Discussion Slides

Deeper

- how could these changes be disruptive?
  - changing requirements
    This could force us to rewrite large amounts of code.
  - changing existing code
    If we change a sub-routine in an incompatible way, other code that calls it might stop working.
  - adding a new feature
    Even if it doesn't break any existing code, bugs in the new feature become bugs in our product … reducing its apparent quality.

Deeper

- Why might change control policy be implemented by version control tools?
  Check-out and Check-in time are obvious points at which to control who is allowed to do what.
- How does mechanism/policy separation apply to change control?
  The rules that determine who is allowed to make what changes when should not be built into the tools that prevent unauthorized updates.

Deeper

Why do we care what versions we're using?

1. *Different versions support different features.*
2. *Different versions contain different bug fixes.*
3. *If support needs to reproduce a problem, they need to try to reproduce it on the same version the user is using.*
4. *If developers are trying to track down a problem, they need to know what version of the code they should be looking at.*

Deeper

Why must we track who changed which modules, when and why?

1. *We need to know, for support reasons, when particular features (or bugs) were added.*
2. *If we don't understand a change, we would like to be able to find the person who made it so that we can ask for their help.*
3. *Knowing why a series of changes were made will often help us to better understand the code.*

Deeper

Why must we be able to reconstruct any version of any file at any time?

1. *to back-out bad changes*
2. *to reproduce or investigate problems in older versions*
3. *to implement a patch for an older version of a product*
4. *in response to a court order*

Deeper

Why must we be prepared to maintain multiple branches of a single file?

1. *Different versions of the module may have been included in multiple supported releases.*
2. *Different versions of the module may be being used in different products.*

Deeper

- Why should modules not be delivered directly (to build/integration/test), but rather be taken from a version control system?
  - *a module obtained from a version control system is a known and versioned object that we can reproduce at any future time.*
  - *a module delivered in source form could be a one-off creation, of unknown provenance, that cannot be reproduced at some future time.*

Deeper

- What does it mean for change control, version control, and defect tracking to be integrated?
  - *committed updates are tagged with defect #s*
  - *change control authorization may be driven by bug priority or commitment.*
- Give an example of how this would be a good thing.
  - *it is easy to figure out which bugs should be fixed by a particular release.*

Deeper

- What are the benefits of change notification
  - *Enable developers to keep track of changes that other people are making to their parts of the system.*
  - *Enable people to quickly learn about problems and fix them ASAP.*
- Does notification stop conflicting updates
  - *No, it merely warns people that they may be happening.*

Deeper

- What are the benefits of locking?
  - *A developer who is working on a module can be certain that nobody else is making changes to the same module at the same time.*
  - *A critical module can be locked so that all changes have to go through a careful review process.*

Deeper

- What are the benefits of enforced locking?
  - *It prevents conflicting changes.*
- What are the problems of enforced locking?
  - *I can lock a file, and then go away, and nobody else can update the file until I unlock it (or until someone can revoke my lock)*

Deeper C1

- Why would we want to have multiple levels of work spaces?
  - *at the bottom might be the code I am trying to get working.*
  - *in the middle is code that is working well enough to be shared with other project members.*
  - *at the top level is code that is ready to ship to customers.*

Deeper C2

- When should check in our changes?
  - *as soon as possible … subject to the goals and rules of the work-space.*
- Why as soon as possible?
  - *smaller increments of work are (usually) more understanable*
  - *to minimize the window for conflicting updates*
  - *so that others can see and work with the new code as soon as possible*

Deeper D1

- Test what you build. Ship what you test.
  - *If you test a different version of a module than the one that you built (from committed versions), you don't know how the official version would work.*
  - *If you ship a different version of a module than the one you tested, can you be certain that the shipped module will work as well as the one you tested.*

Deeper D2

- How could a different library version help or hurt a program?
  - *a new version of a library might contain a new subroutine that our program depends on. We would not work with an older version of the library.*
  - *a new version of a library might contain a version of a routine that contained a bug. Our program might work with the old version, but not with the new version.*

Deeper D3

- How could different compilation options alter the behavior of a program?
  - *different include file directories would contain different versions of include files, supporting different options.*
  - *different library directories would pull in different versions of library routines.*
  - *different* DEFINE*s would enable different features.*
  - *different compiler options would cause different code to be generated (e.g. optimization).*

Deeper D4

- Why must build scripts be maintained by the people who develop the code?
  - *the people who support the code had to work out the build procedures in order to compile and test their code in the first place.*
  - *the people who support the code are in the best position to understand what library versions, definitions, and compiler options are required to make the code work.*

Deeper

- What is good about a “forwards” schedule?
  - *It is an ambitious schedule that shows the minimum time in which a job can be accomplished.*
- What is bad about a “forwards” schedule?
  - *It is ambitious, and may set overly optimistic expectations.*
- When would you use a one?
  - *If someone asked “what is the soonest we could have this done?”*

Deeper

- What is good about a “backwards” schedule?
  - *It is an ambitious schedule that shows us the latest date on which we can start a project.*
- What is bad about a “backwards” schedule?
  - *It contains no slack, and may encourage us to delay starting until too late.*
- When would you use one?
  - *If someone asked “When must we start?”.*

Deeper

- What is good about a spreading the schedule over the available time?
  - *It leaves slack for problems to arise and be addressed and still making our dates.*
- Why is this better than a packed forward schedule?
  - *If one date in a packed forward schedule slips, all other dates must also be changed. These changes can cause disruptions to other peoples’ schedules.*

Project Exercise Legend (major subtasks of Project 1A)

A_1 Concept Development
A_2 Schedule Development
B_1 Competitive Research
C_1 Features Brainstorming
D_1 Initial product descr
D_2 Initial requirements
E_1 Organize Elicitation mtg
E_2 Requirements Elicitation
E_3 Report from Elicitation
F_1 Reqts Reconciliation
F_2 Final Product Proposal
G_1 Post Mortem Meeting
G_2 Post Mortem Write-up