Week 2: Project Scheduling

- **Project Scheduling:** Distributing estimated effort across a project's planned duration in order to allocate effort/resources to certain tasks effectively & efficiently
  - Define all product tasks; Build a task network w/ interdependencies; Find & track critical tasks & paths,
  - Ensure that all delays are recognized
- **Reasons why Projects Are Late:**
  - Unrealistic Deadline from someone outside the group (knows no technical details)
  - Schedule changes don't account for customer requirements changing (poor planning, tracking, managing)
  - People underestimate resources/effort required; • Predictable/unpredictable risks ignored during planning (bad judgement, forced gets)
  - Technical difficulties that couldn't have been predicted in advance (unlucky)
  - Not recognizing that a project is behind schedule, and then taking the steps to correct/advance it
  - Miscommunication between members that causes delays
  - Human difficulties that can't be predicted in advance (unlucky, accidents)
- **Project Scheduling Principles:**
  - **Compartmentalization:** Divide a product/process into a manageable number of tasks & activities
  - **Interdependency:** Find how tasks are connected/related (establish task interrelationships; find critical & non-critical tasks)
  - **Time Allocation:** Find start & completion times for tasks; consider interdependency for a given task
  - **Effort Validation:** Ensure enough members are assigned to a task to complete it on time
  - **Defined Responsibility:** Assign members to tasks; Ensure all tasks have members assigned to them
  - **Defined Outcome:** Ensure each task has a defined output (work product)
  - **Defined Milestones:** Associate all tasks have a milestone (work product gets reviewed for quality)
- **Effort Distribution Heuristics:** Front End Activities (40-50 %); Construction Activities (15-20 %), Testing & Installation (30-40 %)
- **Project Scheduling Steps:** List Project Deliverables, Define Project Milestones, Work-Task Breakdown Structure (WBS), Define Task Network (Critical Path Analysis), Scheduling, Earned Value Analysis (EVA)
- **Work-Task Breakdown Structure (WBS):** Hierarchical Representation of tasks in a project; Helps organize & manage project
  - Linear Responsibility Chart, Establishing Activity Precedence, Activity Graph, Task Network, Critical Path
  - **Critical Path Analysis:** Adding start & end times to nodes/tasks in an activity graph
    - **Critical Path:** Sequence of Tasks w/ the longest duration; Is the minimum project length.
    - **Critical Tasks:** Any nodes/tasks belonging to the Critical Path
    - Don't need to focus on non-critical tasks/paths because we can waste time/delay it without loss
- **Earned Value Analysis:** Measuring project development progress (Earned Value)
  - **Budgeted Cost of Work Scheduled (BCWS):** Planned cost of the total amount of work scheduled to be performed

    $BCWS = \sum_{1}^{\text{num of planned tasks}} \text{planned effort @ } n$

  - **Actual Cost of Work Performed (ACWP):** Actual cost incurred to accomplish the work performed up to a date
    - $\sum_{n=1}^{\#\text{ of Actual Tasks}} \text{actual effort @ } n$
  - **Budgeted Cost of Work Performed (BCWP):** Planned cost of work completed up to a date (work completed only)
    - $BCWP = \sum_{n=1}^{\#\text{ of actual tasks}} \text{planned effort @ } n$

    BCWP = BCWS: On schedule **BCWP < BCWS**: Behind schedule

    BCWP > BCWS: Ahead of schedule
  - **Schedule Variance (SV):** Comparison between amount of work scheduled versus amount of work performed for a given date
    - **SV** = BCWP − BCWS ; SV ≥ 0 : On schedule, Ahead Schedule; SV < 0 : Behind Schedule
  - **Cost Variance (CV):** Comparison between budgeted cost of work performed versus actual cost of work performed
    - **CV** = BCWP − ACWP ; CV = 0 : On budget, CV > 0 : Ahead budget, CV < 0 : Behind budget

- Schedule Performance Index (SPI): SPI = BCWP/BCWS; SPI > 1 = Ahead schedule; SPI < 1 = Behind Schedule
  - Measures the progress/on-track a project is to its schedule
- Cost Performance Index (CPI): CPI = BCWP/ACWP; CPI > 1 = Under budget, CPI < 1 = Over budget
  - Measures how a project is doing relative to its budget
- Cost Schedule Index (CSI): Measures the overall health of a project (stability, safeness)
  - CSI = SPI · CPI; CSI close to 1: Project is healthy/recoverable; CSI far from 1: Project is unhealthy/unrecoverable

Week 3: Cost Estimation | Software Project Cost Estimation: Estimating cost, resources, schedule
- Expert-Judgment Based Approach, Static Approach, Dynamic Approach, Algorithmic & Empirical Methods, Machine Learning Methods
- Process-Based Estimation (Bottom-Up): Finds tasks to create project scope; Finds individual costs of each tasks, adds them up
  - Measure effort for tasks in "person-months" instead of costs (avoid standardization of currencies); 24 P-M: 1 person, 24 months
  - Total cost depends on how long the project is supposed to take, how much member is paid, and how many people are used
  - Ex: 46 P-M for all tasks, average labor rate = $8000/month, Estimated Cost = 46 · $8000 = $368,000
  - $8000/month, if worker works 160 hours per month (8 hours a day, 5 days a week, 4 weeks), then they're paid $50/hour
  - If we have $330,000 to spend each month, then number of workers required = 330,000 = 41.25 → 42 workers
  - If we wanted to cut the number of workers in half (42/2 = 21), then we must pay each worker double ($100/hour), and they work 80 hours/month
- Problem-Based Estimation (Top-Down): Find project scope first, divide it into tasks, find cost of each tasks, add them up
  - Uses Lines of Code (LOC) to find effort, which is then combined with labor rate to get the total estimated cost
  - Ex: Average productivity Rate (historical data): 620 LOC/P-M; Average Labor Rate = $8000/month; Cost per LOC = 8000/620 ≈ $13/LOC
  - If total LOC = 33,200, then total estimated cost = total LOC · $/LOC = 33,200 · 13 = $431,000
  - Total estimated Effort = total LOC / Average productivity Rate (person-months) = 33,200/620 = 54 person-months (P-M)
- Function Point Approach: Using Function Points (FP; Measure of project's size/complexity in functionalities needed to be implemented) to estimate costs
  - 5 Domain Characteristics Required: Internal Logical Files (ILF), External Interface Files (EIF), External Inputs (EIs), External Outputs (EOs), External Inquiries (EQs)
  - Get the number of each 5 Domain characteristics, multiply them each by their low, medium, high weighted factors (unique to each), add up
  - That gives us the UFP; Then find the TDI, where we score 14 GSCs (adjustment questions) from 0 to 5 and add them up
  - AFP = UFP * (0.65 + 0.01 * TDI); VAF = (0.65) + 0.01 * TDI; AFP = UFP * VAF
  - Once we have the total AFP, we can substitute it for LOC to find cost per AFP, total estimated cost, total estimated effort

Week 4 | Software Configuration Management (SCM): Managing different software versions/configurations; Tracking & handling any changes/updates
- software changes inevitable, must maintain & incorporate it, organize software modules/packages/systems/documents, document & record it
- can divide our projects into different versions/releases, must review/audit changes, some versions are released, others are being developed
- Software Configuration Item (SCI): Approved code/documentation/hardware that's used in configuration management; is a distinct entity in the SCM process
  - Includes Design Documents, Source code, Data Files, Development Tools; each with their own Software Configuration Item Version (SCIV)
- Baseline: A Specification/Product that's formally reviewed & approved, and is the basis/foundation for further development; "A milestone"
  - Includes system specifications, software requirements, Design specifications, source code, test plans/procedures/data, operational system
  - A baseline must be formally requested to be reviewed & verified before it's added, modified, or removed.
  - An approved baseline is stored in the project database, to where it can be extracted & modified by engineering tasks before reviewed again
  - Issue: Hard to know when to put items or what items should be placed in configuration control; thousands of entities w/ their own IDs
    - Use configuration control too early = too restricted, slow; too late = too much chaos & unrestricted freedom
  - Configuration Management System (CMS): Tool used to track & manage different software configurations/versions
- Single-Dev: One branch, is the main branch, that's it; Small Group Project: Product manager, few teams w/ their own feature branches, merged into main
- Medium-Group Projects: use push+pull requests to merge between feature branches & main branches, main branch goes into CI/CD before production
- Large-Group Project: Has multiple final branches (main, staging, development) that each go through continuous integration/deployment
- Enterprise-Level Projects: Multiple instances of the Large-Group Projects for different areas of the final project.

- Centralized VCS: Have a "central/main repo" that devs can clone their own local versions of and make changes to (checkouts)
  - The devs' local copies can be "checked-in" back to the main repo, where local changes are integrated on the main repo
  - Has snapshotting ("version history") on each individual file; much slower; Ex: Subversion, ~~[illegible]~~ TFVC
- Distributed VCS: Everyone has their own "local repos" that we "push-and-pull" from each other; ~~all legitimate~~
  - Each local branch is just as legitimate as the server copy, with a complete copy of its data & files
  - Can perform operations on the local repo (check-in, check-out, commit changes) and push them to the main server
  - In the Local Repo, files can be committed (officially stored in the repo), a working copy (checked out & modified, but not committed), or "in staging" (being reviewed & compared to the original copy before being committed
  - Then, we push our local repo's committed files to the server repo, where they can pull them and merge them to the local branches; we have our own central repos and can push/pull them to each other; Ex: Git, Mercurial, Bazaar
  - Has snapshotting of the entire project, which has more redundancy, but is more fast;

Week 5: Continuous Integration: team members integrate their work frequently, with each integration being tested & verified by an automatic/automated build to check for integration testing ASAP

- Deals with medium-group projects, with multiple feature branches that push/pull to the main branch, and then the main branch goes under Continuous Integration & Deployment for testing before being put to production
- Each team member integrates at least once daily, leads to many integrations per day
- Integrating multiple times is better than one late integration, as a single integration at the end leads to late testing, late notification of bugs, and ultimately a late, delayed release; too efficient and slow; compounds w/ project size
- Is better to split up development and have many incremental iterations & integrations, which develops & tests a project overtime
- Each repo commit triggers an integration & deployment build by an automated, dedicated continuous integration server
- Since integration & deployment & testing is automated for each commit, we can have continuous & frequent feedback & bug notification, which results in more efficient, refined development of a project
- Continuous Integration servers can also deploy to other, multiple environments other than production, such as Functional Test Environments & Performance Test Environments (Functional, Non-functional Requirements)

Jenkins: Self-contained, open-source automation testing server, which can automate the testing, building, & deployment of software

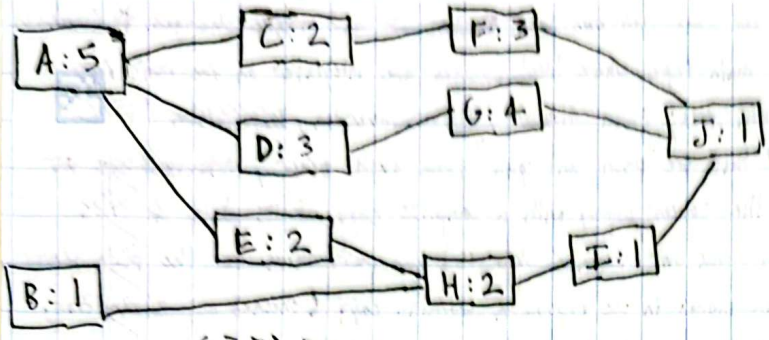
- Can be used to automatically commence building & testing after each commit to a repo
- Also supports VCS like Git, Subversion, etc., and can split up deployment into smaller instances for different tasks

Week 6: Test Automation & Software Defect Automation

- Can automate certain tests for larger-group projects, which deploy to additional functional & nonfunctional testing environments
- Katalon Studio: Test automation software for functional & non-functional requirement testing
  - Record & Replay operations performed on a program/app/website, etc.
  - Catch objects during testing & write scripts to manipulate & use them during testing
  - 2 Modes for testing: Manual Mode & Script Mode
    - Manual Mode: Use the GUI and manually perform the tasks meant to be used & tested on
    - Script Mode: Write script commands in Groovy (similar to Java) for more advanced, specific testing
  - Create & Involve custom variables during testing, better information & data collection
  - Design test data for customizing information collection, helps guide us for using variables to manipulate objects during testing
  - Use multiple test cases & organize them for testing, more customizability & freedom for testing.

Lab 5: Katalon Studio

- New Project → choose type & project directory; Manual Tab → Web Record → Use Url
- Record testing operations, interact & fill in all options; Save script; Show captured objects
- Check Script Tab, see all commands & statements, comment out unnecessary ones
- See. Select Option By Value() & .set Text Commands (), can change values directly or with variables
- Left Side → Data Files → New → Test Data → Data Type: Excel File → Browse → Select File
- findTestData('Filename').getRowNumbers(); findTestData('filename').getValue(column, row)



Activity	Predecessor	Duration
A	\	5
B	\	1
C	A	2
D	A	3
E	A	2
F	C	3
G	D	4
H	B, E	2
I	H	1
J	F, G, I	1

$\overrightarrow{ACFJ}: 11$, $\overrightarrow{ADGJ}: 13$, $\overrightarrow{AEHIJ}: 11$, $\overrightarrow{BHIJ}: 5$

Task	Planned Effort	Actual Effort
1	12	12.5
2	15	11
3	13	17
4	8	9.5
5	9.5	9.0
6	18	
7	10	
8	4	

- BCWS = 12 + 15 + 13 + 8 + 9.5 + 18 + 10 + 4 = 89.5
- ACWP = 12.5 + 11 + 17 + 9.5 + 9.0 = 59
- BCWP = 12 + 15 + 13 + 8 + 9.5 = 57.5
- SV = BCWP - BCWS = 57.5 - 89.5 = -32, behind schedule
- CV = BCWP - ACWP = 57.5 - 59 = -1.5
- SPI = BCWP/BCWS = 57.5/89.5 = 0.64 (behind schedule)
- CPI = BCWP/ACWP = 57.5/59 = 0.97 (over budget)
- CSI = SPI · CPI = 0.64 · 0.97 = 0.62 (less healthy/recoverable)

Lab 3:
- Find/create project folder
- Open folder in Git terminal
- use "git config --global user.name "name"; git config --global user.email "email"", git config --list
- add local repository to project folder w/ git init • touch "filename.filetype" to create file
- git add -A to add all new/updated files to staging • git commit -m "message" to commit/finalize the updates
- can use git commit -a -m "message" to combine git add -A & git commit -m "message"
- git branch "branch name" to create new branch • git checkout "branchname" to switch to that branch
- git merge "branchname" -m "message" to merge the data from "branch name" to the current branch
- gitK shows branch graph

Lab 4:
- git init • git commit -m "message" git branch -M main (rename master to main)
- git remote add origin https://github.com/username/repositoryname.git (add remote/server repo as origin to the local repo)
- git push -u origin main (push local commits to remote/server repo)
- git pull (combines git fetch & git merge; syncs changes from remote branch to local repo)
- git pull "remote-name" "remote-branch-name" → git pull origin branch2
- Check pull request & merge changes from pull request to all branches & repos; main branch is up-to-date
- Jenkins section
  - need UMD campus VPN set up • http://141.215.80.219:8080, username: cis285, password: cis285
  - New item → Freestyle Project → GitHub Project → project url = https://github.com/username/repository name
  - Source Code Management → Git → Repository url = https://github.com/username/repository/name.git
  - Triggers → Poll SCM → schedule: * * * * * (checks every minute)
  - Build Steps → Execute Windows Batch Command: javac Hello.java & or python.filename.py
  - Status → Build → Console Output → java Hello

- verify.TextPresent("string", false) • rowNum=0 • totalRowNum = findTestData('filename').getRowNumbers
  for (rowNum=1; rowNum<=totalRowNum; rowNum++) → starts at 1 • Project → Settings → Test Design → Test Case → Default Failure Handl
  findTestData('filename').getValue(colNum, rowNum)