Deeper

- Why would there be so many different models for Post Mortems?
  - projects happen in different cultures
  - participants have different backgrounds
  - some situations are more emotionally charged
  - some situations demand greater formality

Deeper

- What do you think of having managers present at a post-mortem?
  - Why is it a good idea?
    they may be responsible for implementing changes
  - Is there another way to get that benefit?
    they could just read the summary report
  - Why is it a bad idea?
    their presence could stifle frank discussion
  - Is there any way to mitigate that problem?
    maybe with a strong culture and trusted managers

Deeper

- Diablo II is still Diablo
  - kill/reward, continuous goals/accomplishments
  - always on the edge of a quest/waypoint/level
  - even trivial game interactions are fun
  - spectacular monster deaths
  - random adventure/reward generation
  - obvious play … it passes the “Mom Test”
- Understand what made the game fun
- Understand what it is your users want

Deeper

- Blizzard’s Development Process
  - make it playable ASAP for max refinement
  - always open to improving quality of play
  - hire people who love games
  - everyone can suggest improvements
- optimize your process for key results
- people and process determine the outcome

Deeper

- Character Skill Trees
  - started as a way to add character classes
  - allow branching character development
  - characters develop rather than get skills
  - players can customize their characters
  - makes every game and character unique
- Design details can make huge differences

Deeper

- Massive Quality Assurance Investment
  - Diablo II had many more possibilities
    - characters, equipment, parties
  - characters developed along long paths
  - testing combinatorics were horrendous
- they created a large Q/A group
  - designed a systematic testing plan
  - decomposed work into specialized groups
- testing complex s/w is as hard as building it

Deeper

- World-Wide Release
  - Diablo II would have a large international base
  - multiple independent releases are a bad idea
    - they cost more money
    - they have less impact
    - later customers get mad, go rogue
- release plan is part of the product too
- if you plan it, it can go a lot better

Deeper

- Developing/Launching new Battle.net
  - new features stressed old infrastructure
  - trying to upgrade the old system was a mistake
    - it was hard to do and didn't work very well
    - it limited their capabilities
- Scalability doesn't happen by default
  - new powers of ten mean new problems
    - many more people exploring the edges
    - situations that couldn't happen at smaller scales
  - new features create new usage/load patterns

Deeper

- 640x480 8-bit 2D graphics
  - new 3D tools were slow and difficult to use
  - nobody complained about Diablo graphics
  - by the time D2 came out, this was obsolete
- plan for the future
  - that is where your product will be used

Deeper

- Tools that were not of "tool quality"
  - hacks, abandoned after a single use
  - time wasted by bad (disposable) tools
  - tools that didn't support the work flow
- The tools are as important as the product
  - order of magnitude productivity differences
  - they need to be well designed and maintained

Deeper

- New Save-Game methodology
  - traditional whole-world save was difficult
    - Diablo II world had much more state
    - it doesn't make sense in a multi-player game
  - they chose a much simpler model
    - reset the world to its initial state
    - put the character in the last town visited
- Many players hated it
  - this prevented experimentation
  - it forced players to replay prior situations

Deeper

- How do you know if you need to break a task/deliverable into sub-tasks?

  If you already know how to do it, it probably does not need to be further broken down.

  If you are not yet sure how to do it:

  perhaps it needs to be more completely described

  perhaps it needs to be broken into sub-sub-tasks

  perhaps you need to do some research first

Deeper

- Why do we need to specifically identify the whole team tasks?

  Meetings with other people are harder to schedule and need to be planned well in advance.

Deeper

- How can you know what "might" go wrong?
  - What parts of the problem or proposed solution do we not yet clearly understand?
  - What difficult and necessary skills do we not yet have?
  - What investigations/experiments might not yield the needed results?

Deeper

- How do you decide whether to prevent, monitor and respond, or just deal with it when it happens?
  - ease of detecting and dealing with it later
  - cost of prevention v.s. cost of dealing with it

Deeper

- Why should potentially serious problems be "provoked" as early as possible?
  - some problems may be so difficult that solving (or avoiding) them requires a different plan.
  - understand the problem and its solution while there is still time to deal with it.

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

- 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

- 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

- 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

- 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*