UNIVERSITY OF TECHNOLOGY SYDNEY

ARCHITECTURAL QUALITY ATTRIBUTES

Tom McBride, University of Technology, Sydney

UTS: ENGINEERING AND INFORMATION TECHNOLOGY

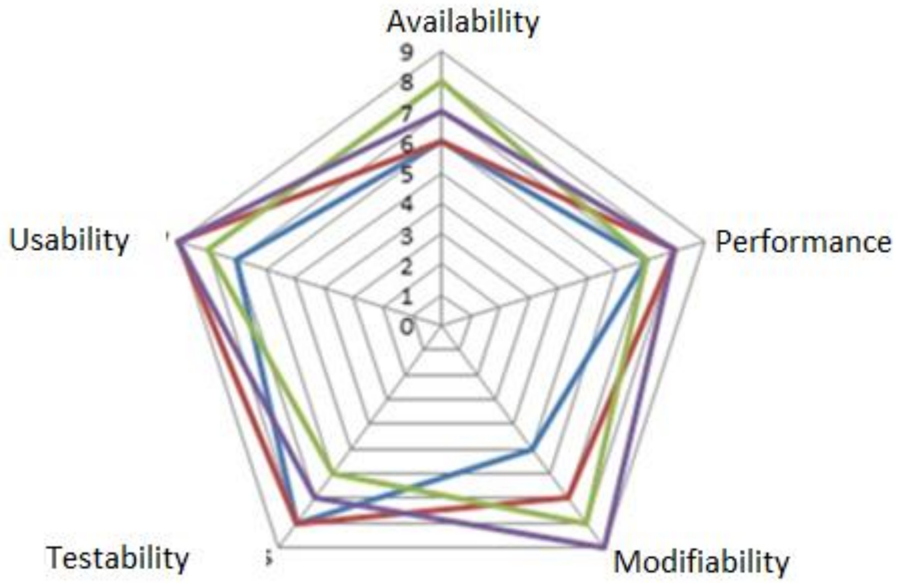
Introduction

- Architecture Quality Attributes
- Determining what qualities you want
  - Availability
  - Modifiability
  - Performance
  - Testability
  - Usability
  - Security
  - Scalability

DETERMINE THE QUALITY REQUIREMENTS

The first problem is establishing what qualities are needed

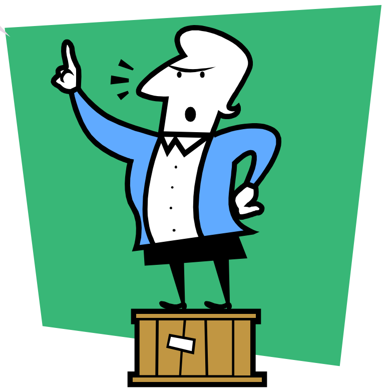
- Stakeholders seldom clearly state their non-functional requirements
- But the software architecture will favour some non-functional characteristcs over others.
- Stakeholders will have different non-functional priorities
- Where and how can we determine the required non-functional characteristics



Usage narratives

- An informal but useful way to describe system functionality as scenarios
- Create “characters” and tell a “story”
- Sometimes seen as a prelude to writing use cases
- Provide context and add meaning to the requirements
- It is difficult to communicate tacit information in formal specifications

He took his vorpal sword in hand:
  Long time the manxome foe he sought
So rested he by the Tumtum tree,
  And stood awhile in thought.



An example usage narrative

"Julie is interested in correlating sightings of Perameles Nasuta (Long nosed bandicoot) in the Northern beaches area of Sydney with bushfire patterns. She brings up tracking data for the last five years and proceeds to sort the data, and then export it into a form that it can be used by a statistical analysis package."

"Julie" is a science officer with the National Parks and Wildlife Service.

Product qualities

- Stakeholders may assign different importance to different architectural qualities
- Which stakeholders will be affected by the product
- How important is each quality to them
  - Give each stakeholder 10 or 20 points to spend on the attributes
  - This reduces “I want all of them”
  - You can’t have all of them, they contradict each other

Non-functional requirements

- Expressed as quality attributes
- Determine how the software behaves
- Architecture deals with identifying, analysing and realising quality attributes
- Generally can’t get everything everyone wants, so must sacrifice some qualities to achieve other qualities

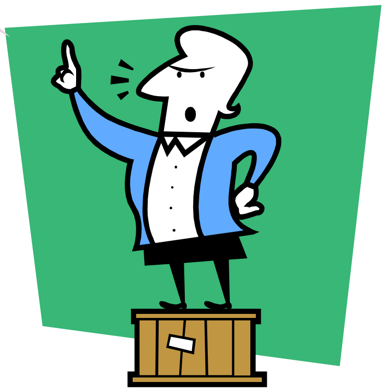
Quality narratives

- I don’t know what you mean when you say “fast”
- Meaning of qualitative terms depends on the user and the context
- Some people understate, some overstate
- Tell me a story that illustrates “fast”

Quality narratives

- What do you mean by "fast"?
- When does it matter and why?
- A narrative or scenario that highlights a quality attribute need
  - Context
  - Action
  - Response

He took his vorpal sword in hand:
 24 hours the manxome foe he sought
So rested he by the Tumtum tree,
 And stood 5 minutes (mean) in thought.



Measurable if possible!

Ask the customer

- A general question like “Does the system need to be reliable?” invites a response of “Of course”
- Try to devise scenarios that illustrate how important is the quality attribute.
  - If the system was being updated when a call came in, would you allow the update to continue or would you want the update to be suspended while the call is processed?
  - If I could make the system run 30% faster at the cost of increasing maintenance costs by 40%, would you accept that?
  - If a hacker managed to penetrate the firewall, is there any data that they must be prevented from accessing?
  - We expect to update this system every six months. Full system testing would normally take one week. Would you prefer a less frequent update with a longer system test?
  - Have you the capacity to run the old system while the updated system is being installed and tested?

The main architectural qualities

- Availability
- Modifiability
- Performance
- Testability
- Usability
- Security
- Scalability

Summary

- Find the required quality attributes in the user stories, narratives or use cases
- You could always ask the customer, but present them with trade-offs between competing qualities.
- You can’t maximise all qualities because they are in conflict – some require a tightly coupled system, others require a loosely coupled system
- The quality priorities are the customer’s decision, not yours.

UNIVERSITY OF TECHNOLOGY SYDNEY

DECIDING ON QUALITY PRIORITIES

Deciding which qualities matter

Stakeholder	Available	Modify	Perform	Test	Secure	Usable	Scalable
Fleet owner	5	1	2		3	0	
Driver	5		1		1	3	
Regulator		2		1	5	2	
V Designer		5		1	1	2	1
Totals	10	8	3	2	10	7	1

Try to discriminate between alternatives

- Tendency to want everything, so give every feature the same rating
- This does not help anyone
- Try to discriminate between alternatives so that it becomes clear what needs to be done
- Give each stakeholder $10 (10 votes) or $20 to buy the features they want

UNIVERSITY OF TECHNOLOGY SYDNEY

AVAILABILITY

Different views of availability

- Small, real-time systems
  - E.g. control system, mobile phone
  - Largely determined by programming
- Large systems
  - Multiple subsystems with multiple processes
  - Deliver overall availability with faulty components

Availability factors

- Fault detection
  - Heartbeat
  - Ping
  - Exception
- Fault prevention
  - Remove from service
  - Process monitor
- Fault recovery - Repair
  - Redundancy
- Fault recovery – reintroduction
  - Rollback
  - Restart

Some situations have enough problems without adding unreliable systems

'WELL, HIGGINS, I SEE EVERYTHING'S UNDER CONTROL.'

Lessons from Erlang

- Erlang is a language used to built telecommunication systems.
- High rate of failure is a few seconds downtime per year.
- Build a system with a lot of small processes that run concurrently.
- Processes can communicate only through messages. No common memory.
- Let failed processes crash.
- Restart processes.

Achieving availability

- Prevent failure
  - Filter data
  - Validate outcomes
- Detect failure
  - Heartbeat between dual subsystems or processes
  - Ping from one subsystem to another
  - Catch an exception
- Cope with failure
  - Redundancy and failover
  - Transaction based systems
  - Graceful degradation
- Recover from failure
  - Restart at a known state then recover

About reliability

- Mobile phones have several mega bytes of software in them
- The phones cost very little
- The cost of recalling a phone to correct a fault would exceed the profit per unit by a very large margin
- Guess who treats reliability very, very seriously

Availability (reliability)

- Reliability needs often depend on criticality:
  - Inconvenience
  - Loss of income
  - Loss of life
- Different systems have different concepts of reliability
  - Financial systems can crash, but dare not lose any data
  - Telecommunication systems can lose data but need to recover and restart
  - Control systems must maintain control, come what may

You don’t need system problems when hot metal is coming toward you

Achieving availability

- An architecture assumes that the software might fail.
- Modular system.
- Frequent integrity checks.
- Delete and restart faulty processes
  - Each process may be faulty but the overall system should be reliable.
- Provide graceful degradation, not sudden failure.

MODIFIABILITY

Modifiability

- Aim of modifiability is to increase the ability to change while minimising its impact and cost
- The world changes constantly, and in ways we didn't predict
- No matter how well designed the software, someone will want to change it

Tactics for modifiability

- Localise changes
  - Maintain semantic coherence
    - Cohesion and coupling
  - Anticipate expected changes
    - Look at similar systems to see how they have changed
- Generalise the module so that specific implementation can be changed easily.
- Make small, single purpose modules that can be modified easily.

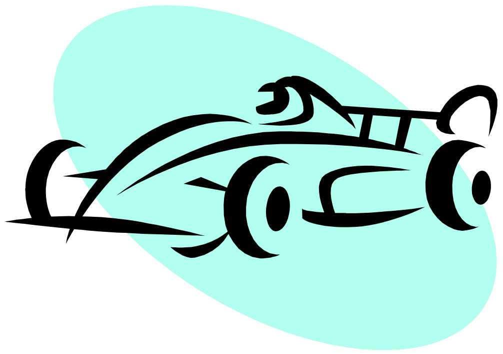
Modifiability interferes with…

- Performance
  - Loosely coupled highly modular systems degrade performance
- Security
  - All those connections provide more potential access points

PERFORMANCE

Performance

- Performance manifests in many ways:
  - Latency
  - Throughput
  - Response
  - Memory efficiency
- Different performance measures may apply to different parts of the system



Some systems won't wait

Nasa launch control

Achieving performance

- Small system
- Tightly coupled
- Minimise low speed interfaces e.g. between servers or between modules
- Monolithic systems are fast but hard to write and maintain
  - Original airline reservation system was one large block of assembler code. Fast but impossible to maintain.
  - Improved CPU speed, better languages, better architecture overcame the need for assembler language systems.
- Many large web systems are moving to a micro-services architecture
  - Performance achieved through horizontal scaling

UNIVERSITY OF TECHNOLOGY SYDNEY

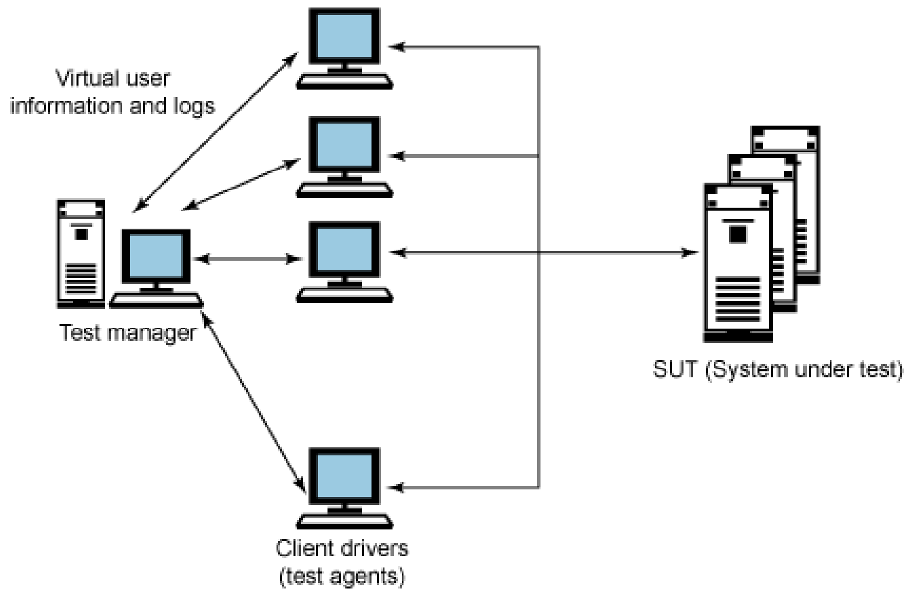
TESTABILITY

Testability

- The goal of testability is to allow for easy testing
  - When new software is developed
  - When correct functioning needs to be proven
- Testing consumes a high percentage of development cost
- Anything done to reduce the cost of testing helps

Tactics for testability

- Separate the interface from the implementation
  - Allows potentially automatable stubbed testing
- Provide a trail of data that can be examined
  - Usually switch this on or off
- Provide specific test functions
  - Start up self-test
  - Specific function test
- Make it modular


Virtual user information and logs
Test manager
SUT (System under test)
Client drivers (test agents)

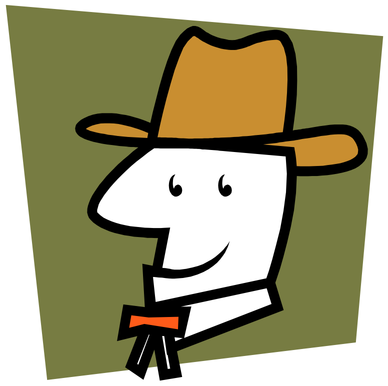
What opposes testability

- Performance and security oppose testing
- Testing opens access into the system, security wants to close access
- Testing adds extra functions, performance wants to remove extra functions

USABILITY

Usability

- Usability has many aspects:
  - Learnability
  - Enjoyability
  - Time to complete task
  - Error rates
- For good results, usability must be assessed by a usability expert
- Some functions, like transaction rollback, need support from the architecture



Usability depends on the user

- To a stockbroker, usability means “do it now”
- To an unsure new user, usability means “Keep me safe”
- To a control system operator, usability means “keep me informed and do what I need done”

Usability depends on the user

Achieving usability

- Separate the user interface from the rest of the system
- Localise user interface changes because they change a lot
- Walk through possible user scenarios
- Model the task separately from the user
  - Model the task
  - Model the user
- Model – view – controller

SECURITY

Security

- Almost any system can be under threat:
  - External attack (network)
  - Policy weakness
  - Data integrity
- High security is very expensive
- Secure expert help

Some things need to be secure

Achieving security

- Resist attacks
  - Authenticate users
  - Minimise exposure
  - Limit access (less gates into the castle)
  - Provide buffer between the system and the external world
- Detect attacks
  - Provide internal checks (security pass)
- Recover from attacks
  - Restore
  - Maintain an audit trail

Lessons from the past

- Castles, forts etc. tend to have very few points of access
- Access points are heavily and easily defended
- Secure systems should have very few access points
- Internal access should be controlled e.g. file access control lists

- However, some recent investigations concluded that there was no such thing as a secure system – internal threats were as big a problem as external threats.
- Accidental damage is just as bad as deliberate damage

Summary

- Secure systems tend to be tightly coupled with few access points
- Access is controlled
- Internal access should be controlled

UNIVERSITY OF TECHNOLOGY SYDNEY

SCALABILITY

Scalability

- Gaining prominence, especially in e-commerce applications
- Some very good lessons being learnt from those who have significant transaction rates.
- As companies shift to public facing systems, they usually experience a significant increase in transactions
- The architecture that suited the user base within a company is seldom capable of serving a global user base.

A bigger ox or more oxen

- Vertical scaling
  - Bigger CPU
  - More resources
- Horizontal scaling
  - More CPUs
  - Cloud deployment

Many are migrating to micro-services architecture

- Extending the principle of small cohesive modules to the process level.
- Decompose the application into a set of collaborating services.
- Develop and deploy services independent of each other.
- Communicate with and between services using a lightweight mechanism.
- Scaling is achieved through starting another instance.
- More flexible scaling and more responsive to modifications.