CECS 434 SWE – Project Plan+Metrics Wk #15 vLect 4/29

o31.2 People p 687
o31.2.2 Team Leaders p 688
- o **Motivate** = push or pull devrs to do assigned tasks
  - o- Carrots, Sticks, and how **polite**, **helpful**, and **warm** mgrs are
- o **Organize** = plan & control **flow of tasks**; ensure they get done;
  and **adjust/recover** if they don't
  - o ID problem task/progress **early** & plan to adjust/recover it
- o **Innovate** = leave room for devrs to **feel creative** (AKA "The Toyota Way")
  - o But ensure quality
- o **Leadership =** loosen reins except where pbms identified
- o **Generous =** reward successful innovation
- o **Social** = understand and try to adjust **stress levels** in devrs

o31.5. The Project p 697
"90-90 Rule" (AKA "90% done" really means (probably) only 50% done)
- o- 1st 90% of prj uses 90% of time == project seems on track
- o- ***Last 10% of tasks use another 90% of time ==> massive overrun (eg 80%)***

Why?
- o- Easy stuff done in the first 90%
- o- Pbm "issues" often delayed, to make "progress"
- o- **Integration** is when you find significant design mistakes
  requiring **significant** run-time bug find/fix time
  - o-- Traditional project estimation assumes 50% time spend on Integration

(**Fix:** Many short "deliveries", tested by users – or at least by pseudo-users
Hence: Incremental Working Deliveries)

Ch 32 Process (M.O.) and Project Metrics p 703
o32.2.1 Size-oriented Metrics p 709
** Bugs are called **Defects after ship, *sometimes*** called **Errors before ship**
Stds:
- o- **LOC** = Lines of Code (error bars 2-3x) [can be "gamed" for benefit]
- o- **FP** = Function Points (no error bars), widely used, [CS: not very good]
  - o-- Reqs 100+hrs training to do "properly" → too complicated

o32.3.1 Measuring Quality p 717
(CF **Tom Gilb** 1988 Princs o SW Proj Mgmt)

Correctness metric: defects/KLOC/year (found by user, & verified by team)
- o- LOC (Lines Of Code) is frowned upon (KLOC = 1,000 LOC)
- o- But its error bars are at most about 3x
- o- Error bars for alternatives are much worse (eg, "Function Points")

Maintainability metric:
Mean/Avg Time to Change/Fix:
- o- Analyze bug, design fix, code, test (& maybe deploy, eg via "patch")

o32.3.2 Defect Removal Efficiency p 718
DRE = E / (E + D) = Errors/(Errors + Defects)
- o- E = errors found before ship, (can be "gamed")
- o- D = defects found after ship (usually) by the users
- o- Best DRE = 1 ==> no defects.

o33. Estimation for SW Projects p 727 (Skipping ahead)
CF Software Estimation: Demystifying the Black Art, **McConnell** 2006

Estimate vs Target-plus-Commitment
When executives/managers ask for an "**Estimate**" of a task/project,
they're often (usually) asking for a
a Target you believe is reasonable, **plus**
a Commitment to complete the task/project by that Target
(even from the hospital bed)

Target: Biz objective; may not be achievable
(*) Objective has Hard and Soft parts
Hard is usually timeframe, and often resources/staff
Soft is often features and/or quality, because these are fuzzier to measure

Commitment: Promise to achieve a Biz Target
Deliver defined (Targeted) **functionality**
at a specific level of (Targeted) **quality**
by a certain (Targeted) **date**
with given (Targeted) **resources**

Estimate: predicted guess (maybe achievable)

Key point of Estimation
(*) Gotta do it: Biz **depends on predictability**
Buy product because: It does X, costs Y, delivered in 2 weeks COD.
(*) Estimate is a Prediction, of **the future**

Key Approach to giving an "Estimate" to "mgmt"
(*)** Always add **error bar**
(*) Never forget **168 hrs/week**: sleep, eat, traffic, holidays, weekends
(*) Never forget **90-90 Rule**: a lot of time spent on Integration
** Never forget that each delivery needs **package/test/install** time
o- Always mention expected functionality, quality, resources

Error Bar examples:
o- "90% likely on or before This Date" "given expected reqts and resources"
o-- → 1 chance in 10 of an overrun
o- "80% likely by before This Date"
o-- 1 chance in 5 of an overrun
(*) Practice as much **estimation** as you can: preferably daily

o**32.4 SW Process + Metrics p 719** (Skipping back)
Most teams don't measure (especially small teams, non-Agile M.O.)
Why:
o1. **Fuzzy**: most metrics have big or **non-existent error bars**
o2. **Time vs Dev**: complicated manual metrics take time away from dev
o3. **Promotes Gaming**: (easy to double LOC metric if used for devr bonuses)
o4. Baseline Setup Time: for same team, same kind projects: 1-3 years usual

o32.4.2 Estab Baseline p 720

Keys for any metric:
o1. **Repeatable predictions**; accuracy v precision
o2. **Many Data Pts**: to better understand statistical distribution involved
o3. **Same Data Kinds** (eg, don't mix size metrics: LOC & Fcn Pts)
o4. **Similar Projects**: to reduce uncertainty (error bars)
o5. **Error bars** == Uncertainty == Standard Deviation size
  o- Usually assume Gaussian/Normal/Bell-shaped distribution

Con:
- o- teams get better over time (usually): more familiar with tools/procedures
- o- teams change members over time: could be better or worse

o32.4.3 Metrics Collection, Calc, Eval p 721

Key: Judge metrics accuracy by metric-based **prediction vs actual results**.
Else, it isn't science
o- Hence, save prediction & compare w results
Cons:
- o- **Diff impacts of reqts changes** on diff projects can invalidate comparisons
  - o-- There are **almost always reqts changes** during project
  - o-- Some times no schedule change is allowed
- o- Hard to **count failed project**; cuz it never completed so don't know whole cost

o32.5 Metrics for Small Shops p 721

95% of shops < 20 devrs on team
- o- No time to collect metrics
- o- 3 year baseline → **30+% devr turnover**
- o- Small team can (maybe) pick 1 metric

Easy metrics:
- **o- Start Delay**: Time till devr begins work on task
- o- **Task Wallclock**: Time till task compoleted
- o- **#Fix Defects/1st_year**

o32.6 Estab SW Metrics Pgm p 722

CF SEI = S/W Engr'g Institute (Carnegie Mellon, Pittsburgh)
- o- for a lot of S/W project mgmt & metrics info & templates

o33.5 SW Proj Estimation p 733 (Skipping ahead)

Based on: **3 Kinds**
o1. **Historical Baseline**, similar completed projects – good for SPLines
Con: poor proj similarity; team mix changes
  o- Error bars
o2. **Top-Down Decomp**
Con: Reqs very good mgr/architect (few); **very subjective**
o3. **Empirical model** (eg COCOMO) Big proj DB & build "scientific" model
  o- Fill in the "tuning" variable "blank"; turn the crank; get the estimate
Con: need baseline for knobs; poor sensitivity; **too many knobs**
  o- Error bars?

McConnell-2006 p 58 "the Black Art"
(*)* One Feature → 100x implementation time
"potential differences in how a single feature
is specified, designed, and implemented
can introduce cumulative differences
of a 100x or more in implementation time"

o**33.10 Make or Buy/Outsource Decision p 748**
Key Qs for "**Make or Buy"** (AKA **COTS** vs **NIH = "Not Invented Here"**)
- o- **Delivery date**: when is S/W package ready, either way (you might fail)
- o- **Cost to Buy + Customize** vs in-house full creation
- o- **How reliable?** (how long it's been out & used; size of user base)
- o- **Supt cost + response time vs in-house devrs immediately available (almost)**

xx03

[nox] 33.2 Proj Planning Process p 729 (Skipping back)
"Task Set" (see Pressman15 p730 for details)
o- Scope, Feasible, Risks, Resources, Effort/Cost, Scd