

quality attributes
scenarios & tactics

do not need to memorize definitions

systems developed to satisfy functional requirements

main problems/reasons for redesign

1. lack of functionality
2. difficult to maintain, patch, scale
3. lack of performance too slow / too large resource footprint
4. lack of dependability
5. lack of security

quality attributes:

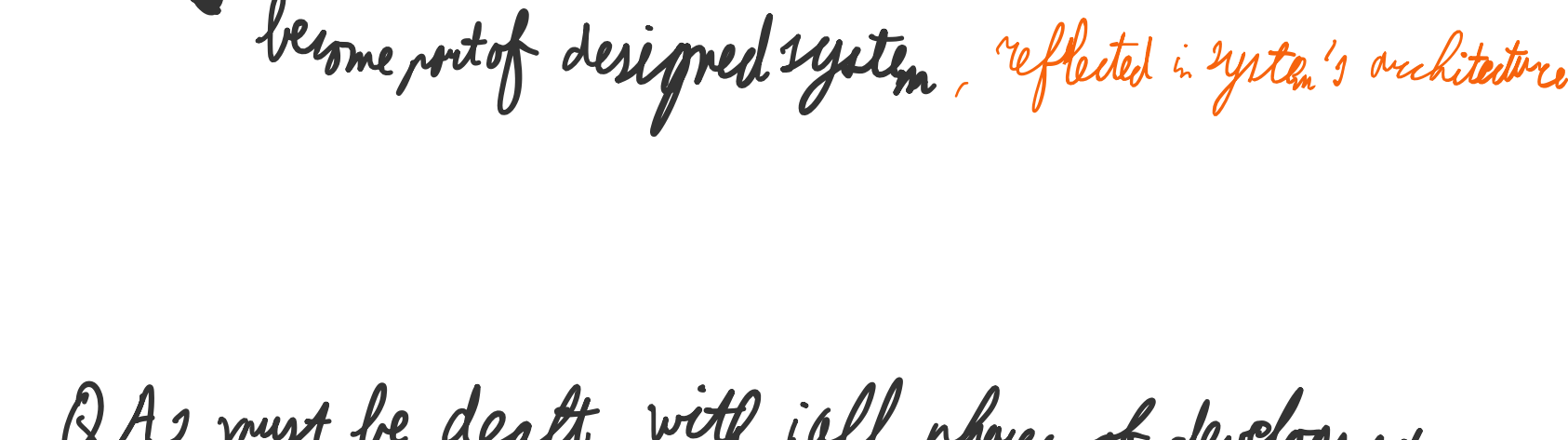
- availability: readiness for correct service expressed as probability
- reliability: continuity of correct service
- modifiability: how difficult it is to introduce desired changes
- performance: timely response to service request costs, throughput, jitter
- testability: how difficult it is to verify correctness of system
- usability: user-friendliness
- portability: how difficult it is to run the system on different platform

dependability attributes: availability, reliability, safety, integrity, maintainability

security attributes: confidentiality, integrity, availability

- QAs in system lifecycle
- quality of developed system
- quality of architecture
- quality of development process
- business qualities time to market, cost/benefit, flexibility

Scenario specification: define what is desired response in particular situation



QAs must be dealt with all phases of development

dependencies exist between QAs

limited manpower + time-to-market results a trade-off

for a specific system, not all QAs are equally important

Brewer's CAP theorem

any networked shared-data system can have at most two of the following three desirable properties

- consistency
- high availability
- tolerance

conflicting QAs require prioritization

QA scenarios specify requirements: inputs, preconditions, required outputs



models may be needed to capture the stimuli

quantifiable metrics may be needed to specify the desired level/intensity of the response

scenarios seem to be specified using

- source
- stimulus
- state of system
- affected resources
- response & tactics
- response measure

assumptions

models

MTTF = mean time to failure

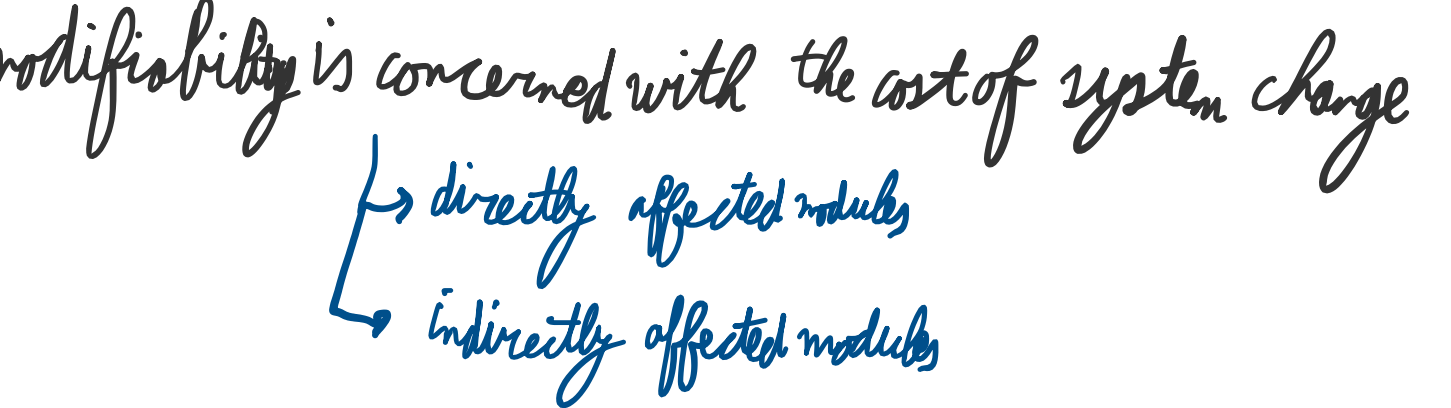
MTR = mean time to repair

MTBF = mean time between failures = MTTF + MTR

reliability = MTTF should be high

availability = $\frac{MTTF}{MTTF + MTR} = \frac{MTTF}{MTBF}$ should be low

expresses a probability



modifiability is concerned with the cost of system change

- ↳ directly affected modules
- ↳ indirectly affected modules

coupling → degree to which responsibilities of two components are related

stronger coupling ⇒ higher cost to modify

single change may affect multiple modules if there are dependent modules overlapping responsibilities

cohesion → degree to which responsibilities of a single module form a meaningful unit

stronger cohesion ⇒ lower cost to modify

modifiability tactics: localise modification

- maintain semantic coherence
- anticipate and isolate expected changes
- abstract common services
- generalize the module/data representations
- limit options

modifiability tactics: prevent ripple effects

- hide information / data encapsulation
- maintain existing interfaces
- insert intermediary between modules

Example: use MVC

modifiability tactics: defer binding time

- runtime registration
- configuration files
- polymorphism
- late (i.e. runtime) binding of calls
- component replacement
- runtime binding of independent services

modifiability is addressed

- in development view
- in process view
- through applied styles & frameworks