Final_Review_Doc

- Matching terms
- 277 terms
- Another review doc

Quiz 1

- First and 2nd questions in all engineering projects?
    - What is the problem?
    - Who is the customer?

- 3 variables in all software projects?
    - Time
    - Money (Resources)
    - Functionality (Scope)
    - Quality

- One constant in software engineering
    - **Change**

- 3 unwanted items in all engineering projects?
    - Surprise
    - Heroes
    - Miracles

- 2 main software development methodologies
    - Agile
    - Waterfall

- In the age of interruption, how do you stay focused
    - OHIO - only handle it once

- Interaction should be **beneficial**
- What to consider when starting an interaction
    - **What is the goal**

Team

- Project manager
    - Responsible for the project
    - Always available and aware of all aspects
    - Earns respect of the team
    - Allocates resources of the team
    - The go-to guy/girl

- Senior system analyst
    - Coordinator of use cases, user story and requirements

- Database specialist

    - Maintains database and related issues

- Software architect
    - Coordinates design, select technologies

- Software development lead
    - Coordinates software development

- Business analyst
    - Researches student diaspora and applies business rules integrated into project

- QA lead
    - Coordinates testing phase, ensures procedures followed

- UI specialist
    - Focuses on the look and feel and user experience

- Algorithm specialist
    - Designer of algorithms and module interfaces

- Te**am meeting should have**
    - **Agenda and record of the meeting**
    - **Allow team members to speak**
    - **Acknowledge accomplishments**

    - Led be PM or deputy
    - Make it productive use of time

- Team discussions
    - **Recognize the time for input**
    - **Recognize the time when decision has been made**
    - **Sideline unresolved issues**

- Team member good qualities
    - **Supportive**
    - **Receptive**
    - **Offer help**
    - **Accept tasks**

- Trust and betrayal
    - Trust is earned
    - Betrayal is a violation of trust

- Leadership qualities good and bad
    - **Honest** **- did not follow through**
    - **Responsive**
    - **Responsible**
    - **Patient**
    - **Parental**
-

- In terms of **criticism**
    - Welcome it
    - Be tactful and respectful
    - Do not be defensive or offensive
    - Goal: **be corrective yet motivating**

Resume & Career

- Purpose
    - Get an interview
- Who is it for
    - Employer

- Interview goal
    - Engage interviewer
    - Display thought process
    - Show interest and attitude
    - Demonstrate what is it like to work with you
    - Get offered a position

- Interviewer's goal
    - Determine whether or not to spend more time with you than family

- What if get an offer
    - Stop interviewing
    - Other job always looks better

Quiz 2

Development

- Software development cycle
  - Requirements
  - Design
  - Coding
  - Unit testing
  - Integration testing
  - Formal / acceptance testing
  - Maintenance

- Mandatory verb in a good requirement - **shall**

-

Agile

- Iterative and incremental development
- Requirements and solutions evolve via cross-functional self-organizing teams
- Adaptive planning rather than predictive planning
- Flexible response to changes
- Software delivery is the measure of progress

- Pros
  - Adaptive methods focus on adapting quickly to changing realities
  - Best for developmental and non-sequential projects.

- Cons
  - Difficult in describing exactly what will happen in the future - **unpredicted**
  - Flaws
    - Building a house without blueprint
  - **Insufficient training ci**ted as most significant cause for failed agile projects
  - **Teams are not focused to meet commitments**
  - Problem **solving in scrum meetings can take time of too many members**
  - **Team members get boxed** into certain roles preventing cross-training
  - **Lack testing automation**
  - Allow**s technical debt to build up if only focusing on increased functionality.**

- Criticism of Agile
  - can be inefficient in large organizations or certain types of projects
  - seems best for developmental and non sequential projects (sequential - waterfall
  - many organizations believe agile is too extreme

- Iterations
    - Short term frames that last 1 - 4 weeks

- **Scrum - early implementation of agile**
    - Core roles
        - Product owner
            - Voice of consumer
        - Development team
        - Scrum master
            - Like PM, facilitates stuff
    - **Scrum is the manner of restarting after minor infraction**
    - **Scrum starts with backlog**

    - **Components**
        - Sprint
            - **Basic unit of development over a fixed period of tim**e (2 weeks)
                - Planning meeting
                    - tasks identified
                - Daily scrum meeting
                    - What did you do, plan to do, obstacles? 15mins
                - Sprint review meeting
                    - Progress reviewed
                - Result - Working product - ready to ship
                    - All backlogged items implemented in Sprint
        - Sprint backlog
            - Items needed to be done prioritized by risk, business value ..
            - Contains
                - Product owner's assessment of effort
                - Developer's assessment of effort
        - **Velocity**
            - **Number of units of work / interval**
        - Burndown chart
            - Chart of work left to do VS time - updated daily
        - Burn-up chart
            - Chart of work completed and total amount of work VS time, updated daily.
        - Key principle
            - A customer can change their minds about what they want during development
        - **Unpredicted challenges are hard to address in a planned manner**
        - **Accept that problem cannot be fully defined**

- **User stories**
    - To capture the description of a software feature from an end-user perspective
    - Short description of something that your user will do when they come to your website
        - Displaying home screen is not a functionality in User stories
        - Login into the system is
            - Because it provides **benefit**
    - Format
        - As a user, I want to …, so that …
    - MoSCoW
        - Way to prioritizing user stories
        - must have, should have, could have, won't have

Waterfall

- Project is divided into sequential phases, with some overlap and splashback acceptable between phases
- **Emphasis on**
    - planning,
    - time schedules,
    - target dates,
    - budgets,
    - implementation of an entire system at one time
- **Tight control**
    - is maintained by **extensive** written documentation, formal reviews
    - Is approval by user and information technology management
- Benefits
    - Time spent early can reduce costs later
    - Well suited for projects where
        - requirements and scope are fixed
        - Product is firm and stable
        - Technology is clearly understood
- Drawbacks
    - **Clients may not know exact requirements** until they see working software
    - Impossible to perfectly complete a phase before moving onto next
    - Clients may change their requirements constantly.
    - **Designers may not be aware of future difficulties** when writing a design for an unimplemented software product.
    - Project stakeholders may not be fully aware of the capabilities of the technology being implemented.
    - **Lengthy delivery cycle.**
    - Changes in requirements lead to
        - redesign

        - Development
        - Retesting
        - Long development to market timeline
    - Increased costs
- Phase**s**
    - **Conception**
    - **Initiation**
    - **Analysis**
    - **Design**
    - **Construction**
    - **Testing**
    - **Production / implementation**
    - **Maintenance**

- **Planning poker (agile)**
    - Number of sequence - **fibonacci**
    -

- **Use cases (follow waterfall methodology**
    - Detailed description and the steps involved with a user's interaction with the application on how it provides one specific functionality without specifying **technology, implementation or specific user entry**
        - Displaying home screen is not a functionality
        - Login into the system is
            - Because it provides **benefit**

    - **CRUD - create, read, update, delete**
        - Basic functionality of persistent storage
    - **Starts with - the system shall / the user shall**

    - Components
        - Title & number
        - Priority
        - Status
        - Description
        - User goal
        - Desired outcome
        - Actor
        - Dependent use cases
        - Requirements
        - Pre-condition
        - Post-condition
        - Trigger

        - **Workflow**
            - **Main difference between use cases and design use cases**
            - **In DUC, MVC should be mentioned**
        - Alternative workflow

- **Designed Use Cases**
    - Detailed description of a functionality that benefits the user
    - **Different from UC, it mentions essential implementation details**
        - **MVC aspects of the project by specifying which files, API requests, calls, etc, are being used in the <u>Workflow</u>**
    - **User action comes first in the workflow**

    - MVC - model view controller
        - Software architectural pattern for implementing user interfaces

Layered architecture

- Presentation layer
    - The layer of code processing input from screens
        - Form class - validation without database access
            - Send flow of control to **action class**
            - **eg: checking data entry (names starting with letters, birth date within last 120 years, entry not empty)**
        - Action class - user requested action with valid data
            - **eg: AddUser method**
                - **Collects input from user form object parameter**
- Business logic
    - High level functionality invoked from presentation layer
        - Dispatch class - validation with database access
            - **eg: AddUser method**
                - **checking if user has credentials to add a new user**
                - **checking another user doesn't exist already with the same username**
        - Manager class - manage data access objects
            - **eg: AddUser method**
                - **No validation**
                - **Calls data access methods**
- Data access
    - Low level database interface methods invoked from manager layer
        - DAO class
            - **eg: AddUser method**
                - **Add user to database**
                - **No validation**
- Database - database connectivity code

UML - unified modeling language

- Provide a standard way to visualize the design of a software system

Quiz 3

Design principle

- A software project can either be Object-oriented or Aspected-oriented, or both at the same time

Aspect-Oriented Software Development
- Code that spans all aspects of the project
- eg:
    - Calls to audit system
        - Made from most if not all Manager class methods
        - Maintains table of state transitions for messages and users
    - Error processing
        - Made from most if not all methods
        - Exceptions thrown when errors encountered
        - Exceptions caught and processed at central location

- Separation of concerns
    - Concern is a set of information that affects the code of a computer program
    - Design principle for separating a computer program into distinct sections, such that each section addresses a separate concern

Test Driven Development
- Code is only written to pass tests
- Pros
    - Programmers to be more productive
    - Less need for debuggers
    - Resultant code is modularized **flexible and extensible**

- Cons
    - Tests must be maintained as part of project overhead
    - Refactoring or design changes may result in many changes in tests
    - High number of passing tests lead to **false sense of security**
    - Developer and test author are same leading to some blind spots
- **Steps**
    - **Fail**
    - **Pass**
    - **Refactoring**

True/false question :
- When the developer and test author is the same person, TDD leads to the same blind spots in test coverage. - T

- Using TDD, writing code to implement new functionality is done only after writing a failing test verifying the new functionality is not working. - T

Requirement

- **Functional requirements** describe a function user want (**ALL ABOUT USER**)
    - Must start with “system shall”
    - Can’t require anything from user
    - Especially for the system
- **Data Model** / **Business Rule** defines the fields of an object:
    - “The system shall define a user profile to be”
- NO **if, when**
- NO **why**
- NO **vague terms → not testable**
- NOT **extensible, polymorphic**
- Derived from **steps of use cases**
- Should be
    - **cross referenced.**
    - **Numbered** as subsystem based to be able to easily extend

Financial Part

- Working hours
    - Official hours
        - 8am - 5pm typically. One hour from noon to 1pm - lunch time
        - Administrative employees must follow
            - **Receptionist, secretaries, payroll, human resources**
        - Executives, managers, engineers do not follow
            - Given flexibility to come and go as needed as long as
                - Job gets done
                - Hours are worked per week
                - Present for meetings and when otherwise needed
                - Co-workers know your schedule
        - **40 hours per week is full-time**
            - 5 * 8 hours

- Overtime
    - Working more than 8 hours per day or 40 in a week
        - “Casual” overtime → unpaid time given to company
            - Sometimes expected, sometimes unusual
        - Sometimes “extended work week” → paid time for hours worked beyond normal
            - Must be approved by management
            - Impacts salary budgets
            - Sometimes begins after initial amount of casual OT
            - Often paid your straight hourly rate
        - Sometimes forbidden
            - Violation of employment contract
            - Cause for termination
    - **A non-exempt employee is always paid for OT hours**

- Compensation
    - Salary
        - Determined at time of hire
        - Adjusted at a performance review
        - Paid a short delay after work performed
            - Weekly
            - Alternating week
            - Bi-monthly
            - Monthly
        - Exempt or non-exempt
            - Eligible for OT
                - Yes → non-exempt; No → exempt
    - Vacations -- paid time off for recognized holidays

        - Each company has a list
        - Floating holiday - day off of your choice
        - Comprehensive leave
            - **vacation, sick time, doctor's appointment**
        - Lump award at beginning of year or accrued over time
        - Duration extends with years of service
            - eg: 2 weeks year 1, 2 weeks years 2 - 5, 3 weeks years 6 - 10, 4 weeks years 10+
    - Jury duty - fixed or unlimited number of days
    - Bereavement
        - On approval - fixed
    - Tuition reimbursement
        - On approval - reimbursement on receiving A or B grade in job related classes
    - Bonus
        - One time payment to recognize achievement, employee sign on, or referral
        - Could be cash, stock, stock options or combination
        - Stock and stock options are often vesting
            - **"Golden handcuffs"**
                - Money is yours if you stay long enough, forfeited if you do not
        - Annually awarded at performance review
        - "Spot" bonuses awarded anytime
    - Life insurance - company provided and employee supplemented
    - Legal service
    - Discounts at amusement parks

    - Employee stock ownership plan (ESOP)
        - Buying stock from company through payroll deductions
            - Regular deductions, purchase made quarterly
            - Limit to percentage of salary - ex: 10%
            - Defined enrollment period (once per quarter)
            - Discounted price or with company match - ex: 15%
            - Set price
                - Private stock: internally set price
                - Public stock: average traded price for the period
    - Retirement vehicles - many options
    - Health / dental / vision insurance
    - Optional FLEX spending accounts (health care, day care)
        - **Pre-tax dollars in an account used for eligible expenses**

- Stock options

    - **Option to purchase a set amount of stock for a duration in the future at set price**
        - Price established at time of stock option award
    - Exercise - Purchase stock using your stock option
    - **“Underwater”**
        - If price at purchase time is less that stock option price
        - They are **worthless**
    - Tax basis - reset to value at the data of purchase
        - Difference between option price and today’s price is a capital gain taxable in year of stock purchase
            - Sometimes taxed as ordinary income

- **Retirement options**
    - Many plans allow loans against balance up to 50%
        - Paid back with interest via payroll deductions
        - Early penalty applies if not paid back
            - Taxable event: 10% penalty fee, plus income taxes at normal rate

    - Employee stock retirement plan (ESRP)
        - Company contribution to retirement fund
        - Earnings are tax-deferred
        - % of salary given to all employees eligible
            - May need to be an employee on Dec 31 the prior year
            - May need to have worked fixed number of hours - 850 in year
            - May have sale restrictions
            - Forfeit allotment if no longer employed at the end of calendar year
            - Non-employee stock assets are not employee directed
            - Contribution at end of fiscal year
    - **401k (profit)**
        - Pre-tax dollars diverted into investment account**(401k-pre)**
            - Earnings are tax-deferred
        - Post-tax dollars diverted into investment account**(401k-roth)**
            - Earnings are tax-free
        - Can withdraw at age 59 ½
        - Employers match funds diverted"100% of the first 6%"
        -
            - **100% match on first $3000**
            - **10% match on next amount up to 10% of salary**
            -
            -
        - Highly compensated employees max based on average of deferrals by non-highly compensated employees

            - $120,000 (2016)
            -
        - Max dollar limit per year
        - Contributions through payroll deduction
        -
    - **403b (non-profit)**
        - Employee **pre-tax** diverted into investment account for non-profit organizations
    - **457b**
        - Employee **pre-tax** dollars diverted into investment account for **governmental** employees
        - Contribution limits same as 401k and 403b
    - Pension plans
        - Company and / or employee contributes
        - Distribution in either lump sum or based on age, year of service and salary

- **Employment policies**
    - Employment **"at will"**
        - The employer is free to discharge individuals "for **good cause, or bad cause, or no cause at all**," and the employee is equally **free to quit, strike, or otherwise cease work**
    - Probation employment
        - Specified employment period with a termination
    - Contract employees
        - Fixed contract duration
        - Often paid more due to no benefits

- Performance Review
    - Company vehicle for feedback, growth, salary adjustments, promotion
        - Typically done annually at your anniversary date of hire
        - Can be done as "focal point" review
            - Entire organization does review at the same time
        - Initial review could be after 6 months
        - Out-of-cycle reviews are possible
        - UCSD academics: Once every three years.

- Performance Review Process
    - Employee phase:
        - Employee writes assesses goals made during the prior cycle
            - Initial goals done at hire
        - Employee writes goals for following year
        - Ethics statements are reviewed

    - Time Charging procedures are reviewed
  - Manager phase
    - Manager writes assessment of employee's goals
    - Manager approved goals for the following year
    - Manager writes performance evaluation listing strengths and areas for growth
  - Compensation phase
    - Salary adjustment announced with effective date in future
      - **Merit** – increase due to job performance or increased responsibilities
      - **Promotion** – increase due to increased responsibilities
      - **Equity** – increase to raise to market values: Amazon & Google: 10%
- Roth IRA(IRA:Individual Retirement Account)
  - $5,500 contribution to retirement account (2016 amount) in **post tax** dollars
  - Age 50+: can add additional (2016: $1,000)
  - Grows tax **free**
  - **Contribution limit dwindles** as income rises ($117,000: full - $132,000: none)
- Traditional IRA
  - $5,500 contribution to retirement account (2016 amount in **pretax** dollars
  - Age 50+: can add additional (2016: $1,000)
  - Grows tax **deferred**
  - **Pre-tax amount dwindles as income rises** (2015: <$61,000) if combined with 401K

- Adjusted Gross Income (AGI)
  - **Salary + Interest + Dividends + / - capital gains - personal exemption - adjustment**
  - **SIDCPA**
    - Personal exemption $4050 (2016)
    - If your adjustments exceed your personal exemption: itemize them
- **Tax forms**
  - W2: employer listing
  - W4: employee to determine withholding rates
  - 1099 forms
    - Report income other than wages
  - 1040 starting form for Federal Income Tax
    - Schedule A: deduction
    - Schedule B: Interests and dividends
    - Schedule C: self employment expenses
    - Schedule D: capital gains and losses
    - Schedule K1: partnership
  - California 540: starting form for California income tax

Quiz 4

- Patterns recur in many applications
- Favor **composition** over **inheritance**
- Classes should be closed for **modification** but open for **extension**
- Program to an **interface, not an implementation**
- Strive for **loosely** coupled design for objects that interact
- Encapsulate aspects of your application that **varies** and separate them from what **stays** the same
- Depend upon **abstraction, not on concrete classes**

- In the decorator pattern,
    - Decorators and objects being decorated are of the **same** type

- Design patterns rock
- Java IO, networking, sound APIs
- Rubberducks make a squeak
- Most patterns follow from OO principles
- Not your own failures
- High level libraries -- frameworks
- Pattern that fixed the simulator -- strategy
- Patterns go into your brain
- Learn from the other guy's success
- Patterns give us a shared vocabulary

Testing

- Unit Test
    - **White box**
        - Testing **individual units** of code
        - Verifies that individual components work individually
        - Performed **locally** by software developer
        - Performed (usually) alone
        - Done alongside **coding**
        - Performed (sometimes) as an **independent phase** of development
        - Sometimes **Informal**

- Developer Integration Testing
    - **White Box**
        - Testing **all code**
        - Verifies that **all code** works together
        - Performed **locally** by software developers
        - Usually performed as **a separate phase** of development
        - Sometimes informal, sometimes formal

- System Integration Testing
    - **White box**
        - Testing all code on **development platform**
        - Perform on test platform by developers
    - **Gray box**
        - Verifies system works when **installed from scratch**
        - **Non-code**:
            - Load testing, performance, security
        - **Destructive testing**:
            - Verification within allowable failure limits
        - System installed fresh
        - Perform on test platform by others
    - Usually performed as a separate phase
    - Informal/Formal
    - Code requires authorization of change making

- Formal Testing
    - **Gray box**
        - Independent testing on test platform
        - Verifies system works when tested by non-developers
        - System installed fresh
        - Performed on test platform by test engineers
        - Performed as a separate phase
        - Code freeze, changes require **authorization**
        - Formal test **plans** followed, **reports** generated

- Verification and Validation Testing
    - **Gray box/Black box**
        - Independent testing on test platform (gray)
        - Performed by outside agency or customer (black)
        - Performed as separate phase
        - Code freeze. Authorization required for code changes
        - Formal test plans followed, reports generated

- Alpha or Beta Testing
    - **Black box**
        - Limited released to few customers on production site
        - Coordinated with customer
        - Performed as a separate phase
        - Code changes require
            - New build
            - Formal testing approval
            - Customer approval

- **Acceptance**: successful passing of alpha or beta testing

- Regression Testing
    - **All Colors**
    - Aspect of testing that verifies **existing functionality** that still works
    - Crosses all phases of testing

Quiz 5

- **Cohesion (low → high)**

    - Degree to which the elements of the module are functionally related.
    - Coincidental ⇒
        - Grouped together
        - Ex: a utilities class
    - logical ⇒
        - Categorized to do the same thing, even if different natures
        - Ex: grouping all input of mouse and keyboard routines
    - temporal ⇒
        - Grouped by program execution
        - Ex: a function called after catching an exception which closes files, creates an error log and notifies the user.
    - procedural ⇒
        - Grouped by sequence of execution
        - Ex: a function which checks file permissions and then opens the file
    - communicational ⇒
        - Grouped by operation on same data
        - Ex: code which operates on the same record of information
    - sequential ⇒
        - Output is another's input
        - Ex: a function which reads data from a file and processes the data
    - Functional -- **preferred**
        - Grouped from contribution to single task
        - Ex: tokenizing a string of XML

- **Coupling**
    - Degree to which each program module relies on another module
    - low preferred for good design
    - high readability/ maintainability

    -
    - High → low
        - Content ⇒
            - Module relies on inner workings of another
        - Common ⇒
            - Modules share global data
        - External ⇒
            - Modules share externally imposed data formal
        - Control ⇒
            - One module controls flow of another
        - Stamp/ Data structured ⇒
            - Share **data structure** and only use part of it
        - Data ⇒
            - Share **data** by passing parameters
        - Message ⇒
            - Decentralization with communication through parameters or message
        - **No -- preferred**
            - Do not communicate with each other
    - Tight coupling
        - A change has **ripple effect** of changes in other modules
        - Assembly of modules requires more efforts due to increased inter-module dependencies
        - **Reuse is difficult** due to dependent modules that must be included

- What do you want ⇒ **loosely coupled, highly cohesive**
    - **Highly cohesive**
        - Pros
            - Robustness
            - Reliability
            - Reusability
            - Understandability
    - **Loosely coupled**
        - Pros
            - High readability
            - High maintainability
    - Tight coupling causes **Ripple effect**

Decision and Ethics

- Etiquette
    - Code of behavior that delineates expectations for social behavior
    - **Hygiene manners**
    - **Courtesy manners**
        - One’s ability to put the interests of others before oneself
    - A set of written and unwritten rules of conduct

- Guideline
    - Statement by which to determine a course of action
    - Following is not mandatory, not enforced

- Policy
    - Deliberate system of principles to guide decisions and achieve rational outcomes
    - Generally adopted by a governance body within an organization

- Law
    - A system of rules that are created and enforced through social or governmental institutions to regulate behavior

- Decision making
    - Decision-making is the process of identifying and choosing alternatives based on the values, preferences and beliefs of the decision-maker

- Programming ethics
    - The ethical guidelines that developers are expected to follow and apply when writing programming code

Design Pattern

- Patterns give a shared vocabulary
- Patterns recur in many application
- Adapter
    - **Converts the interface** of a class into another interface clients expect
    - Use an adapter where you need to use an existing class and its interface is not the one you have

- Facade

    - The pattern provides a unified interface to a set of interfaces in a subsystem
    - High-level interface that makes the subsystem easier to use
    - One advantage of facade - decoupling
    - Loose coupling

- Singleton
    - The pattern ensures you have at most one instance of a class in your application
    - A "singleton" is a class that manages an instance of **itself**
    - Lazy initialization:
        - the instance is only created when the static method is first called
    - “**public static** synchronized Singleton getInstance()” (synchronized keyword prevents two threads from entering the method at the same time)

- Iterator (tested in quiz 4)
    - The pattern allows a way to access the elements of an aggregate object **sequentially** without showing its underlying representation

- Template method (tested in quiz 4)
    - Define the skeleton of an algorithm in an operation, deferring some steps to sub-classes
    - Let subclasses redefine certain steps of an algorithm without changing the algorithm’s structure
    - Abstract methods
    - Is defined in an **abstract** class

- Composite (tested in quiz 4)
    - Compose objects into tree structures to represent part-whole hierarchies
    - Let clients treat individual objects and compositions of objects uniformly

- State
    - Allows an object to alter its behavior when its internal state changes its class
    - The object will appear to change its class
- Proxy
    - Provides a surrogate placeholder for another object to control access to it

- Observer

    - One-to-many dependency between objects so that when object changes state, all its dependents are notified and updated automatically
    - Listening to “button” class
    - One subject likes to talk to **many** observers
    - Observers are **dependent on** the subject
    - Like to be notified when updates happen

- Strategy
    - Define a family of algorithms, encapsulate each one and makes them interchangeable.

- Factory
    - Defines an interface for creating an object but lets subclasses decide which class to instantiate

- Decorator
    - Attach additional responsibilities to an object dynamically
    - Keeping the **same interface (supertype)**

- Command
    - Encapsulates a request to an object letting you parameterize clients with different requests
    - Developer Integration Testing sometimes is also called - DIT

    - The MVC is a **compound** pattern
        - Consisting of
            - Observer
                - Listen to “button” class
                - Observers are dependent on the objects
                - Observers like to be **notified** when **updates** happen
            - Composite
            - Strategy
        - The model of MVC makes use of the **observer** pattern
        - The View in MVC uses **composite** pattern to implement the user interface, which usually consists of nested components like panels, frames and buttons.
        - the Controller in MVC uses **strategy** pattern because the View can use different implementations of the controller to get different behavior.

- **"Good enough" design is better than "perfect design".**
- **Slider**
    - **Scope**
    - **Time**
    - **Quality**
    - **Cost**

Guest Lecture

- What makes a successful software engineer
    - Managing complexity
    - Requirement, planning, integration, testing, design
    - SWE > Programmer
- Software engineering is all about **managing complexity**
- Programming is about **getting code to work**

- PPP - pseudocode programming process
- Software engineering is a set of **accepted** practices implemented to product software
- Avoid “clever” code
- Design principles matter
    - **Abstraction**
    - **Encapsulation**
    - **Strong Cohesion**
    - **Loose Coupling**
    - **Modular Design**
    - **Object-oriented**
    - **Inheritance**

    - The most successful app from our guest speaker was based on a jigsaw puzzle game

- Two main things to evaluate a project
    - **MVC**
    - **Layer**

- Over-engineering
    - There is a **cost** to SW engineering processes
    - Make sure the cost is not higher than it yields

- Personal Relations
    - Your attitude is critically important
    - Best SW Engineer ever + bad attitude = **dead career**
        - Constantly negative
        - Overly sarcastic / snarky
    - Nobody will want to work with you
        - Including people who make hiring or promotion decisions

- Never respond to email while angry
    - Wait for 24 hours
    - Write your cathartic response but do not send
    - People remember negative interactions for a long time

- Personal Relation cont..
  - The best colleagues..
    - Humble
    - Make time for you
    - Don't make you feel dumb
    - Say "hi" to people :)

- Work Ethic
  - A good work ethic is necessary to career success
  - A bad work ethic is just as toxic as a bad attitude
  - Be proactive. Break up your work into small milestones to keep from getting behind
  - Be curious
    - Read up on new technologies languages, etc.

- Enterprise Systems
  - Not all software is **consumer-based**
  - Business (Enterprise)
  - Turnkey systems
  - Databases, order management, account services, customer services
  - Other Devs may be your customer

- Your Career
  - You should enjoy what you do
  - Do not be afraid to change
  - You do not have to be a SW engineer

- Come out of your shell
  - More personal interaction as your career advances
  - Meetings, code reviews, work with manager designers, QA
  - You will be expected to participate
  - Presentations
  - Building software is collaborative

    - Do not stress. It is a skill like any other that can be improved with practice

- Non-Standard hours
    - Releases & client deployments
        - Weekend “rollouts”
        - “On-call”
        - Shifts
    - Production outages
        - Emergency work

- Time Management
    - Set aside work time
        - Blick time on your calendar
    - Plan your day first
        - Look at calendar
        - Do not dive right into work
    - Handling interruptions
        - Keep context switching low
        - Respond to emails at set time per day
        - Acceptable response time is 24 hours

- Monetize an app
    - Features are great! But **short-lived**
    - Advertise your app in another APP
        - Web, email, TV, & print → not effective

    - ARPU - average revenue per user

Design Pattern Bullet Points

Overall

- Knowing the OO basics does not make you a OO designer
- Good OO designs are **reusable, extensible, maintainable**
- Patterns show you how to build systems with good OO design qualities
- Patterns are proven object-oriented experience
- Patterns do not give you code, they give you general solutions to design problems. You apply them to your specific application
- Patterns are not invented, they are discovered
- Most patterns and principles address issues of change in software
- Most patterns allow some part of system to vary independently of all other parts
- We often try to take what varies in a system and encapsulate it
- Patterns provide a shared language that can max the value of your communication with other developers

Observer pattern

- The observer pattern defines a one-to-many relationship between objects
- Subjects or as we also know them, observables, update observers using a common interface
- Observers are **loosely** coupled in that the observable knows nothing about them, other than that they implement the observer interface
- You can push or pull data from observable when using the pattern
- Do not depend on a specific order of notification for your observers
- Java has several implementations of the observer pattern, including the general purpose **java.util.Observable**
- Watch out for issues with the java.util.Observable implementation
- Do not be afraid to create your own Observable implementation if needed
- Swing makes heavy use of the observer pattern, as do many GUI frameworks
- You will also find the pattern in many other places, including **JavaBeans and RMI**

Factory pattern

- All factories encapsulate object creation
- Simple factory, while not a bona fide design pattern, is a simple way to decouple your clients from concrete classes
- Factory method relies on inheritance: object creation is delegated to subclasses which implement the factory interface
- Abstract factory relies on object composition:

    - Object creation is implemented in methods exposed in the factory interface
- All factory patterns promote loose coupling by reducing the dependency of your application on concrete classes
- The intent of factory method is to allow a class to defer instantiation to its subclasses
- The intent of abstract factory is to create families of related objects without having to depend on their concrete classes
- The dependency inversion principle guides us to avoid dependencies on concrete types and to strive for abstractions
- Factories are a powerful technique for coding to **abstractions**, **not concrete classes**

Singleton pattern

- At most one instance of a class in your application
- provides a **global access** point to that instance
- A **private** constructor, a **static** method combined with a **static** variable
- Examine your performance and resource constraints and carefully choose an appropriate singleton implementation for multithreaded applications
- Beware of the double-checked locking implementation; it is not thread-safe in versions before java 2, version 5
- Be careful if you are using multiple class loaders; this could defeat the singleton implementation and result in multiple instances
- If you are using a JVM earlier than 1.2, you will need to create a registry of singleton to defeat the garbage collector

Command pattern

- decouples an object, making a request from the one that knows how to perform it.
- A command object is at the center of this decoupling and encapsulates a receiver with an action (or set of actions).
- An invoker makes a request of a Command object by calling its execute() method, which invokes those actions on the receiver.
- Invokers can be parameterized with Commands, even dynamically at runtime.
- Commands may support undo by implementing an undo method that restores the object to its previous state before the execute() method was last called.

- Macro Commands are a simple extension of Commands to be invoked. Likewise, Macro Commands can easily support undo().
- In practice, it is not uncommon for “smart” Command objects to implement the request themselves rather than delegating to a receivers.
- Commands may also be used to implement logging and transactional systems.

Adapter & Facade pattern

- When you need to use an **existing** class and **its interface is not the one you need, use an adapter.**
- An adapter changes an interface into one a client expects.
- Implementing an adapter may require little work or a great deal of work depending on the size and complexity of the target interface.
- There are two forms of the Adapter Pattern: object and class adapters. Class adapters require multiple **inheritance.**

- When you need to **simplify** and **unify** a large interface or complex set of interfaces, use a facade.
- A facade decouples a client from a complex subsystem.
- Implementing a facade requires that we compose the facade with its subsystem and use delegation to perform the work of the facade.
- You can implement more than one facade for a subsystem.

- An adapter wraps an object to change its interface, a decorator wraps an object to add new behaviors and responsibilities, and a facade “wraps” a set of objects to simplify.

Template method pattern

- Defines the steps of an algorithm, deferring to subclasses for the implementation of those steps
- Gives us an important technique for code reuse
- The template method’s **abstract class** may define **concrete methods, abstract methods and hooks**
- Abstract methods are implemented by subclasses
- **Hooks are methods that do nothing or default behavior in the abstract class, but may be overridden in the subclasses**
- To prevent subclasses from changing the algorithm in the template method, declare template method as **final**

- The **hollywood principle** guides is to put decision-making in high-level modules that can decide how and when to call low level modules
    - Don't call us, we'll call you
- ~~You will see lots of uses of the template method pattern in real world code, but do not expect it all to be designed by the book~~
- The strategy and template method pattern both encapsulate algorithms, one by inheritance and one by composition
- The factory method is a specialization of template method

Iterator pattern

- Allows access to an aggregate's elements **without exposing its internal structure**
- Takes the job of iterating over an aggregate and encapsulates it in another object
- When using an iterator we relieve the aggregate of the responsibility of supporting operations for traversing its data
- An iterator provides a common interface for traversing the items of an aggregate, allowing you to use polymorphism when writing code that makes use of the items of the aggregate
- We should strive to assign only one responsibility to each class

Composite pattern

- Provides a structure to hold both individual objects and composites
- Allows clients to treat composites and individual objects uniformly
- A component is any object in a composite structure. Components may be other composites or leaf nodes
- There are many design trade-off in implementing composite. You need to balance transparency and safety with your needs

State & Strategy pattern

- Allows an object to have many different behaviors that are based on its internal state.
- Unlike a procedural state machine, the State Pattern represents state as a full-blown class.
- The context gets its behavior by delegating to the current state object it is composed with.

- By encapsulating each state into a class, we localize any changes that will need to be made.
- The State and Strategy Patterns have the same class diagram, but they differ in intent.
- Strategy Pattern typically **configures Context classes with a behavior or algorithm.**
- State Pattern allows a **Context to change its behavior as the state of the Context changes.**
- State transitions can be controlled by t**he State classes or by the Context classes.**
- Using the State Pattern will typically result in a **greater number of classes** in your design.
- State classes may be shared among Context instances.

Proxy pattern

- Provides a representative for another object in order to control the client's access to it. There are a number of ways it can manage that access.
- A **Remote** Proxy manages interaction between a client and a remote object.
- A **Virtual** Proxy controls access to an object that is expensive to instantiate.
- A **Protection** Proxy controls access to the methods of an object based on the caller.
- Many other variants of the Proxy Pattern exist including caching proxies, firewall proxies, copy-on-write proxies, and so on.
- Proxy is structurally similar to Decorator, but the two differ in their purpose.
- The Decorator Pattern adds behavior to an object, while a Proxy **controls access**.
- Java's built-in support for Proxy can build a dynamic proxy class on demand and dispatch all calls on it to a handler of your choosing.
- Like any wrapper, proxies will increase the number of classes and objects in your designs.

MVC

- The model view controller pattern is a **compound** pattern consisting of the **observer, strategy and composite** patterns
- The model makes use of the observer pattern so that it can keep observers updated yet stay decoupled from them

- The controller is the strategy for the view. The view can use different implementations of the controller to get different behavior
- The view uses the composite pattern to implement the user interface, which usually consists of nested components like panels, frames and buttons
- These patterns work together to decouple the three players in the MVC model, which keeps designs clear and flexible
- The adapter pattern can be used to adapt a new model to an existing view and controller
- Model 2 is an adaption of MVC for web applications
- In model 2, the controller is implemented as a servlet and JSP & HTML implement the view

Main topics Gary mentioned in class:

- Review old quizzes
- Review design patterns and design principles
    - Know the definitions and basics of functionality
- Review end of the chapter bullet point
- Review testing
- Review cohesion and coupling
- Review agile and waterfall methodologies
- Review financial lecture focusing on high-level ideas(cover in quiz review)