CSSE 373 Terminology

Exam 3 –Multiple Choice questions on the terms listed below

system – anything you want to think about as an indivisible unit

interface – describes how to think about a system as a unit, separates the inside from the outside

client – a role played by an agent that views a system from the outside

implementer – a role played by an agent that views a system from the inside

components – are themselves systems (subsystems) used to construct a larger system

separation of concerns – provides a separation between what a system does (captured by a contract) with respect to how a system does it (captured by the system's implementation)

abstraction – describes what a system does, not how it does it

information hiding – deliberately omits internal implementation details from the system's behavioral description

encapsulation – a technique for enclosing a system's internal data members, often enforced by a programming language constructs, e.g., private keyword

design-by-contract – is a standard policy that governs the engineering of software and is based on separation of concerns

external contract – aka operation contract, captures what must be true at the time an operation is called (i.e., its preconditions) and what will be true when the operation terminates (i.e., its postcondition)

steady-state values – all the values of an operation's parameter that are handled by the operation's implementation

non-steady-state values – all the values of an operation's parameter that are not handled by the operation's implementation

requires clause – is an external contract associated with an operation's header and captures what must be true about the operation's parameters at the time of a call to the operation (i.e., the preconditions); in terms of steady-state values, the requires clause explicitly identifies all steady-state values

trivial requires clause – when an operation handles all possible values of its parameters, then the requires clause is just the following: `requires true`

ensures clause – is an external contract associated with an operation's header and captures what will be true about the operation's parameters at the time the operation terminates, assuming that the conditions in the requires clause were met at the time of the call to the operation

internal contracts – these are the contracts that augment an operation's or a module's implementation code, e.g., loop invariants a loop and correspondence and module level invariant for a module

loop invariant – is an internal contract that accompanies a loop which captures what is not changed by the loop

correspondence – is an internal contract that maps the values a module's internal variables to an abstract value for the module. For example, a Queue implemented using a linked list. The correspondence captures how the values stored in the linked list is mapped to the queue's abstract value.

module level invariant – aka representation invariant, is an internal contract that specifies constraints on a module's private internal variables. For example, if a Queue is implemented on top of a List, then a module level invariant might constrain that all values remain in the List's right part. Each public method from Queue can depend on this module level invariant holding at the time it is called and each public method must guarantee it holds upon termination.

defensive programming – the design of an operation with a non-trivial requires clause so that the operation's implementation checks to make sure that requirements of listed in the requires clause are satisfied at that time of the call to the operation

defensive do-nothing operation – a defensive operation that performs its normal computation if the incoming parameter values are steady-state values but performs no action if the incoming parameters are non-steady-state values