Second Team-Based Evaluation

Eyoel Gizaw, Nik Nandi, Pruthak Patel, Saron Tesemma, Dahir Yonis

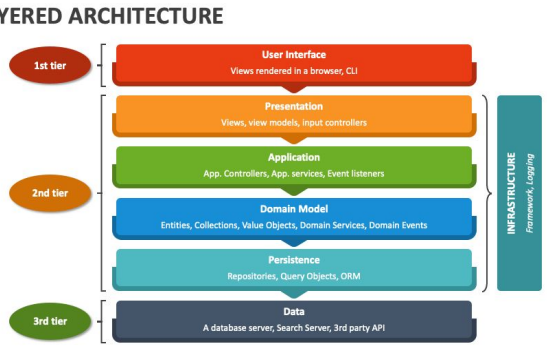
Georgia Tech

1 - Misuse of Story Points in Agile

- **Context**
  - Successful corporation shift to Agile
  - Story point hard conversion to 5 hours has led to developer frustration
- **Problem**
  - Story points do NOT equate to time worked, rather complexity
  - Fixed conversions take away from flexibility of Agile
  - Estimations are inflated -> Morale drops -> Quality suffers
- **Suggested Approaches**
  - Measure velocity as opposed to hourly output
  - Prevent burnout via Work In Progress limits
  - Emphasis on value delivered instead of numbers
  - Honest retrospectives

2 - Tor's Architectural Style

- **Context**
  - Tor Browser anonymizes web traffic via nodes & multiple relays
  - Masks IPs, browser activity and is used by journalists, whistleblowers, etc
- **Layered Architecture**
  - Tor traffic passes through entry, middle, and exit nodes—different layers
  - Layers only know (and need to know) next node
  - Separation of concern enhanced security
- **Effectivity**
  - Prevents traffic traceability
  - Layers act independently to reduce attack surface
  - Mirrors Layered Design Pattern from class



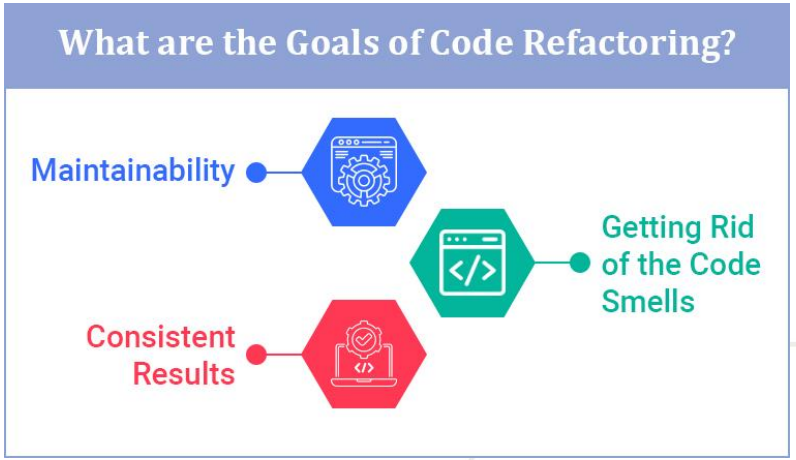
Georgia Tech

3 - Composition Over Inheritance

- **Context**
  - Traditional OOP favors inheritance (rigid hierarchies)
  - Rust & Go favor composition with traits/interfaces
- **Inheritance Issues**
  - Deep hierarchies are hard to understand and maintain
  - Violates single responsibility & open-closed principles
  - Fragile Base Class problem: Edits to base class break derived classes
- **Composition Benefits**
  - Encourages high cohesion and loose coupling
  - Easy testing and refactoring
  - Promotes reusability

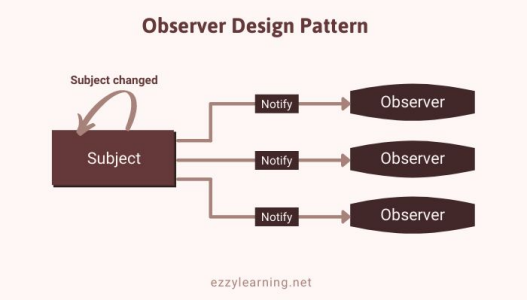
4 - Technical Debt & Refactoring

- **Context**
  - Codebase has high coupling and low cohesion from quick feature ships
- **If We Don’t Refactor:**
  - Bugs become significantly harder to isolate
  - Productivity drops as new features break old ones
  - No one wants to fix extremely messy code
- **If We Do Refactor:**
  - Modularity is improved
  - Faster and safer iteration
  - Easier testing
  - Boosts long-term velocity of the project!



5 - Centralized Logging, Observer Pattern

- **Context**
  - Proposed centralized logging service where each microservice publishes logs to single logging service
- **Best Design Pattern Here: Observer**
  - 1 subject notifies multiple observers (logging service to log subscribers)
  - Each microservice is an observer, logs published to central subject
  - Good for enabling real-time monitoring without tight coupling
- **Other Cases For Observer**
  - Stock price dashboards
    - UI components observe stocks (the subject) update as prices change
  - Chat applications (Discord, Slack, etc)
    - Users observe channels (the subject) have new messages update



Georgia Tech

6 - Modular Payment Handling, Strategy Pattern

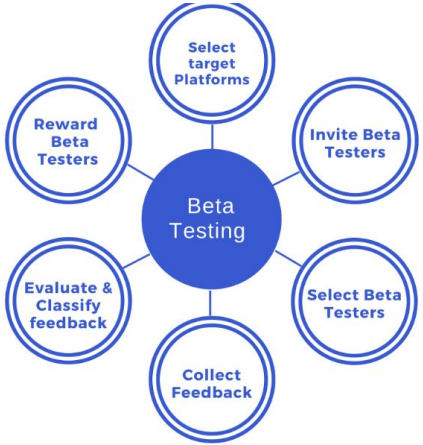
- **Context**
  - Fragile financial processing system because of unique pipelines
  - Want to refactor through building generic handler models
- **Best Design Pattern Here: Strategy**
  - Strategy defines algorithm families and encapsulates each
  - Different payment methods (i.e credit, PayPal) become strategies
  - Payment module uses common interface to choose correct one at runtime
  - Avoids duplicated code, supports open/closed principle
- **Other Cases For Strategy**
  - Sorting algorithms in Excel
    - User picks between ascending, descending and custom - different strategies
  - Game character movement
    - Strategy changes on different states(i.e aggressive, defensive, idle)

7 - Discord, Scalable Architecture for Real-Time Chats

- **Context**
  - Discord prominently uses **microservices architecture**
- **Microservices Architecture**
  - Different functions (i.e messaging, voice chat) are separate and independently deployable services
  - Services communicate via APIs and message queues
  - Ensures rapid development from separate teams
- **Why?**
  - Discord serves multiple niches (large servers, voice chat, direct messaging, threads, group chats, etc) that could develop at different rates
  - Fault isolation and reducing bottlenecks in development is key for innovation. Microservices accomplishes that

8 - Early Constant Testing for QC

- **Context**
  - Developing anonymous feedback platform. Needs thorough testing
- **Developer Testing**
  - Performed by engineers when features are built
  - Catches logic bugs, syntax, broken components early
- **Alpha Testing**
  - Internal team & stakeholders use feature in near real situations
  - Focuses on usability, edge cases, design flaws, and feedback before users
- **Beta Testing**
  - Limited user release
  - Shows performance issues, UX gaps, device compatibility
  - Ensures readiness for full deployment



Thank You!

Georgia Tech