BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Architectures

SECLZG651/SSCLZG653

Parthasarathy

SECLZG651/SSCLZG653 – CS#3
Quality Attributes

Agenda for CS #3

1) Recap of CS1-2
2) Getting started with Quality attributes
3) Usability and its tactics
4) Availability and its tactics
5) Modifiability and its tactics
5) Q&A!

Usability

- How easy it is for the user to accomplish a desired task and user support the system provides
  - Learnability: what does the system do to make a user familiar
  - Operability:
    - Minimizing the impact of user errors
    - Adopting to user needs
    - Giving confidence to the user that the correct action is being taken?

Usability Scenario Example

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
End user	User Wants to • Learn system feature • Use systems efficiently • Minimize the impact of errors • Adapt system • Feel comfortable	**Whole System** **•At run time** **•At configure time**	• Learn ✓ Context sensitive help, familiar interface • Efficient use ✓ Aggregation of data and command, reuse of already entered data, good navigation, search mechanism, multiple activities • Error impact ✓ Undo, cancel, recover, auto-correct, retrieve forgotten information	• Task time • Number of errors • User satisfaction • Gain of user knowledge • Successful operations • Amount of time/data lost
End User	Downloads application	Runtime	Uses application productively	Takes 4 mins to be productive

Usability Tactics

Usability is essentially Human Computer Interaction. Runtime Tactics are

User initiative (and system responds)

- Cancel, undo, aggregation, store partial result

System initiative

- Task model: understands the context of the task user is trying and provide assistance
- User model: understands who the user is and takes action
- System model: gets the current state of the system and responds

User Initiative and System Response

Cancel

- When the user issues cancel, the system must listen to it (in a separate thread)
- Cancel action must clean the memory, release other resources and send cancel command to the collaborating components

Undo

- System needs to maintain a history of earlier states which can be restored
- This information can be stored as snapshots

Pause/resume

- Should implement the mechanism to temporarily stop a running activity, take its snapshot and then release the resource for other's use

Aggregate (change font of the entire paragraph)

- For an operation to be applied to a large number of objects
    - Provide facility to group these objects and apply the operation to the group

System Initiated

Task model

- Determine the current runtime context, guess what user is attempting, and then help
- Correct spelling during typing but not during password entry

System model

- Maintains its own model and provide feedback of some internal activities
- Time needed to complete the current activity

User model

- Captures user's knowledge of the system, behavioral pattern and provide help
- Adjust scrolling speed, user specific customization, locale specific adjustment

Usability Tactics and Patterns….

Design time tactics- UI is often revised during testing. It is best to separate UI from the rest of the application

- Model view controller architecture pattern
- Presentation abstraction control
- Command Pattern
- Arch/Slinky
  - Similar to Model view controller

Design Checklist

Allocation of Responsibilities

- Identify the modules/components responsible for
  - Providing assistance, on-line help
  - Adapt and configure based on user choice
  - Recover from user error

Coordination Model

- Check if the system needs to respond to
  - User actions (mouse movement) and give feedback
  - Can long running events be canceled?

Data model

data structures needed for undo, cancel

Design of transaction granularity to support undo and cancel

Resource mgmt

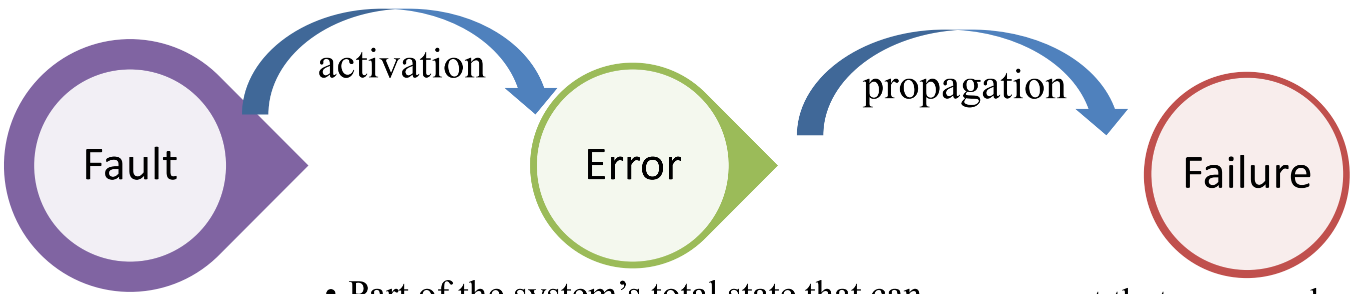
Design how user can configure system's use of resource

Technology selection

To achieve usability

Availability and its Tactics

Faults and Failure



- Hypothesized cause of error in the software
- Part of the system's total state that can leads to failure
- event that occurs when the delivered service deviates from correct service

Not every fault causes a failure:

- Code that is "mostly" correct.
- Dead or infrequently-used code.
- Faults that depend on a set of circumstances to occur

Cost of software failure often far outstrips the cost of the original system

- data loss
- down-time
- cost to fix

Primary objective: Remove faults with the most serious consequences.

Secondary objective: Remove faults that are encountered most often by users.

- One study showed that removing 60% of software "defects" led to a 3% reliability improvement

Failure Classification

- Transient - only occurs with certain inputs
- Permanent - occurs on all inputs
- Recoverable - system can recover without operator help
- Unrecoverable - operator has to help
- Non-corrupting - failure does not corrupt system state or data
- Corrupting - system state or data are altered

Availability

- Readiness of the software to carry out its task
  - 100% available (which is actually impossible) means it is always ready to perform the intended task
- A related concept is Reliability
  - Ability to "continuously provide" correct service without failure
- Availability vs Reliability
  - A software is said to be available even when it fails but recovers immediately
  - Such a software will NOT be called Reliable
- Thus, Availability measures the fraction of time system is really available for use
  - Takes repair and restart times into account
  - Relevant for non-stop continuously running systems (e.g. traffic signal)

What is Software Reliability

- Probability of failure-free operation of a system over a specified time within a specified environment for a specified purpose
  - Difficult to measure the purpose,
  - Difficult to measure environmental factors.
- It’s not enough to consider simple failure rate:
  - Not all failures are created equal; some have much more serious consequences.
  - Might be able to recover from some failures reasonably.

Availability

- Once the system fails
  - It is not available
  - It needs to recover within a short time
- Availability $A = \frac{MTTF}{MTTF + MTTR}$
- Scheduled downtime is typically not considered
  - Availability 100% means it recovers instantaneously
  - Availability 99.9% means there is 0.1% probability that it will not be operational when needed

System Type	Availability (%)	Downtime in a year
Normal workstation	99	3.6 days
HA system	99.9	8.5 hours
Fault-resilient system	99.99	1 hour
Fault-tolerant system	99.999	5 min

Availability Scenarios

innovate achieve lead

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
Internal or External to System	Fault causing ➢ System does not respond ➢ Crash ➢ Delay in response ➢ Errorneous Response	Infrastructure and/or application •During normal operation •During degraded mode of operation	When fault occurs it should do one or more of ✓ detect and log ✓ Notify the relevant stakeholders ✓ Disable the source of failure ✓ Be unavailable for a predefined time interval ✓ Continue to operate in a degraded mode	• Specific time interval for availability • Availability number • Time interval when it runs in degraded mode • Time to repair

Two Broad Approaches

- Fault Tolerance
  - Allow the system to continue in presence of faults. Methods are
    - Error Detection
    - Error Masking (through redundancy)
    - Recovery
- Fault Prevention
  - Techniques to avoid the faults to occur

Availability Tactics

Fault detection	Error Masking	Recover From Fault	Fault prevention
➢Ping/echo ➢Heartbeat ➢Timestamp ➢Data sanity check ➢Condition monitoring ➢Voting ➢Exception Detection ➢Self-test	➢Active redundancy (Hot) ➢Passive redundancy (Warm) ➢Spare (Cold) ➢Exception handling ➢Graceful degradation ➢Ignore faulty behavior	➢Rollback ➢Retry ➢Reconfiguration ➢Shadow operation ➢State resynchronization ➢Escalating restart ➢Nonstop forwarding	➢Removal of a component to prevent anticipated failure– auto/manual reboot ➢Create transaction ➢Software upgrade ➢Predictive model ➢Process monitor- that can detect, remove and restart faulty process ➢Exception prevention

Availability Tactics- Fault Detection

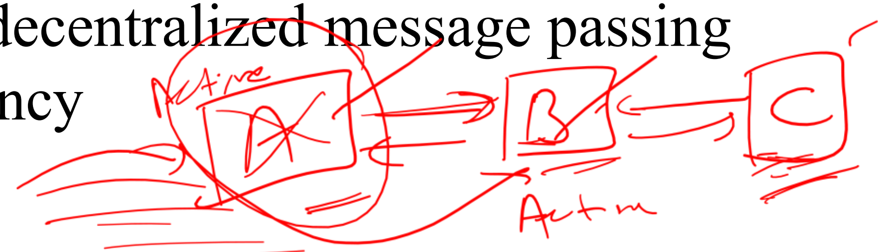
- Ping
  - Client (or fault-detector) pings the server and gets response back
  - To avoid less communication bandwidth- use hierarchy of fault-detectors, the lowest one shares the same h/w as the server
- Heartbeat
  - Server periodically sends a signal
  - Listeners listen for such heartbeat. Failure of heartbeat means that the server is dead
  - Signal can have data (ATM sending the last txn)
- Exception Detection
  - Adding an Exception handler means error masking

More details- Heartbeat

- Each node implements a lightweight process called heartbeat daemon that periodically (say 10 sec) sends heartbeat message to the master node.
- If master receives heartbeat from a node from both connections (a node is connected redundantly for fault-tolerance), everything is ok
- If it gets from one connections, it reports that one of the network connection is faulty
- If it does not get any heartbeat, it reports that the node is dead (assuming that the master gets heartbeat from other nodes)
- Trick: Often heartbeat signal has a payload (say resource utilization info of that node)
  - Hadoop NameNode uses this trick to understand the progress of the job

Detect Fault

- Timer and Timestamping
  - If the running process does not reset the timer periodically, the timer triggers off and announces failure
  - Timestamping: assigns a timestamp (can be a count, based on the local clock) with a message in a decentralized message passing system. Used to detect inconsistency



- Voting (TMR)
  - Three identical copies of a module are connected to a voting system which compares outputs from all the three components. If there is an inconsistency in their outputs when subjected to the same input, the voting system reports error/inconsistency
  - Majority voting, or preferred component wins

HA|DR

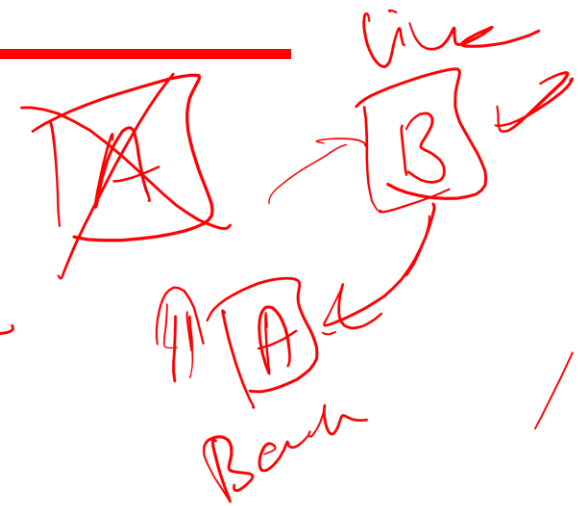
Availability Tactics- Error Masking

- Hot spare (Active redundancy)
  - Every redundant process is active
  - When one fails, another one is taken up
  - Downtime is millisec
- Warm restart (Passive redundancy)
  - Standbys keep syncing their states with the primary one
  - When primary fails, backup starts
- Spare copy (Cold)
  - Spares are offline till the primary fails, then it is restarted
  - Typically restarts to the checkpointed position
  - Downtime in minute
  - Used when the MTTF is high and HA is not that critical

Error Masking

- Service Degradation
  - Most critical components are kept live and less critical component functionality is dropped
- Ignore faulty behavior
  - E.g. If the component send spurious messages or is under DOS attack, ignore output from this component
- Exception Handling – this masks or even can correct the error

Availability Tactics- Fault Recovery



- Shadow
  - ~~Repair the component~~
  - Run in shadow mode to observe the behavior
  - Once it performs correctly, reintroduce it
- State resynch
  - Related to the hot and warm restart
  - When the faulty component is started, its state must be upgraded to the latest state.
    - Update depends on downtime allowed, size of the state, number of messages required for the update..
- Checkpointing and recovery
  - Application periodically "commits" its state and puts a checkpoint
  - Recovery routines can either roll-forward or roll-back the failed component to a checkpoint when it recovers

Availability Tactics- Recovery

- Escalating Restart
  - Allows system to restart at various levels of granularity
    - Kill threads and recreate child processes
    - Frees and reinitialize memory locations
    - Hard restart of the software
- Nonstop forwarding (used in router design)
  - If the main recipient fails, the alternate routers keep receiving the packets
  - When the main recipient comes up, it rebuilds its own state

Availability Tactics- Fault Prevention

- Faulty component removal
  - Fault detector predicts the imminent failure based on process's observable parameters (memory leak)
  - The process can be removed (rebooted) and can be auto-restart
- Transaction
  - Group relevant set of instructions to a transaction
  - Execute a transaction so that either everyone passes or all fails
- Predictive Modeling
  - Analyzes past failure history to build an empirical failure model
  - The model is used to predict upcoming failure
- Software upgrade (preventive maintenance)
  - Periodic upgrade of the software through patching prevents known vulnerabilities

Design Decisions

Responsibility Allocation

- For each service that need to be highly available
  - Assign additional responsibility for fault detection (e.g. crash, data corruption, timing mismatch)
  - Assign responsibilities to perform one or more of:
    - Logging failure, and notification
    - Disable source event when fault occur
    - Implement fault-masking capability
    - Have mechanism to operate on degraded mode

Coordination

- For each service that need to be highly available
  - Ensure that the coordination mechanism can sense the crash, incorrect time
  - Ensure that the coordination mechanism will
    - Log the failure
    - Work in degraded mode

Design Decisions

Data Model

- Identify which data + operations are impacted by a crash, incorrect timing etc.
  - Ensure that these data elements can be isolated when fault occurs
  - E.g. ensure that "write" req. is cached during crash so that during recovery these writes are applied to the system

Resource Management

- Identify which resources should be available to continue operations during fault
- E.g. make the input Q large enough so that can accommodate requests when the server is being recovered from a failure

Design Decisions

Binding Time

- ➢ Check if late binding can be a source of failure
- ➢ Suppose that a late bound component report its failure in 0.1ms after the failure and the recovery takes 1.5sec. This may not be acceptable

Technology Choice

- Determine the technology and tools that can help in fault detection, recovery and then reintroduction
- Determine the technology that can handle a fault
- Determine whether these tools have high availability!!

Hardware vs Software Reliability Metrics

- Hardware metrics are not suitable for software since its metrics are based on notion of component failure
- Software failures are often design failures
- Often the system is available after the failure has occurred
- Hardware components can wear out

Software Reliability Metrics

- Reliability metrics are units of measure for system reliability
- System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure
- A long-term measurement program is required to assess the reliability of critical systems

Time Units

- Raw Execution Time
  - non-stop system
- Calendar Time
  - If the system has regular usage patterns
- Number of Transactions
  - demand type transaction systems

Reliability Metric POFOD

- Probability Of Failure On Demand (POFOD):
  - Likelihood that system will fail when a request is made.
  - E.g., POFOD of 0.001 means that 1 in 1000 requests may result in failure.
- Any failure is important; doesn’t matter how many if the failure > 0
- Relevant for safety-critical systems

Reliability Metric ROCOF & MTTF

- Rate Of Occurrence Of Failure (ROCOF):
  - Frequency of occurrence of failures.
  - E.g., ROCOF of 0.02 means 2 failures are likely in each 100 time units.
- Relevant for transaction processing systems
- Mean Time To Failure (MTTF):
  - Measure of time between failures.
  - E.g., MTTF of 500 means an average of 500 time units passes between failures.
- Relevant for systems with long transactions

Rate of Fault Occurrence

- Reflects rate of failure in the system
- Useful when system has to process a large number of similar requests that are relatively frequent
- Relevant for operating systems and transaction processing systems

Mean Time to Failure

- Measures time between observable system failures
- For stable systems MTTF = 1/ROCOF
- Relevant for systems when individual transactions take lots of processing time (e.g. CAD or WP systems)

Failure Consequences

- When specifying reliability both the number of failures and the consequences of each matter
- Failures with serious consequences are more damaging than those where repair and recovery is straightforward
- In some cases, different reliability specifications may be defined for different failure types

Building Reliability Specification

- For each sub-system analyze consequences of possible system failures
- From system failure analysis partition failure into appropriate classes
- For each class send out the appropriate reliability metric

Failure Class	Example	Metric
Permanent Non-corrupting	ATM fails to operate with any card, must restart to correct	ROCOF = .0001 Time unit = days
Transient Non-corrupting	Magnetic stripe can't be read on undamaged card	POFOD = .0001 Time unit = transactions

Activity [10 min]

Scenario 1:

You're designing a self-service ***online railway booking system*** *for passengers of all ages and digital literacy levels, including first-time users.*

- Which **usability tactics** would you apply in the architecture/design, and why?

Scenario 2:

You're designing a hospital patient-monitoring system that tracks and alerts about vitals in ICU. Downtime is not acceptable.

- Which **availability tactics** would you apply, and why?

Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in