Software Architecture

Lecture 5
Quality Attributes

João Pedro Sousa
George Mason University

previously
architectural styles

- **data flow**
  - batch sequential
  - dataflow network (pipe & filter)
    - acyclic, fan-out, pipeline, Unix
  - closed loop control
- **call-return**
  - main program/subroutines
  - information hiding
    - objects, naive client-server
  - SOA
- **interacting processes**
  - communicating peers
    - asynchronous messages
  - event systems
    - implicit invocation
    - publish-subscribe
- **data-oriented repository**
  - transactional databases
    - true client-server
  - blackboard
  - modern compiler
- **data-sharing**
  - compound documents
  - hypertext
  - Fortran COMMON
  - LW processes
- **hierarchical**
  - tiers
    - interpreter
    - N-tiered client-server

today

- architectural drivers
  - capturing quality attributes
  - tradeoffs
- QA scenarios
  - discovering scenarios: QAW
  - scenario guidelines
    - for availability, modifiability, performance, security
- case study

Acknowledgment
some of the material presented in this course is adapted from 17655, taught to the MSE at CMU by David Garlan and Tony Lattanze

functionality matters
structure matters

- two systems may have different structure and provide the same function
- what else matters?
  - modifiability, performance, and many others
- what are the effects of architectural decisions on quality attributes?
  - each structure promotes different qualities
    - design for modifiability
    - design for security
    - design for performance...

one decision may affect one or more QAs

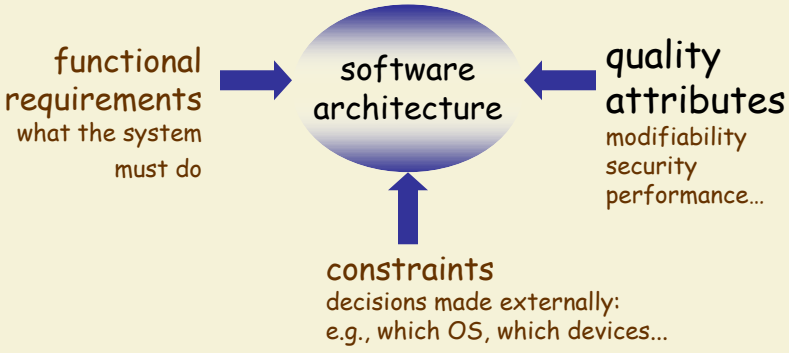
example: usability

- UI decision: choice of radio buttons, dialog boxes, or command line
  - affects usability
- structural decision: encapsulate UI functionality to make it easy to later replace command line by dialog boxes
  - affects modifiability and development cost
  - not usability

often a design decision promotes some QAs, inhibits others

- tradeoffs
  - a change in structure that improves one QA may inhibit or promote the achievement of others
  - a pattern/style that improves one QA may inhibit or promote the achievement of others
- QAs are not independent of each other
  - architecture is critical to balancing *QA tradeoffs* before detailed design, implementation, or investing in system upgrades

architectural drivers
requirements that shape the architecture



- architecture is critical to the realization of QAs
  - you cannot design a system and then go back and add quality

are QAs
non-functional requirements?

- yes, but:
  the term *non-functional* suggests a false partitioning
- QAs cannot be described independently of functionality
- "ility" names are ***not*** enough
  - vocabulary varies widely
    there is no widely accepted standard
  - descriptions are vague and lack quantifiable measures
  - dictionary definitions are superficial
  - debates on what "ilities" really mean are not productive

the system shall be modifiable
wrong way to describe QAs

- you cannot design a system that is modifiable for all potential changes
- every system is modifiable with respect to some set of changes and not with respect to some other
- ask yourself what is really the issue
- the issue is:
  how can we minimize the ***cost*** *of* ***these kinds*** *of changes?*
- be specific:
  - what kinds of changes?
  - how to measure the cost?

right way to describe QAs

- functional reqs are often captured with *Use Cases*
  - why not use *Use Cases* to capture QA requirements?
- *QA Scenarios*
  - quantify QA
  - prioritize QAs
    it's often not possible to have it all
  - make architectural decisions to reach the best possible balance in the architectural drivers
    - may need to tradeoff some QAs by others or functionality by QAs...

QA Workshop
method to discover QA scenarios

- system-centric
- stakeholder focused
- used before the software architecture is designed

1. introductions and QAW presentation
2. business/mission presentation
3. architecture plan presentation
4. identify architectural Drivers
5. scenario brainstorming
6. scenario consolidation
7. scenario prioritization
8. scenario refinement

iterate as necessary with broader stakeholder community

role of QA Workshop

produces

- raw QA scenarios
- prioritization of QA scenarios
- refined QA scenarios

can be used to

- refine requirements
- prioritize development
- identify and mitigate risks
- target prototypes
- design the architecture

QA scenarios
have six parts

1. stimulus
   a condition that affects the system
2. source of the stimulus
   the entity that generated the stimulus
3. environment
   the conditions under which the stimulus occurred
4. artifact stimulated
   the artifact that perceives the stimulus firsthand
5. response
   the activity that results from the stimulus
6. response measure
   the measure by which the system's response will be evaluated

take 5

outline

- architectural drivers
  - capturing quality attributes
  - tradeoffs
- QA scenarios
  - discovering scenarios: QAW
  - scenario guidelines
    - for availability, modifiability, performance, security
- case study

scenario guidelines
availability

- things to consider
  - preventing catastrophic system failures
  - detecting system failures
  - recovering successfully from system failures
  - the amount of time needed to recover from system failures
  - the frequency of system failures
  - degraded modes of operation due to system failures
- example scenario
  - *when an unanticipated external hardware error is received by the initialization process during startup, the initialization process prevents an engine start and illuminates the check engine light*

checklist
availability

stimuli
- omission, crash, timing, events

sources
- internal, external, software, hardware

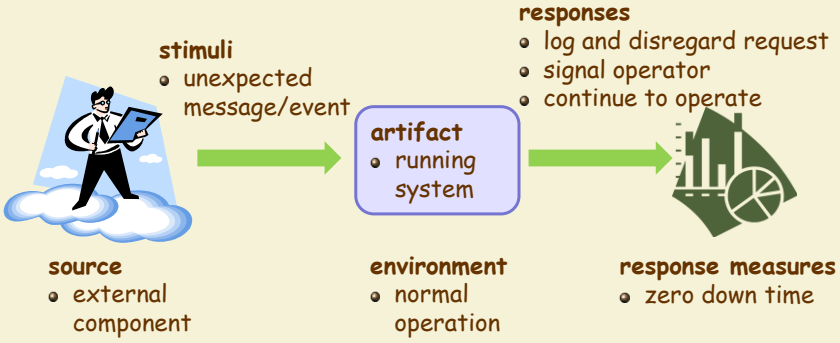
artifacts
- systems, processors, software, hardware, storage, networks

environments
- operational, test, development, load, quiescence

responses
- detect and notify, record, disable, continue operation in degraded mode

response measures
- repair time
- critical time intervals when the system must be available
- time intervals in which the system can operate in a degraded mode

example scenario
availability



scenario guidelines
cost of change aka modifiability

- things to consider
  - what can change
    - functions, platforms, hardware, operating systems, middleware, systems it must operate with, protocols…
    - QA reqs: performance, reliability…
  - when will the change be made
  - who will make the changes
- example scenarios
  - *the existing engine control processor used for 4 cylinder motors must also be used for future 6 and 8 cylinder motors*
  - *the engine control software is able to accommodate the use of dual processors with no impact to the engine control software source code*

checklist
cost of change aka modifiability

stimuli
- add/delete/modify functionality or QA req

sources
- end user, developer, system administrator

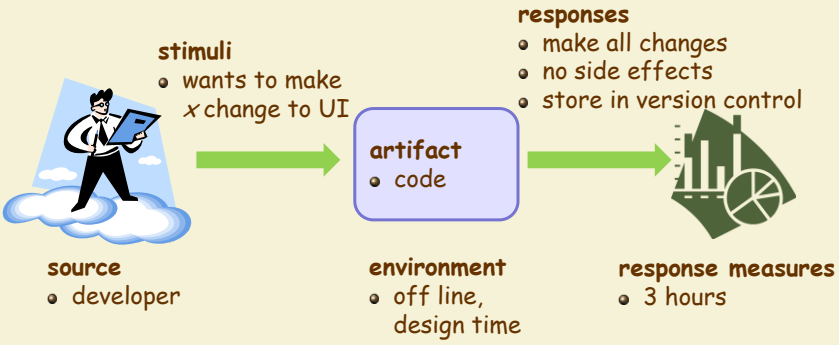
artifacts
- systems, hardware, software
  via human operator
  unless self- - more later*

environments
- run time, compile time, build time, design time

responses
- locate places to be modified
- make modifications without side affects
- test the modification with minimal effort
- deploy the modification with minimal effort

response measures
- extent of changes, effort, cost
- side effects on other functions and/or QAs

example scenario
cost of change aka modifiability



scenario guidelines
response time aka performance

- things to consider
  - various sources of events
    interrupts, messages, requests from users, transactions...
  - arrival rates and patterns
    sporadic, periodic, stochastic, or some combination
- example scenario
  - *under normal operating conditions users initiate an average of 1,000 transactions per minute, sd 100, and each transaction is processed with an average latency of two seconds, sd 0.5 s*

checklist
response time aka performance

stimuli
- periodic, sporadic, or stochastic event arrival

sources
- internal, external, software, hardware

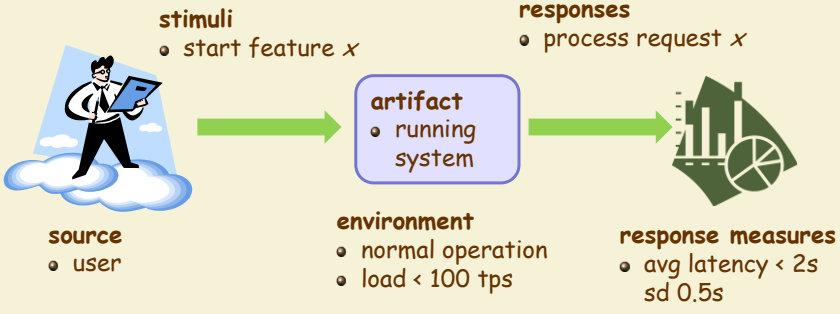
artifacts
- systems, OS, hardware, software

environments
- operational, test, development, load, quiescence

responses
- processes stimuli

response measures
- latency, deadline, throughput, jitter, miss rate, data loss

example scenario
response time aka performance



scenario guidelines
security

- things to consider
  - confidentiality
    information is not released to unauthorized access
  - integrity
    data and services are delivered as intended
  - non-repudiation
    participants in a transaction cannot deny their role
  - authentication
    participants in a transaction are who they say they are
  - auditing
    track activities to enable the reconstruction of events

scenario guidelines
security

- things to consider
  - confidentiality, integrity, non-repudiation, authentication, auditing
- example scenario
  - *when a unauthorized person gains access to the system and tries to modify consumer data, the system detects the malicious behavior, maintains an audit trail of the person's actions, notifies system administration, and shuts down the system*

checklist
security

stimuli
- entity tries to display data, change/delete data, access system services

sources
- an unknown entity that is identified correctly/incorrectly, who is internal/external to the system, authorized/not authorized, and has access to limited/vast resources

artifacts
- systems, services, data

environments
- online/offline, connected/disconnected, protected/unprotected

checklist
security

responses
- authenticate the user
- hide the identity of the user
- block access to data/services
- grant/deny access to data/services
- record access/attempts to access data, modifications to data
- store data in an encoded format
- recognize unexplainably high or unusual demands for services/data
- report/restrict access

response measures
- time/effort/resources required to circumvent security measures with success, restore data/services
- probability of detecting attacks
- probability of identifying individual responsible for the attack
- percentage of services still available under a denial-of-services attack
- extent to which data/services were damaged and/or legitimate access was denied

example scenario
security

stimuli
- modifies write once data

artifact
- data *x*

responses
- log illegal access
- restore original data
- continue to operate

source
- authenticated user

environment
- normal operation

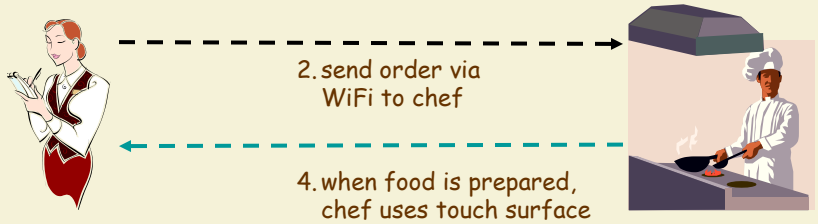
response measures
- data restored within 1 hour

there are many more QAs

- availability, modifiability, performance, and security are some of the most frequently considered, but there are others:
  - scalability
  - safety
  - usability
  - interoperability
  - extensibility
  - portability
  - learnability
  - maintainability
  - testability
  - buildability
  - ...
  - *calibrateability* (some are domain-specific)

in either case, make sure to write the QA scenarios

example
automated order system for restaurant



in class exercise
automated order system for restaurant

- what are the architectural drivers:
  - high-level functional requirements?
  - constraints?
  - QAs?
- can you characterize one of the QAs using a scenario?

in summary

- architectural drivers shape the architecture
  - high-level functional requirements
  - constraints
  - quality attributes (QAs)
- QA names are vague:
  need to characterize QAs using scenarios
- QAW is a method to elicit
  and prioritize QA scenarios
- can't have it all:
  architectural design is about balancing tradeoffs

additional reference

- Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; Wood, W. *Quality Attribute Workshops (QAWs), 3rd Edition* Technical Report CMU/SEI-2003-TR-016: Software Engineering Institute, 2003.