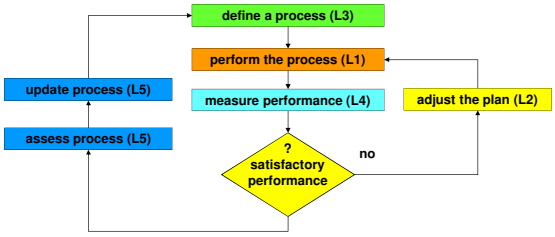
Basic Project Skills

- Project Post-Mortems
  - motivations and their use in this course
  - examine the Diablo II post mortem
- Project Planning
  - work break-down, dependency analysis
  - risk assessment and mitigation
  - estimation and scheduling
- Version Control
  - motivation and its use in this course
  - git and GitHub

Process Improvement 1A (the Capability Maturity Model)



Post Mortems/Retrospectives

- complex skills must be developed/refined
- every project is a learning opportunity
  - improve our skills with existing methodology
  - try new techniques, confront new problems
  - learn from our mistakes
- post-mortems are pro-active learning
  - reflect and discuss as a group
  - what worked and what didn't work?
  - what should we do differently next time?

Post Mortems/Retrospectives

- there are many techniques and formats
- all have the same basic requirements:
  - honesty: willingness to recognize mistakes
  - introspection: willingness to analyze them
  - safety: no penalties for admitting mistakes
- in this course
  - you will develop post mortem processes
  - they will help you learn from the projects
  - you will learn to use them as a learning tool

Diablo II Post Mortem

- Gamasutra article by the D2 design lead
  - brief history leading up to this project
    born from D1 wish-lists and complaints
  - organization, methodology, tools
  - discussion of things that worked
    D2 is Diablo, hiring/development process, new skill trees, massive Q/A effort, world-wide release
  - discussion of things that didn't work
    the old battle.net, # of users, non-state-of-the-art graphics, weak tools, a new game-save feature

Diablo II – Executive Summary

- What worked
  - understand what your customers want
  - people and process determine the result
  - better design can yield a better product
  - you need a plan for how to test & deploy it
- What didn't work
  - saving time/money by repurposing old parts
  - not recognizing that "much bigger" is different
  - not having the right tools to do the work
  - changing key features w/o customer input

Ex A: tasks & dependencies

1. enumerate the project 1 deliverables
2. enumerate major sub-tasks under each
3. make a card for each subtask
   - note the task name, and its due date
   - note which are whole-team v.s. individual
   - note the dependencies between them
     - step (b) requires the output of step (a)
     - we will be better able to do (b) after (a)
     - our grade for (a) will guide us in doing (b)

Ex B: risks and mitigation

1. list 2-3 major (high expectancy) risks
2. classify each identified risk:
   - plan to prevent it from happening
     - how will you prevent this from happening?
   - plan to monitor and deal with it if it does happen
     - how will you know it has happened?
     - how will you deal with it if it does happen?
   - not worth planning for (unlikely, easy to fix)
3. do these incline you to change your plan?
   - a different plan might avoid some risks
   - large risks should be addressed sooner

Ex C: estimation & scheduling

- for each identified sub-task:
  - note optimistic and pessimistic estimates
    if range is wide, discuss approaches and difficulties
- order tasks based on dependencies
- spread tasks between now and due dates
  - leaving slack in proportion to perceived risk
- assign owners and due dates to each

Plan Check List

- task descriptions:
  - everyone believes the list to be complete
  - all task owners understand:
    - what their tasks mean, and how to do them
    - what they will deliver, when, in what form
- risk exposures and mitigation:
  - everyone agrees w/assessments & plans
- schedule:
  - achievable w/adequate room for problems

version control: GitHub

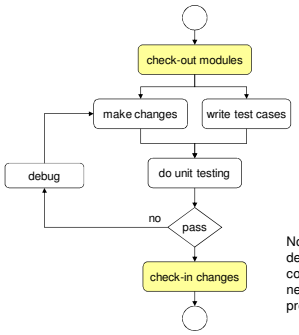
- version control is critical to any s/w project
  - track changes, previous versions, backup
  - work products come from version control
- centralized control is the old paradigm
  - new projects are distributed collaborations
  - distributed version control is more powerful
  - "git" is today's dominant open-source tool
- all project submissions will be on GitHub
- http://www.cs.pomona.edu/classes/cs181f/info/git.html

For next lecture

- McConnell 3-3.2
  - the importance of having a plan
- Kampe: S/W Process Models
  - introduction to project phases and models
- Boehm: Spiral Development
  - iterative development: what, why, and how
- Ambler: "Big Requirements Up Front"
  - the agile critique of the classic Waterfall

Back up slides

software development process



The checked-out modules may include program source code, test cases, make files, configuration databases, etc.

Note that what development delivers is often not executable code. Rather they check in new versions of their work products.

the laws of version control

- All of our work products are versioned
  - we can tell what version we are dealing with
- All official changes are tracked
  - we know exactly what changes were made
  - we know who made each change, when, why
- We can reconstruct any version at any time
  - not just the current version, any prior version
- Files exist in multiple parallel branches
  - each of which has its most current version

version control procedures

- never deliver a work product directly
  - rather, deliver a version-controlled file
  - ensures proper recording of all work
- build from the version controlled files
  - extract specific (or current default) versions
- associate versions with deliverables
  - release has a list of all versions used to build it
  - test/bug reports associated w/specific releases
  - bug fixes are associated w/new file versions
  - work product approvals specify a version

change control

- who can make what changes, where, when?
- sometimes, some change is good
  - it represent progress as work is completed
  - such changes should be facilitated
- sometimes, some change is bad
  - changes can be disruptive to the product
  - we need processes to detect & prevent these
- hopefully these processes are adaptive
  - adjusting the burden in response to the risk

change control mechanisms

- may be performed by version control tools
  - may control who can modify which files
  - may notify interested parties of changes
  - these features are usually configurable
- may be managed by human processes
  - publication and objection
  - designated component reviewers
  - change control boards
- should have mechanism/policy separation

Conflicting Updates

- People may work at cross purposes
  - independent changes to same module
  - different understandings of how things work
- File Change Notifications
  - subscribe to notifications for selected files
- File Locking
  - at check out time, or independently
  - locks can be advisory or enforced
- Change merge assistance
  - automatic difference analysis, proposed merge