Deeper

- What information do we gain in trying to find a minimal failure case?
  - *We find a range of situations that fail.*
  - *We find variations that succeed.*
  - *Keep records of what does and does not work. This may enable us to infer necessary and sufficient conditions for failure.*
  - *This alone may be enough to diagnose the problem.*

Deeper

- What characteristics of a bug do we use to determine its initial priority?
  - *Impact (how severe are the consequences)*
  - *Likelihood (how often will it be encountered)*
- How does isolating a minimal failure case support this assessment?
  - *Isolating the necessary and sufficient conditions for failure will help us to determine how often the problem is likely to occur.*

Deeper

- Why must we make such predictions and confirm our hypothesis?
  - *To ensure that we truly understand the problem. If we do not yet understand the problem, we are not yet ready to fix it.*
  - *If we cannot make predictions and explain all the symptoms we misunderstand the program, the problem, or both.*
  - *This is one of the reasons that almost half of all bug fixes, themselves, contain bugs.*

Deeper

- How can adding diagnostic code change the symptoms of a problem?
  - *It slows the program down, changing the relative timing of events.*
  - *It changes the size of the program, changing the addresses of data structures and code.*
  - *It changes the call structure, changing uninitialized data values on the stack.*

Deeper

- Give an example of the difference between a symptom of a bug, and the real bug?
  - *A routine might blow up because it dereferenced a null pointer, which it was passed as a parameter. Perhaps the code is insufficiently robust, but this is only a symptom.*
  - *Why is somebody passing a null pointer? This is more likely to lead us to the real problem.*

Deeper

- What would warn us to look for other similar bugs?
  - *If the bug was not a random mistake, but rather was the result of a misunderstanding, it is likely that misunderstanding found expression in multiple places.*
  - *Examples:*
    - *the meaning or proper use of some variable*
    - *the parameters or effects of some method*
    - *the proper initialization or finalization of an object*
    - *how to enumerate a list or array*
    - *synchronization of some resource*

Deeper

- How should one go about making fixes?
  - *Use version control so you can get back to the original version (in case we make matters worse).*
  - *Make and test one change at a time (so we don't have to deal with interactions between multiple changes).*
  - *Thoroughly test after each change (make sure it is right before continuing).*
  - *Make minimal changes, avoid rewriting code on general principles (you are more likely to do harm than good).*

Deeper

- It has been suggested that every bug that makes it into final code is evidence of at least two problems. Explain this.
  - *There is the defect that gave rise to the reported problem.*
  - *There was apparently a bug in our test plan, that we did not recognize this as a situation that needed testing.*
  - *There may also be problems in our specification, design, and review processes.*

Deeper

- Give an example of random bugs (that do not arise from an underlying problem).
  - *There are surprisingly few. Perhaps:*
    - *mis-spelled variable or routine names.*
    - *mis-typed expressions.*

Deeper

- Give examples of bugs that stem from a root (e.g. educational or methodological) cause, and may therefore be preventable.
  - *uninitialized variables*
  - *failure to check return values for errors*
  - *missed error cases*
  - *dead locks*
  - *buffer overflows*

Deeper

- Why do we record the diagnosis and work arounds in a bug report?
  - *So that other people can assess how well the diagnosis explains all of the reported symptoms.*
  - *If future users encounter the same problem, they can quickly find out how to get around the problem, or where to get the necessary fix or patch.*

Deeper

- Why must we clearly describe the behavior we believe to be in error?
  - *The problem may be a disagreement about how the component is supposed to work.*

Deeper

- Why must we clearly describe the impact the problem has on users?
  - *This will help to prioritize the bug … higher impact problems get quicker attention.*

Deeper

- Why must we clearly describe the platform and software versions involved?
  - *The problem may only occur on a particular platform, or with a particular version of the software. It may be completely irreproducible on any other configuration.*

Deeper

- Why should users, testers, or support staff have to refine the problem down to minimal failure cases?
  - *Complex failure cases (involving large files and many operations) may be very difficult to replicate. If development cannot replicate the problem, they cannot fix it.*
  - *A clearer description of the failure case may lead to an immediate understanding of the problem.*
  - *Specific reports tend to get quicker attention than vague ones.*

Deeper

- How can project management use this information?
  - *To decide if product is ready to ship.*
  - *To decide how long it will take to fix outstanding bugs.*
  - *To estimate how many more bugs will be found.*
  - *To assess development quality and efficiency.*
  - *To identify processes that need improvement.*

Deeper

- How, objectively, can we prioritize bugs?
  - *We can ladder (partially order) them on the basis of probability, impact, and availability of a work-around.*
  - *Where to draw the line is less objective, but it probably involves:*
    - *How consistent is this error with our general expectations for this release?*
    - *How much time, effort, and risk do we believe to be associated with fixing this bug?*
    - *Choosing the lesser evil*

Deeper

- What information guides our initial hypothesis formulation?
  - *Information about which test cases work and which ones fail.*
  - *Output that suggests where, in the processing, the problem arose.*
- How does this guide hypothesis formulation?
  - *Look for code that is involved only in the cases that fail, or that might be sensitive to those conditions.*

Deeper

- How can we confirm a hypothesis?
  - *Make predictions about other circumstances that should surround the failure, and look at the data to confirm them.*
  - *Make predictions about other cases that should and should not fail, and run the program to confirm them.*
  - *Make predictions about exactly how the code must be written, and examine it to confirm them.*
  - *A good hypothesis enables many predictions and explains all the observed symptoms.*

Deeper

Why can't we just change the code until it works?

Why can't a surgeon cut away at your brain until your symptoms go away?

If you don't fix the real problem, the symptoms may change, but that doesn't mean the problem is fixed.

1. *Understand how program was meant to work.*
2. *Understand that plan went wrong.*
3. *Understand the correct solution*
4. *Implement it.*

Deeper

- How should we test a bug fix?
  - *create new automated regression test cases that check for all instances of the bug (not just the one that was originally reported)*
- Why should these tests be automated?
  - *– automated test cases are cheap*
  - *– exercise the bug and verify the fix*
  - *– ensure that the bug(s) don't reappear later*
  - *– prevent us from making similar mistakes again*

Deeper

- Why is the one line description so critical?
  - *People looking for already reported problems will review these one line summaries to see if the problem has already been reported (and if a fix or work around is available)*
  - *It is the way people will refer to the bug, so it should be as accurately descriptive as possible.*
  - *It will affect peoples (at least superficial) perception of the bug's priority.*

Deeper

- Why is it important to provide the name of the component that is suspected to be at fault?
  - *Because problems are routed to developers based primarily on this information. If this is correct, the problem will quickly be routed to the right developer. If it is incorrect, the problem may be bounced around for a while.*

Deeper

- Why do we have an unconfirmed state for new bugs?
  - *Because there are so many bad bug reports.*

Deeper

- Why is it a good idea to close and consolidate multiple reports of the same bug?
  - *If there is only one problem to solve, multiple bug reports may hide this.*
  - *Get all of the information associated with all instances of a problem into one report.*
- Is it always clear that a new problem is another instance of an existing bug?
  - *No. The symptoms of a bug may vary greatly depending on context.*

Deeper

- Why don't we close a bug report as soon as the fix is verified?
  - *The problem still exists until the fix is available to customers.*