CSCE 742 - Practice Midterm
Name:

This is a 75-minute exam. On all essay type questions, you will receive points based on the quality of the answer - not the quantity.

Make an effort to write legibly. Illegible answers will not be graded and awarded 0 points.

There are a total of 8 questions and 100 points available on the test.

Problem 1

How is Software Architecture different from Software Design? How is it the same?

Answer: There are many possible answers here. Some of it depends on your definition of architecture: you can use my definition, the 4+1 IEEE definition, or any of the other definitions from the book or from early lectures.

Here is a reasonable answer:

In the first lecture, architecture was defined as those decisions that support partitioning of a large system into smaller chunks that can be developed independently and which integrate correctly to create a functioning system. Therefore, the architectural decisions (from a functional view) are a subset of the design decisions: those that affect this partitioning. Additionally, architecture focuses on a range of non-functional qualities *that are not traditionally associated with software design (though they are associated with* system *design), such as availability, scalability, and ease of operational support.*

On the other hand, design is a continuum. It can be difficult to sometimes assess whether a design decision is architectural or not, and this is also a matter of perspective. A "design detail" for an enterprise architect may be a critical architectural decision for an application designer. In other words, a decision may not be critical to the success of an enterprise (perhaps the application is not so important from this perspective) but it is of course important to the application's architect.

Design tends to be lower-level, and focused on how we create a solution that realizes the requirements. Architecture tends to form a bridge between the abstract requirements and the detailed design - performing initial decomposition of requirements into functionality and decomposition of the monolithic idea of a system into independent subsystems.

Problem 2

1. What are differences between requirements and goals?
2. List two requirements and two goals for the airport parking example.

1. *Requirements are specific and testable; goals are "fuzzier" and may be unlikely to easily quantify in terms of the expected system behavior. Goals often are higher level than requirements and in this case a single goal can be refined into several requirements. This is not to say goals are unimportant; a consistent theme and presentation to the user can be as important as the functional behavior of the device.*
2. *Architectural requirements would be of the form:*
    a. *Under normal operation, the system shall raise the gate within one second of user confirmation of credit card payment.*
    b. *During a loss-of-communication event with the credit card service, the system shall record credit transactions to deferred transaction log for later processing.*

    c. *Architectural goals:*
        i. *The airport parking system should be usable to people without any experience with computer systems.*
        ii. *The airport parking system should be assembled, when possible, of COTS components to minimize cost and maintenance effort.*

Problem 3

The benefit of viewpoints is that they allow separation of concerns and prevent overwhelming users with information. However, in many cases, the viewpoints are tightly coupled, in that a change in one viewpoint necessitates a change in another viewpoint. Consider the Deployment and Concurrency viewpoints.

1. Give examples of information that is duplicated between the two viewpoints and unique to each viewpoint.
2. Should these two viewpoints be merged? Argue in terms of cohesion and coupling of the information between the two views.

1. *Example: The processes and their interconnections in the system are duplicated between viewpoints. The physical resource containing the process is unique to the deployment view and the synchronization primitives that are used are unique to the concurrency view.*
2. *Sample answer (you can argue otherwise, if your argument is well supported): No, in the general case, I would argue that the viewpoints should not be merged. Although you duplicate portions of the system across the viewpoints, and could perhaps overlay the physical resources on top of the concurrency model (or,*

equivalently, the synchronization primitives on the deployment model) the reason for constructing them is quite different. The systems are coupled via a fairly abstract notion (a process), but each has significant internal cohesion because of the concerns that they address. For deployment, it is the physical resources involved, which is the primary concern of the operations and support staff. For concurrency, it is (for the most part) logical issues related to concurrency such as race conditions and deadlock, which are issues to be addressed by the development and testing team.

Problem 4

Are the following business principles, technology principles for a specific system, or low-level object-oriented design advice? Explain why you chose each answer.

1. All external access to the system should be via two factor authentication, involving something you have and something you know, e.g. smart card and password
2. Security measures should be applied appropriately to the level of risk defined for a given system or database.
3. Encapsulate actions into objects that can be stored, replayed, or undone.
4. Adopt an appropriate level locking strategy, typically using optimistic locking for frequently changed data and pessimistic for data where there is a low rate of change.
5. Minimize the number of security interactions needed for the web store.

Answer: I would argue that 2,5 are business principles, 1,4 are technology principles, and 3 is a design pattern, due to the scope of the concerns. I would justify this as follows:
For (1) at an enterprise level, I would not expect to require two-level authentication for all data,
For (2), this is a broad principle that requires further elaboration for a particular application, but defines an enterprise-wide "good idea".
Case (3) is pretty clearly a design pattern (Command). I would argue that it does not rise to the level of an architectural principle.
Case (4) could be argued either as a technology principle or general advice. I would lean towards this principle being an application-level principle because it describes a policy on relational databases, which may be irrelevant for most applications. However, one could view it as a policy over all applications that require a relational database.
Case (5) could be argued as either a business or technology principle. If you view the web store as an application, then it is a technology principle; if you view the web store as a network of connected services, then it is a business level principle.

Problem 5

Making servers stateless (in terms of session state), like in the REST style, has many benefits for scalable and reliable client/server systems. Please describe at least three *drawbacks* of building stateless systems. Give two examples of systems where constructing stateful servers is a good idea.

Significantly more complex programming model!
Can make application much slower to reconstruct state.
May require significant rewrite of existing systems (expense).
May require substantially more server resources to host due to state reconstruction.

Applications where initialization and caching requires significant amount of state (medical imaging, CAD/CAM, many others)
Applications where it would be cost-prohibitive to rewrite (SOA)

Problem 6

In class and in the Garlan and Shaw paper, we considered architectural styles, including:

a. pipe and filter
b. event-based
c. layered
d. repository

Suppose that you are to design an automotive system whose subsystems (a-h) are enumerated below. For each style discussed above, describe a subsystem where the style would be an appropriate structuring mechanism - and why - or describe why this style does not apply to any of the subsystems.

a. On-star communications: manages communications with satellite
b. sensor management: turns noisy sensor data into useful information
c. motion control: operates the motors and provides position and velocity
d. Image processing system to identify highway lanes
e. UX vehicle management involving touch screen
f. Health/status monitoring: checks status of all other subsystems to ensure correct operation
g. Collision avoidance system
h. Dashboard displays

pipe-and-filter: sensor management; there are several de-noising and sensor fusion transforms that are straightforward to describe using pipe-and-filter architectural styles. Similar arguments could be made for communications with filters for compression / encryption.

event-based: Error handling is often performed using an event-based approach, where the events trigger a reset in affected subsystems to clear out faulty data. Supervisory control is often event based; when an objective is reached, send an event to replan.

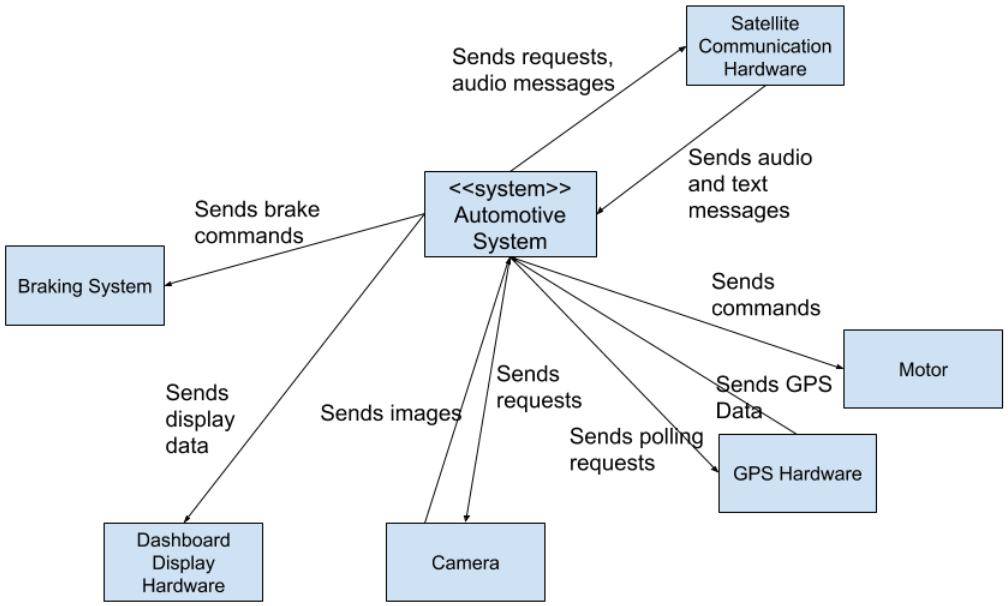
layered: there is a natural layering between supervisory control, navigation control, and motion control. Communications systems themselves are often layered as well (see TCP runs over IP which runs over physical comms)

repository: Health and status monitoring is often performed using a repository architecture, where vehicle health from many systems is aggregated into one place. Also could be used as a data-plane underlying many of the systems mentioned here.

Problem 7

Create a context diagram for the automotive example in the previous question.

[Draw it out: control system in the middle; several sensors and actuators (GPS system, camera, etc.) for controlling the car (motors, braking)]



Problem 8

Consider the software for air-traffic control at an airport (say, CAE). Identify one performance, one availability, and one usability requirement that you think would be necessary for this software and develop a quality attribute scenario for each.

[7 points per example. Requirements should be specific and testable. Scenarios should have single stimuli and specific, measureable system responses]

Glossary:
Normal load is defined as Deployment environment working correctly with less than 500 tracked aircraft.

Performance Requirement: *Under normal load, displayed aircraft positions shall be updated at least every 50 ms.*

Performance Scenario: *responsiveness*
Overview: Check system responsiveness for displaying aircraft positions

System and environment state: Normal load is defined as Deployment environment working correctly with less than 500 tracked aircraft.

External stimulus: 50 Hz dispatch of ATC system.

System response: radar/ADSB sensor values are computed and fused, new position is displayed to the air traffic controller with maximum error of 5 meters.

Response measure: Fusion and display process completes in less than 50 ms.

Availability Requirement: *The system shall be able to tolerate the failure of any single server host, graphics card, display or network link.*

Availability Scenario: *primary display card fails during screen refresh*

Overview: One of the monitor display cards fails during transmission of a screen refresh

System and environment state: System is working correctly under normal load with no failures.

External stimulus: display card fails

Required system response: display window manager system will detect failure within 10 ms and route display information through “hot spare” redundant graphics card with no user-discernable change to ATC aircraft display. Graphics card failure will be displayed as error message at bottom right hand of ATC display.

Response measure: no loss in continuity of visual display and failover with visual warning completes within 1 s.

Usability Requirement: *The user interface shall be designed to minimize user mistakes by providing a route projection and requiring confirmation of decisions. Users shall make fewer than two mistakes per month.*

Usability Scenario: *Routing Mistakes*

Overview: A user is tasked with making a routing decision.

System and environment state: System is working correctly under normal load with no failures.

External stimulus: A pilot indicates readiness to take off, and the user selects an initial trajectory.

Required system response: Display window manager shall show a projection for that trajectory, and ask the user to confirm their decision.

Response measure: The user shall select and confirm a route that could lead to a potential collision fewer than two times per month.

Security Requirement: *The system shall maintain audit logs of any logins to the ATC database.*

Security Scenario: *malicious login*

Overview: a malicious agent gains access to flight records database in the ATC.

System and environment state: System is working correctly under normal load.

External stimulus: malicious agent obtains access to the flight records database through password cracking, and downloads flight plans for commercial aircraft.

Required system response: audit log contains login and download information to support future prosecution of user.

Response measure: system audit contains time, IP address, and related information w.r.t. download.