How to Manage Technical Debt

By : Jason Ebueng, K. Ngo, K. Nguyen, T. Yau

Technical Disciplines and Practices to Manage Technical Debt

- Continuous Integration
- Collective Ownership
- Test Driven Development
- Pair Programming
- Refactoring
- Incentives

Continuous Integration

- Problem: Late integration of software often run into misalignment problems. The later the discovery, the more difficult and expensive it will be to fix. The more infrequent integration happens, the higher the risk of major development issues.
- Solution: Continuous Integration
  - Objective: frequent integration of software and hardware to reduce cost of misalignment late in development efforts
- Benefits of continuous integration
  - Better schedule and cost
    - Integration testing can start sooner
    - Misalignment problems can be resolved earlier
  - Better communication and collaboration between teams

Continuous Integration (continued)

- Suggested implementation
  - Set up code base on a build server
  - Encourage frequent commits
    - Record number of commits
      - Number of commits should be larger than number of developers on the team
  - Build frequently
    - Ie. every time developer commits code, periodically, etc
  - Run all tests on build
  - Address failures immediately
  - Measurement
    - How often build fails
    - How often failures go unaddressed
    - Reward low failure rate

Collective Ownership

- What it is
  - Collective ownership encourages all development team members to contribute to all segments of the project
  - Any developer can contribute by
    - Fixing bugs
    - Adding new features
    - Refactor code
- Benefits
  - Addresses the issue of team members leaving the company
  - No bottleneck for code changes
  - Improved collaboration
  - Improved sharing of information

Test Driven Development (TDD)

- TDD is a software development process with 3 main repeated steps
  - 1) Write an automated unit test
  - 2) Write enough codes to pass
  - 3) Refactor - Improve code quality

- The main idea of TDD is to improve and make codes with high quality by repeating the process in a series of short development cycles.

Test Driven Development (continued)

- Developers can measure their progress by documenting it through a series of written unit tests

- Because TDD can be measured,
  - It cleans up issues that may come about due to SCRUM
  - Refactoring is done without messing up the codes

Pair Programming

- Pair programming is a technique for writing quality codes
  - It involves two developers (hence the word pair)
  - The two developers work side by side on a one computer to write codes

- There are two roles for the two developers working together
  - The driver writes the codes
  - The navigator reviews the code

Measuring Software Quality

- Measure test coverage, measure number of defects reported per sprint to determine if there is sufficient test coverage
- Measure number of tests that need to be executed, and try to minimize that number especially if they are manual tests; goal = develop confidence in test suite so that product can be deployed based on pass/fail of the test suite
- Measure size of test code, total amount of test code should be about the same as production code

Measuring Software Quality (continued)

- Measure test velocity, tests should run in minutes rather than hours
- Measure test breakage, design tests so that tests are minimally impacted by changes in production code
- Measure Cyclomatic/Code complexity, refactor functions and other code bits that are too complex
- Measure function and class size

Measuring Software Quality (continued)

- Measure dependency metrics
- Utilize static analysis tool identify code weaknesses and flaws, can be used to identify duplicate code
- Measure number of commits each day, encourage frequent code commits
- Measure amount of time continuous build fails
- Measure done-ness by executing story tests in the CI/CD flow and measuring number of story tests that pass/fail

Offer incentives

- Objective: empower teams and improve morale by offering incentives for good practices
- Reward both quality and speed
  - Context: Scrum teams develop quickly. Therefore technical debt may grow quickly.
  - Reason
    - Rewarding only speed will quickly increase technical debt
    - Rewarding both equally will give incentive to develop quickly and with quality
- Record measurements
- Make results and metrics transparent and visible
  - Use a big visible screen to post metrics
- Celebrate milestones
  - Team parties
  - Give out trophies
  - Cash reward

Refactoring

- Refactoring is the process of refining and improving existing code without changing its behavior
- This improvement process helps manage code in Technical Debt as it ages and becomes more complex
- Benefits include
  - Improved code length and simplicity
  - Improved understanding
  - Produces re-usable code and modules
  - Less Bugs
- Test Driven Development promotes refactoring

Refactoring

- XP style programming promotes refactoring “whenever and wherever possible”
- With Test Driven Development, tests can be used against expected behavior as code is refined
- IDEs and modern tools have automated suites that can test behavior constantly and consistently

Refactoring

- A series of small changes and updates for incremental improvements
  - This manages code behavior during refinement
- Behavior should remain consistent, so new functionality isn't the goal
- Refactoring helps promote team cohesion and collective code ownership
  - Constant audit and improvement with the whole team promotes a unified understanding about the architecture
- Refactor still requires cost and time
  - Stakeholders must consider and accept these resource requirements