors/owners
  - Dev staff, incl maintenance/operations
- Functional requirements
  - What the system has to DO
  - "Functionality"
- vs Non-functional requirements
  - Constraints on the development/operation of system
  - aka the "-ilities"
  - Reliability
  - Availability ("uptime")
  - Usability = "user-friendly"
  - Consistency
  - Applicability
  - Efficiency
  - Utility
  - Readability
  - Maintainability
  - Vulnerability (security)
  - Scalability = How much data/ how many users can it support?

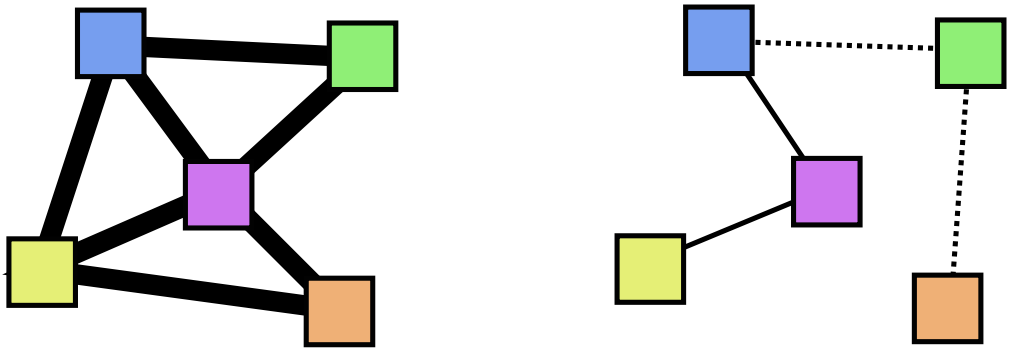
Want requirements to be **specific, realistic, testable.**

Part of being specific is that they should be **quantified** (especially the ilities). Here are some examples of quantification

- Availability: 99% vs 99.9% vs 99.99% "nines"
- Usability: what kind(s) of users, how long should it take them to become proficient?
- Efficiency: eg, process N GB data in M seconds

Design

- Modularity
- Separation of concerns
- Strong coupling (bad) vs Loose/Weak coupling (good)



Verification / validation

- Verification = conformance to a specification. Does it meet the **spec?**
- Validation = Does it meet the **need?**
- Verification
  - Dynamic verification (aka Testing) = **Run** the system to see what it does.
  - Static verification (aka Inspection). Code reviews.
  - Automation

Misc

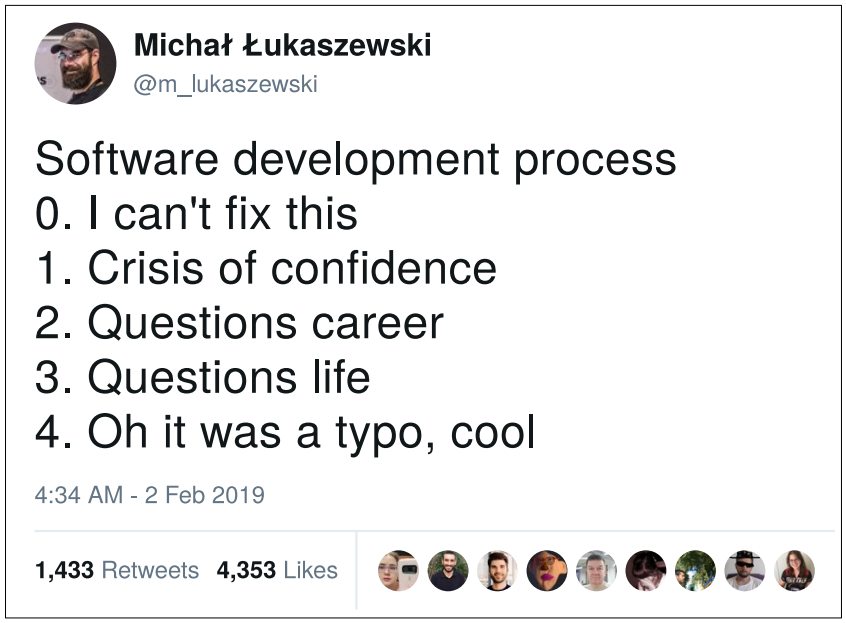


Figure 1: `@m_lukaszewski` on Twitter