CSCI 330 Exam 1 Study Guide

True/False questions all relevant topics.

TLAs (2 pts each)

1. **TLA** - three letter acronym

2. **SIT** - System integration testing

3. **COTS** - Commercial/Commodity off the Shelf

4. **ADT** - abstract data type

5. **CMM** - capability maturity model

6. **V&V** - validation and verification

7. **UML** - unified modeling language

8. **XP** - extreme programming

9. **UP** - unified process

10. **ACM** - association for computing and machinery

11. **SEI** - software engineering institute

12. **ISO** - international standards organization

13. **CASE** - computer aided software engineering

14. **KLOC** - thousands of lines of code

15. **API** - application programming interface

16. **IEEE** - institute of electrical and electronics engineers

17. **TDD** - test driven development

18. **LOC** - lines of code

19. **CABTAB** - code a bit, test a bit

20. **YAGNI** - you ain't going to need it

Term Definitions: Define each term (1 to 2 sentences).

1. **Client** – person with the money requesting a product

2. **Developer** – person who goes gathers requirements and creates a product

3. **User** – person who is the end user of the product

4. **Scenario** – a possible interaction between the user and the system (ordering a pizza)

5. **Use** **case** – list of steps taken to get to a goal (steps through a scenario)

6. **Verification** – doing things right (whitebox) - Are we building the product right?

7. **Validation** – doing the right thing (blackbox) - Are we building the right product?

8. **Traceability** – being able to trace an activity back in time to a requirement gathered from client

9. **Quality** – value and/or usefulness of a thing

10. **“git** **commit”-** commits changes to local clone before a push is performed

11. **“git** **pull”** - grabs from the master branch and merges with current local branch

12. **“git** **push”** - pushes your branch to a remote

13. **Design** **Patterns** – metaphors that describe common organizations of software components and their relationships to each other.

14. **Copy-lefting** – part of a software license that says this software is open source, any changes made to the software must also be open-source

15. **Class** **vs.** **Object** – category(class), object(specific instance of a class)

16. **Coupling** **vs.** **Cohesion** –

* **cohesion** refers to the *degree to which the elements of a* [*module*](http://en.wikipedia.org/wiki/Module_(programming)) *belong together (want high cohesion)*
* ***coupling*** *or* ***dependency*** *is the degree to which each* [*program module*](http://en.wikipedia.org/wiki/Module_(programming)) *relies on each one of the other modules (want low coupling)*

Long Answer: Provide the answer to the following questions.

**1. What is software construction?**

* It is the process that includes
  + Problem definition
  + Requirements development
  + Construction planning
  + Software architecture, or high-level design
  + Detailed design
  + Coding and debugging
  + Unit Testing
  + Integration Testing
  + Integration
  + System Testing
  + Corrective Maintenance

**2. What is the role of design and implementation in software construction?**

* Design and implementation are tightly integrated, “design is a wicked problem”, design on new products is like a dark alley, you don't know until you get to the end of construction whether or not is a bad design, should always be going back and redesigning (advantage of a agile design)

**3. Why would you choose one programming language over another?**

* Java - Garbage collection, exception handling, VM(Byte-code) vs. Machine code, the overhead involved in the extra features, security risk
* C++ - more control over resources, can do system calls, pointers
* Risk mitigation

**4. What is software’s primary technical imperative, why and how?**

* Used to address a set of requirements, and figuring out how to manage the complexity of those requirements. Careful analysis of requirements, encapsulations, abstraction, language choice, etc.

**5. What is defensive programming?**

* Pearls before swine, ensure the user can't do something that you never intended for.
* Attitude towards software construction – your software will almost certainly be used in a way you never intended.

**6. What is the role of checklists in software engineering?**

* Process improvement
* Make sure everything gets done

**7. What is continual process improvement?**

* Going back and changing processes that are ineffective, and thus, improving the process.

**8. What is a validation-centric software lifecycle model and why is it dangerous?**

* Don't look at what is inside, just pay attention to see if right results were found.
* Validation looking at it like a black box, not caring what is inside, just the output.
* Verification looking at the inside, and insuring good practices were involved, insuring the output is valid.
* Overtime, if no one cares about what is inside, it will look scary and diseased.

**9. What are the five phases in the TriBITS Lifecycle (EP, RS, PG, PM, UM) and how do they differ?**

* **EP – Exploratory** – explore alternative approaches and prototypes, no real/sufficient unit test cases, not a real foundation for the real product
* **RS – Research Stable –** strong unit tests written as product is developed, high quality design, lacks documentation, meant for “expert users”, meant to be a good foundation for continued development
* **PG – Production Growth –**  includes all good qualities of RS stage, increased checking of input and error checking, more documentation, implements user interfaces, clean structures through refactoring, better regulated backwards compatibility, better portability, more customer codes
* **PM – Production Maintenance –** includes all good qualities of production grown, bug fixes, performance tweaks, maintain rigorous backwards compatibility, maintenance through user community if needed
* **UM – Unspecified Maturity** – no official indication of maturity or quality

**10. What do we mean by “Better, Faster, Cheaper: Pick two of the three”?**

* You can either build a better and faster product, at the cost of it being cheaper to produce, or you can build a faster and cheaper product, at the cost of a lack of quality, or you can build a better and cheaper product, at the cost of it being slower.

**11. What do we mean by “Better, Faster and Cheaper: Get all three” ?**

* Having enough time to implement all three. This project is set to attain all three levels of better, faster, and cheaper.” You have enough time to design and build a superior product, that is faster, and that mitigates costs.

**12. Describe the Pseudo Programming Process.**

* Design the Routine:
  + Check the Prerequisites – check to make sure the routine is absolutely needed by the projects' prereqs.
  + Define the Problem to be solved by the routine – should at least include
    - info routine will hide
    - inputs
    - outputs
    - preconditions
    - postconditions
  + Name the routine
  + Test the routine
  + Think about error handling
  + Think about efficiency
  + Research the algorithms and data types
  + Write the pseudocode
  + Check the pseudocode
  + Try a few ideas, keep the best
* Code the Routine:
  + Write the declaration
  + Turn pseudocode into high-level comments
  + Fill in the code below each comment
  + Check whether code should be further refactored
* Check the Code:
  + Mentally check the routine
  + Compile the Routine
  + Step through the code debugger
  + Test the code
  + Remove errors from the routine
* Clean up Leftovers
  + routine interface, general design quality, routine's variables, routine's statements and logic, routine's layout, routine's documentation, remove redundant comments.
* Repeat as Needed – if of poor quality, back up to pseudocode

**13. What are the two categories of secrets when hiding information?**

* Hide complexity
* Protect against change

**14. What are the common design patterns and what are examples of each?**

* **Abstract Factory** –Supports creation of sets of related objects by specifying the kind of set but not the kinds of each specific object.
* **Adapter –** Converts the interface of a class to a different interface.
* **Bridge –** Builds an interface and an implementation in such a way that either can vary without the other varying.
* **Composite –** Consists of an object that contains additional objects of its own type so that client code can interact with the top-level object and not concern itself with all the detailed objects.
* **Decorator –** Attaches responsibilities to an object dynamically, without creating specific subclasses for each possible configuration of responsibilities.
* **Façade** – Provides a consistent interface to code that wouldn’t otherwise offer a consistent interface.
* **Factory** **Method –** Instantiates classes derived from a specific base class without needing to keep track of the individual derived classes anywhere but the Factory Method.
* **Iterator –** A server object that provides access to each element in a set sequentially.
* **Observer –** Keeps multiple objects in synch with one another by making an object responsible for notifying the set of related objects about changes to any member of the set.
* **Singleton –** Provides global access to a class that has one and only one instance.
* **Strategy –** Defines a set of algorithms or behaviors that are dynamically interchangeable with each other.
* **Template** **Method**  - Defines the structure of an algorithm but leaves some of the detailed implementation to subclasses.
* **Find all the ones in the book pg 104.**

**15. For what types of system are agile approaches to development particularly likely to be successful?**

* Risky, really well suited to risky where that “dark alley” goes into account.

**16. Why are incremental process models considered by many to be the best approach to software**

development in a modern context?

* As you design and implement, you don't know what the end-design will be.
* Iterations – do some work, go back and redesign to make better, and then go into next phase

**17. What is the distinction between functional and non-functional requirements?**

* Fuctional – what it does, features, etc.
* Non-Fuctional – how it is done, security, how it does those functions

**18. Give reasons why eliciting requirements is difficult?**

* Customers don't know what they want, fuzzy notion
* Even if they do know what they want, describing it to you is something more difficult.
* Misunderstandings between client talk and nerd talk

**19. Briefly describe 3 approaches that may be used to identify object classes?**

* Identify the nouns (will often be the classes) – **find technical name for this**
* Look at the external entities, (interfaces in java)
* Data stores, represent state (state is/exists)

**20. Briefly summarize the test-driven development process.**

* Design a test to a class, designed to fail
* Write the test before the method/use case, then see if it passes

**21. What is a distributed source management systems and what are its strengths and weaknesses?**

* CVS – master source management system – one master file, people checkout a snapshot of the current state of the model
* Git – distributed – get the entire history of a repository, take a clone of the master.
* Disadvantage – having to work with incompatibilities on a regular basis, not forced to merge working branches on a regular basis.

**22. What is the procedure for starting/stopping a development session using a git clone when**

**pushing/pulling from remote clone, including contingencies for conflicts?**

* Pull - > Commit -a -> Pull -> Fix Conflicts -> Test -> Pull (to insure nothing new between test and next commit) -> Commit (optional) -> Push

**23. Briefly discuss the role of traceability in the software lifecycle.**

* Being able to trace your actions back to the requirements, step-by-step (k = k-1, and back), and eventually working your way back to the requirements.
* You should then be able to retrace your way back up to your current actions.

**24. Describe the “plan-do-check-act” cycle from in the ISO 9001:2000 standard as it is applied to**

**assessing the quality management elements of a software project.**

**PLAN**

Establish the objectives and processes necessary to deliver results in accordance with the expected output (the target or goals). By establishing output expectations, the completeness and accuracy of the [specification](http://en.wikipedia.org/wiki/Specification) is also a part of the targeted improvement. When possible start on a small scale to test possible effects.

**DO**

Implement the plan, execute the process, make the product. Collect data for charting and analysis in the following "CHECK" and "ACT" steps.

**CHECK**

Study the actual results (measured and collected in "DO" above) and compare against the expected results (targets or goals from the "PLAN") to ascertain any differences. Look for deviation in implementation from the plan and also look for the appropriateness/completeness of the plan to enable the execution i.e.,"Do". Charting data can make this much easier to see trends

over several PDCA cycles and in order to convert the collected data into information. Information is what you need for the next step "ACT".

**ACT**

Request [corrective actions](http://en.wikipedia.org/wiki/Corrective_and_preventive_action) on significant differences between actual and planned results. Analyze the differences to determine their root causes. Determine where to apply changes that will include improvement of the process or product. When a pass through these four steps does not result in the need to improve, the scope to which PDCA is applied may be refined to plan and improve with more detail in the next iteration of the cycle, or attention needs to be placed in a different stage of the process.