CECS 343 SWE – Quality + Testing Wk #14 vLect #2 4/20

o13.3.1 Arch Styles p 258

Data-Centered Arch

Big DB, many **CRUD** clients (Create, Read, Update, and Delete data)
Issues:
ACID: Atomicity Consistency Isolation Durability
Atomicity: all or nothing rule (parts of xtn)
Consistency: only valid data in db (wrt constraints)
Isolation: seq of overlapping xtns can't interfere w each other
Durability: DB atomicity if crash during xtn

BASE: (Wordplay on "ACID" from Chemistry) [**Nox**]
Basic Availability (AKA local availability)
Soft-state (AKA inconsistent for newest changes)
Eventual consistency (AKA consistent for older changes)
o- Badly chosen expressions (hard to remember)

CAP theorem:
Consistency: all processors see same thing
no Avail: (by delaying local access till remote change is here too)
no Partition: (by having no remote processors)
Availability: distant changes available immediately
(by delaying local access)
can't have **Consistency**, **Availability**, and **Partition** tolerance,
you have to settle for two out of three.
(Eric Brewer) [**Nox**]

MVC: Model-View-Controller (**Model does calc & state**; Views display;
Controllers provide input & display manipulation)
o- Model has fcn call API. Model tells each View about state updates. Ctlr calls API.
Publish-Subscribe Arch: Subscribers **sign up for data updates** with Publisher.

Client-Server Arch: Server has agents to help handle big DB, Client does all UI.
Data-Flow Arch
Raw data passes through network of transforms/filters; UNIX "pipes and filters"
Call/Return Arch: simplest: call fcn, & **wait** ("**blocking**") to get its return value.
RPC Arch: (Remote Procedure Call (AKA Call-Ret)) multiple 'nodes', each running async
(*) Blocking (wait for answer from remote machine)
Callback Arch: Pass fcn to call when done, and continue with your stuff till answer comes
AKA **Non-Blocking Call** – Relies on O.S. Interrupt handling
Polling Arch: Opposite of Callback: Caller keeps asking if anything is ready to use.
Event-Loop Arch: Simulation, RTOS (Realtime O.S.)
Agent Arch: (AKA OO Arch) agents send msgs to each other
P2P Arch (Peer-to-Peer Network Arch)
Network Archs
Layered Arch: Lower Layer provides concepts/tools/fcns used by next higher level.
Main EX: TCP/IP: 4-layers App level down to Net Packet level.
Tiered Arch: (Side-by-Side) Cli-Svr is a 2-Tier arch; Cli-BizLogic-Svr(+DB) 3-Tier

o12.2.1 Software Quality Guidelines and Attributes p227
- o- **Non-Fcnl Reqts** (AKA **Quality Reqts**) (AKA the "**Ilities**")

FURPS
- o- **F-**unction**ality** (Actually, NOT a **non-Fcnl** Reqt; included by "committee")
- o- **U-**sab**ility** (Doesn't make the user work hard [**UI Rule #1**])
- o- **R-**eliab**ility** (~ always works when the user needs it)
- o- **P-**erformance (AKA Efficiency) (**fast**, small, frugal)
- o- **S-**upportab**ility** (AKA Maintainab**ility**) (easy to **find/fix/extend**)

o19. Quality Concepts p412

o19.2.3 **ISO 9126** Quality Factors p418

FURPS + Portab**ility** (to another CPU, OS, Framework, Platform, Language)
- o- Was superseded a few years ago (now with 30+ ilities, and counting)

o20. Review Techniques p 431

o- *"The later you find a mistake/defect, the costlier the fix."*

Kinds, for coding: (usuallly by team mgmt, of junior coders)

o- **desk check**: informal, by yourself, go over the code, try to find mistakes (design & code)

o- **walk-through**: formal, go over the code w/ 3-6 onlookers, task issues to be fixed
- o-- time-consuming, so expensive

o- **inspection** (Gilb93 nox): formal, go over randomized (say ~ 20%) sample of code, pass if no issues in sample (*) saves time

** Some people use "walk-through" and "inspection" as **synonyms**.

*- XP's **pair pgmg** has built-in "continuous review" (two heads at one desk/screen/kybd)
- o-- Mgmt: paying two people to do the work of one?

*- "**Lessons Learned**" **after project** (AKA Post-mortem analysis)
- o-- to improve processes for next project

o- Agile "**retrospective**", after each "sprint" (AKA devel period)
- o-- velocity & quality, stats
- o-- story points estim: error bars, trend line
- o-- workflow hiccups
- o-- umbrella issues // outside dev team
- o-- big **tech debt** issues (**Tech debt** is stuff you saw that **needs cleanup**, but wasn't)

o21. SW Quality Assurance p 448

SQA
Goal: that stds are actually followed
Audits: **Reviews** are called Audits
Testing: ** To find errors (Mindset: "**Break it**") (**AKA Red-Team,** playing "bad guys")
SQA checks test planning, test execution processes, and test result docs

(*) **Improve via measurement**
"To measure is to know."
"If you can not measure it, you can not improve it."
Q: What are you not measuring?
Q: Can your measurements **predict reliably**? (most S/W measurements don't)
Q: What % of devr time do your measurements **take away from productivity**?
Q: Is it simple to measure? Does measurement involve more than 4 parameters?
o- If more than 4, probably not worthwhile
Std:
LOC/SLOC (KLOC/KSLOC) = Lines of Code or S/W Lines of Code
(big error bars – 3x?)
Function Points (>4 parms)
Cyclomatic Complexity
(Works for small fcns, mostly)