Final Review

CSCE 742 - Lecture 24 - 12/06/2018

The Final...

- 150 minutes: December 11, 9:00 - 11:30
- In this room!
- Closed book, no notes!
- Graded on quality of answers, not how much you wrote.
- Very similar in format to the practice final and midterm.
- Not cumulative!
- Study your homework.

The Final

- Topics:
    - Concurrency Viewpoint
    - Development Viewpoint
    - Deployment Viewpoint
    - Operational Viewpoint
    - Performance and Scalability Perspective
    - Security Perspective
    - Availability Perspective
    - Real-time systems
    - Architectural Description Languages
    - Services and Service-Oriented Architecture
        - REST/SOAP
    - ML Systems

General Questions

- Today: Go over practice final questions.
- First - any general questions on course content or homework?

Question 1

Speculate as to why none of the architecture description languages other than UML have achieved widespread use.

Question 1

Speculate as to why none of the architecture description languages other than UML have achieved widespread use.

Many possible answers here. Some starting points:

- Difficult to create critical mass to attract tool builders.
- Most notations only capture a small portion of architectural concerns.
- Lack of communication between “academic” software engineering and commercial SE.
- Too domain specific; have not evolved to describe new kinds of architectures (e.g. web services).
- Too hard to use / not understandable to developers.

Question 2

Create an attack tree describing how an attacker might attempt to steal money from an Automated Teller Machine (ATM).

Attack Trees

- Structured notation for categorizing threats and their probability.
  - Represented visually as a tree as a nested list.
  - Root of the tree shows the goal of the attack.
  - Branches classify the different types of attacks that could be attempted.
  - Create a tree for each goal an attacker may have. Can be used to analyze security policies.

Goal: Obtain customer credit card details.

1. Extract details from database.

   1.1 Access database directly.

      1. Crack/guess database passwords.
      2. Crack/guess OS passwords that bypass database security.

   1.2 Access via a member of the administration staff.

      1. Bribe a database administrator (DBA).
      2. Conduct social engineering by e-mail to trick the DBA into revealing details

Attack Tree Example

2. Extract details from Web interface.
    2.1. Set up a dummy Web site and e-mail users the URL to trick them into entering credit card details.
    2.2. Crack/guess passwords for user accounts and extract details from the GUI.
    2.3. Send users a program by e-mail to record keystrokes.
    2.4. Attack the domain name server to hijack domain name and attack 2.1.
    2.5. Attack the server software directly to try to find loopholes in its security.
3. Find details outside the system.
    3.1. Conduct social engineering by phone/e-mail to get customer services staff to reveal card details.
    3.2. Direct a social-engineering attack on users by using public details from the site to make contact.

Question 2

Create an attack tree describing how an attacker might attempt to steal money from an Automated Teller Machine (ATM).

Goal: Steal money from ATM

1. Physical attack
    a. Break ATM casing and steal money
        i. [AND]
            1. Steal entire ATM for later dismantling
            2. Procure vehicle capable of transporting ATM
    b. Card-data stealing attack
        i. Card-based attack using new cards
            1. [AND]
                a. Buy/steal card producing device
                b. Buy/steal card stock for new ATM cards
        ii. Data capture for ATM spoofing
            1. Capture ATM track 1&2 data and valid PIN using a skimming device
            2. Buy data and PINs from black market
2. Capture/guess ATM data from online banking site
3. Capture/guess ATM data through ATM software vulnerability
4. Get valid card in someone else's name (id theft)
5. ...

Question 3

What difficulties do cyclic component dependencies lead to in an architecture?

What can be done to break cyclic dependencies?

Question 3

What difficulties do cyclic component dependencies lead to in an architecture?

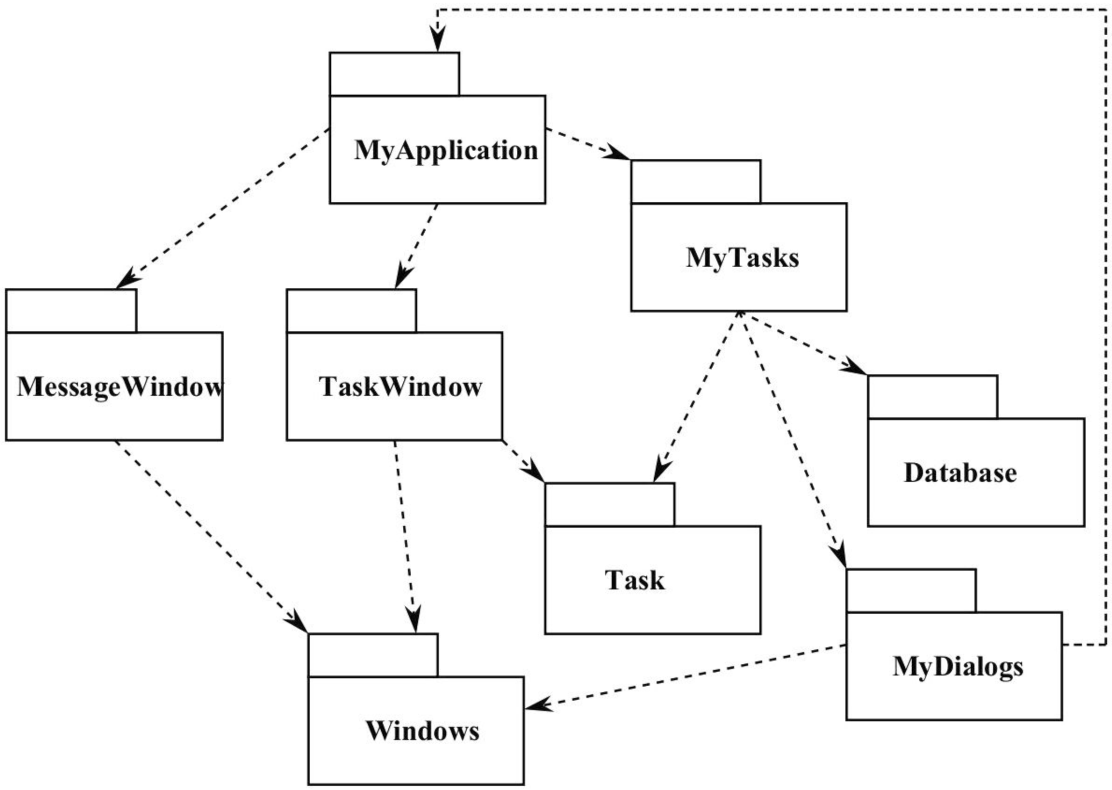
- Components that contain circular dependencies are much more difficult to test and maintain.
  - They are tightly coupled to the interface of the others.
  - For testing, a usual strategy is to use “leveling” to test.
    - Test components that have no dependencies first, then once you have confidence in them, test components that depend only on those components.
  - If you have circular dependencies, it is much more difficult to test in this way. We have to examine the behavior of all of these components simultaneously.

Question 3

What difficulties do cyclic component dependencies lead to in an architecture?

- For maintenance, it is often difficult to modify one of the cyclically connected components without changing all of the components in the cycle.
  - This can be problematic if the components containing the cyclic references reside in multiple packages.
  - For versioning, if components mutually depend then they must be installed and updated simultaneously.
- The same thing is true for compilation. A change in one triggers compilation of all others.

Cyclic Example (Bad)



Release:

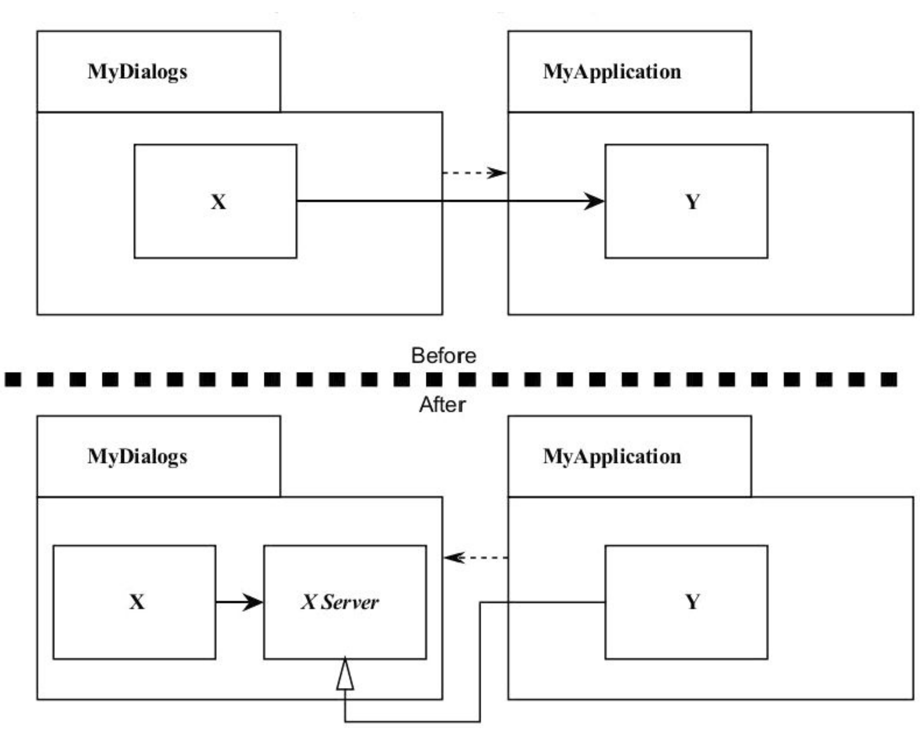
- Must be simultaneous with MyApplication.
- But this means MyTasks must also be coordinated (it is a dependency of MyApplication and depends on MyDialogs).
- This means that it must also be coordinated with Task and Database (dependencies of MyTasks).

Testing:

- MyDialogs requires MyApplication, so…
- **MyDialogs is dependent on all packages(!) for testing!**

Fixing Circular Dependencies

- Apply the Dependency Inversion Principle.
    - Create an abstract class with the interface MyDialogs needs.
    - Put the class into MyDialogs
    - Inherit into MyApplication.
    - Reverses the dependency, breaking the cycle.



Package Refactoring

- Can also create third package with class(es) that both MyApplication and MyDialog depend on.
- Package contents and dependency hierarchy must be actively managed and refactored.
  - Dependencies will change as system expands.
  - Circular dependencies must be pruned out.
  - Coordinating this movement is important job for architect.

Question 4

RPCs and messaging schemes are inter-process communication mechanisms. For one process with multiple threads:

1. **Are there analogous concepts to RPCs and messaging between threads?**
2. **Describe an additional means of communication that is available between threads.**
3. **Do the same benefits/drawbacks between RPCs and messaging schemes exist when considering inter-thread communication as interprocess communication?**

Question 4

Are there analogous concepts to RPCs and messaging between threads?

- Messaging:
  - Constructed from semaphores (limits number of consumers for a resource).
  - Threads communicate through event queues (send event to queue).
  - Delivery is guaranteed, as we act within one process.
- RPC:
  - No immediately analogous concept.
  - Local procedure calls are similar, but not a means of cross-thread communication.

Question 4

Describe an additional means of communication that is available between threads.

- Shared memory
  - Threads can read and write from the same memory space. Form of message passing, indirectly causes changes to occur in other threads.
- Monitors
  - In Java, objects have a monitor that ensures that only one thread can execute a critical section of code at a time.
    - **wait()** tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.
    - **notify()** wakes up a thread that called wait() on same object.
    - **notifyAll()** wakes up all the threads that called wait() on same object.

Question 4

Describe an additional means of communication that is available between threads.

- Monitors
  - wait() tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.
  - notify() wakes up a thread that called wait() on same object.
  - notifyAll() wakes up all the threads that called wait() on same object.
    - Threads enter to acquire lock.
    - Lock is acquired by one thread.
    - Now, thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
    - If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
    - Now thread is available to acquire lock.
    - After completion of the task, thread releases the lock and exits the monitor state of the object.

Question 4

Describe an additional means of communication that is available between threads.

- Semaphores
  - Counter that acts as a permit for a shared resource.
  - When a thread gets a permit, the counter goes down.
    - Otherwise, the thread will be blocked until it can get a permit.
  - When finished, the thread releases the semaphore
    - (counter increments).
  - Semaphores can be used to control synchronization between threads.

Question 4

Do the same benefits/drawbacks between RPCs and messaging schemes exist when considering inter-thread communication as interprocess communication?

- RPCs may not be useful between threads. They share an address space. Why not just use the active thread?
  - No performance benefit since RPCs are synchronous.
- Messaging is still useful.
  - Allows decoupling of processes from UI, etc.
  - Allows a clean interface between threads, preventing deadlock/race conditions.
  - No guarantee of persistence (like inter-process communication), but no process lifecycle issues.

Question 4

Do the same benefits/drawbacks between RPCs and messaging schemes exist when considering inter-thread communication as interprocess communication?

- Performance:
  - Threads communicate faster than processes, so lower cost of splitting tasks.
- Isolation:
  - Threads are not isolated (like processes), so some benefits of IPC do not impact inter-thread communication.
- IPCs allow processes to be spread across multiple machines. ITC does not.
  - No scalability benefit.

Question 5

What are the benefits and drawbacks of using XML vs. binary protocols for messaging between processes?

Question 5

What are the benefits and drawbacks of using XML vs. binary protocols for messaging between processes?

- Custom binary protocol may require less network bandwidth than XML.
- If processes can agree on format, binary protocol may require less translation and parsing at network boundaries.
    - For applications that are network intensive, this may lead to better performance and network utilization.

Question 5

What are the benefits and drawbacks of using XML vs. binary protocols for messaging between processes?

- XML will be easier to maintain. Can use standard tools to process message traffic and diagnose errors.
- XML supports transparent re-hosting on different platforms.
- XML may lead to better performance given static data. Can perform pre-built caching.
- Can more easily add additional data to XML messages without disrupting existing clients.
- Several tools for manipulating and routing XML messages, making it easier to allow new applications into a system.

Question 6

Big Bang, Parallel Run, and Staged Migration are techniques for upgrading existing software installations.

1. **Briefly explain each technique.**
2. **Describe advantages and disadvantages for each technique in terms of data migration, complexity, and rollback in case of failure.**
3. **Describe installation scenarios where you might use each technique.**

Question 6

Briefly explain each technique.

- Big Bang: Pick a day, turn the old system off and the new system on.
- Parallel Run: Choose a period in which the old and new systems run in parallel.
- Staged Migration: Swap out pieces of the old system with pieces of the new system over an installation period or migrate portions of the organization over a period.

Question 6

Describe advantages and disadvantages for each technique in terms of data migration, complexity, and rollback in case of failure.

- Big Bang:
    - Data migrates in one shot.
    - Simple in the sense that only one system runs at a time. Hard if systems do not have downtime.
        - How do you do “immediate switchover”?
    - Hard to recover from failure, may require reverse data migration.

Question 6

Describe advantages and disadvantages for each technique in terms of data migration, complexity, and rollback in case of failure.

- Parallel Run:
  - Data migration happens in parallel with older system running (if new system fails on certain queries, route to old system).
    - Less risk in data migration. Need to have support for continuous migration as data is added to new system.
  - More complex than big bang.
    - Policies needed for synchronizing and routing traffic between old/new system.
    - How is the new system validated?
    - Need to maintain redundant copies of data.
  - Rollback can be run in lockstep.
    - Can also run split-stream traffic, then reverse data migration may be required.

Question 6

Describe advantages and disadvantages for each technique in terms of data migration, complexity, and rollback in case of failure.

- Staged:
  - Data migration needs facades to allow new system components to work with old system components.
    - These facades translate data back and forth.
  - Since you need facades, there is some complexity.
  - If migrating parts of organization, you may also need policies for handling discrepancies between the new/old system.

Question 6

Describe installation scenarios where you might use each technique.

- Big bang: Home web server upgrade
  - Just move everything over and be done.
  - Only affects you.
- Parallel run: Banking transaction software
  - Can test new system with a subset of users.
  - If any issues, use the existing reliable system.
- Staged migration: Air traffic control
  - Hard to keep old and new synchronized, but want to control release of new system to ensure each component is reliable.

Question 7

What are the availability benefits and risks associated with the following architectural styles: Pipe and Filter, Repository, Event-based, Layered.

Question 7

What are the availability benefits and risks associated with the following architectural styles: Pipe and Filter.

- Can duplicate streams of data to multiple CPUs transparently (from filter perspective).
  - If failure occurs, can redirect to a good stream.
- Can easily introduce “voters” to look for disagreements in results from multiple filters.
- Risks: On a single pipe & filter chain, any single failure will cause whole system to fail because filters do not know about each other.

Question 7

What are the availability benefits and risks associated with the following architectural styles: Repository.

- Risky for availability.
  - Central store, so consistency problems in presence of multiple readers/writers.
  - Central point of failure.
- Consolidates critical data in a single location, so single-point logging & recovery.
- Many schemes for high-availability repositories (database clusters).
  - Expensive, but work well in practice.

Question 7

What are the availability benefits and risks associated with the following architectural styles: Event-based.

- If there is a centralized event broker, can be risky for availability (and performance) because it introduces a central point of failure.
- Also, can be difficult to understand the composite behavior of event-based systems.
  - “Event storms” can occur if one event leads to a cascade of many events that can reduce system reliability and availability.
- On the other hand, failover mechanisms when constructing highly-available systems are event-based.
  - A heartbeat event, sent at regular intervals, is the means by which system health is monitored. If a sibling system does not send a heartbeat, then failover is performed.

Question 7

What are the availability benefits and risks associated with the following architectural styles: Layered.

- Layers help create highly available systems because they limit the kinds of failures that must be accounted for.
    - Web is reliable because of isolated handling of classes of failure by different layers. IP handles routing, TCP handles packet retransmission, load balancing handles server failures, etc.

Question 8

Write the guarantees for the following AADL component describing a dishwasher mode controller:

```
system Dishwasher_Mode
     features
          door_closed: in data port Base_Types::Boolean;
          time_remaining: in data port Base_Types::Integer;
          pump_on: out data port Base_Types::Boolean;
          dishwasher_mode: out data port Base_Types::Integer;

     annex agree {**
          const SETUP_MODE : int = 0;
          const WASHING_MODE : int = 1;
          const RINSE_MODE : int = 2;
          const DRYING_MODE : int = 3;

          guarantee "the pump shall be off if the door is open" : true;
          guarantee "If the dishwasher was in WASHING_MODE and time
remaining is zero, it shall enter RINSE_MODE" : true;
          guarantee "The dishwasher shall never transition directly from
WASHING_MODE to DRYING_MODE" : true;
          guarantee "The dishwasher shall start in SETUP_MODE" : true;
     **};
end Dishwasher_Mode;
```

Question 8

Write the guarantees for the following AADL component describing a dishwasher mode controller:

```
system Dishwasher_Mode
    features
        door_closed: in data port Base_Types::Boolean;
        time_remaining: in data port Base_Types::Integer;
        pump_on: out data port Base_Types::Boolean;
        dishwasher_mode: out data port Base_Types::Integer;

    annex agree {**
        const SETUP_MODE : int = 0;
        const WASHING_MODE : int = 1;
        const RINSE_MODE : int = 2;
        const DRYING_MODE : int = 3;

        guarantee "the pump shall be off if the door is open" :
        pump_on => door_closed;
        (alternate): (not door_closed) => (not pump_on);
**};
end Dishwasher_Mode;
```

Question 8

Write the guarantees for the following AADL component describing a dishwasher mode controller:

```
system Dishwasher_Mode
    features
        door_closed: in data port Base_Types::Boolean;
        time_remaining: in data port Base_Types::Integer;
        pump_on: out data port Base_Types::Boolean;
        dishwasher_mode: out data port Base_Types::Integer;

    annex agree {**
        const SETUP_MODE : int = 0;
        const WASHING_MODE : int = 1;
        const RINSE_MODE : int = 2;
        const DRYING_MODE : int = 3;

        guarantee "If the dishwasher was in WASHING_MODE and
time remaining is zero, it shall enter RINSE_MODE" :
```

true -> ((pre(dishwasher_mode) = WASHING_MODE and time_remaining = 0) => dishwasher_mode = RINSE_MODE)

```
    **};
end Dishwasher_Mode;
```

Question 8

Write the guarantees for the following AADL component describing a dishwasher mode controller:

```
system Dishwasher_Mode
    features
        door_closed: in data port Base_Types::Boolean;
        time_remaining: in data port Base_Types::Integer;
        pump_on: out data port Base_Types::Boolean;
        dishwasher_mode: out data port Base_Types::Integer;

    annex agree {**
        const SETUP_MODE : int = 0;
        const WASHING_MODE : int = 1;
        const RINSE_MODE : int = 2;
        const DRYING_MODE : int = 3;

        guarantee "The dishwasher shall never transition
directly from WASHING_MODE to DRYING_MODE" :
```

true -> (pre(dishwasher_mode) = WASHING_MODE => (not dishwasher_mode = DRYING_MODE))

```
    **};
end Dishwasher_Mode;
```

Question 8

Write the guarantees for the following AADL component describing a dishwasher mode controller:

```
system Dishwasher_Mode
    features
        door_closed: in data port Base_Types::Boolean;
        time_remaining: in data port Base_Types::Integer;
        pump_on: out data port Base_Types::Boolean;
        dishwasher_mode: out data port Base_Types::Integer;

    annex agree {**
        const SETUP_MODE : int = 0;
        const WASHING_MODE : int = 1;
        const RINSE_MODE : int = 2;
        const DRYING_MODE : int = 3;

        guarantee "The dishwasher shall start in SETUP_MODE" :
        dishwasher_mode = SETUP_MODE -> true.
    **};
end Dishwasher_Mode;
```

Question 9

- **What is the difference between response time and throughput?**
- **Give an example of a system with excellent throughput but poor response time and vice versa.**

Question 9

What is the difference between response time and throughput?

- Response time is from the client’s perspective.
  - How long does it take to service my request?
- Throughput is from the server’s perspective.
  - How many requests can be processed in a given time period?

Question 9

Give an example of a system with excellent throughput but poor response time and vice versa.

- A pipelined system may involve several processors working in tandem to solve a particular problem.
  - It may be able to process very large volumes of transactions (high throughput) due to partitioning the problem into segments that are handled sequentially, while still exhibiting poor response time.
    - (each segment takes time t, with number of segments s, so the total response time is n*s).
- Instead, imagine a single processor non-pipelined system that processes requests sequentially.
  - If there are few requests, it will have better response time than the pipelined system because there is no latency in servicing the request.
  - However, it will have very poor throughput under heavy load.

Question 10

What is the distinguishing characteristic of a *real time,* as opposed to a *non-real time* system? What is the difference between hard and soft real time systems?

Question 10

What is the distinguishing characteristic of a *real time,* as opposed to a *non-real time* system? What is the difference between hard and soft real time systems?

- For a real-time system, an operation's correctness depends not only on logical correctness, but the time required to complete it.
- In a hard real-time system, computation of an answer after its deadline is considered failure.
- Soft real time systems can tolerate missed deadlines as long as there is a bound on the number of missed deadlines within some time scale.

Thanks for a great semester!!!!!!!!!!!!!!!!!!!!

Next Time

- The Final
  - Please e-mail questions + office hours today.
  - Exam will be proctored by a student, so ask me questions before the day of the test.

- Homework:
  - Project, Part 4 - Due tonight
  - Assignment 3 - Due on December 9