1. Triage
   1. Definition
   2. Origin – WWI
   3. Triage Tags in emergency rooms
2. Triage Steps
   1. Environment
      1. QA vs Production (example of when Ryan said the one exchange site was down because of Raven)
   2. How many people are affected
      1. Example of when it was only a couple of people but we assumed it was widespread.
         1. Not to downplay the importance of those issues
      2. Example of when we thought it wasn’t happening across the board
   3. Symptoms
      1. When I click x then y happens
   4. Narrow down what application is having the problem
      1. Is it in the front-end?
      2. Is it in a particular back-end service
      3. Is it in the database
3. Debugging
   1. Gather details like IDs, time of occurrence, what’s happening and what should’ve happened.
   2. Is the issue reproducible? In QA?
   3. If so, is it safe to reproduce the issue in production?
   4. Log files are spread around, make sure to check them all
4. Logging
   1. Don’t log too much
   2. Don’t log too little
   3. Goldilocks
   4. Log at info for production, anything you’d like to see normally in the logs should be info
   5. Verbose is for more details when running locally or in dev, or if you really need it you might need it in production
   6. Log exceptions whether they’re handled or not, especially if they are unhandled. Log exceptions that will crash the process.
      1. AppDomain and Task unhandled exceptions
   7. If all else fails, check the Event Log
      1. Helpful in diagnosing startup issues that might happen before logging has initialized
      2. If there is nothing in the event log, the startup may be failing because there are errors in the app’s config file.
   8. Quality over quantity, a few log statements with vital details make all the difference
5. Exercise (everything in the DB is timing out):
   1. Ask audience to diagnose the issue
   2. “The SSC isn’t working”
   3. “When I open a screen it freezes with a spinner”
   4. “Now we’re getting some reports that the website won’t load”
6. Memory Dumps
   1. Only as a last resort
   2. Memory leaks are easier to find with a profiler
   3. Random crashes
      1. May have logged the exception, so check that first, or the Event Log
   4. Infinite loops
7. WinDbg
   1. Ctrl + D to open a dump
   2. “.loadby sos clr” must be called first for managed applications
   3. !analyze –v is useful when opening a crash dump. It will check the last exception that was thrown and show you details about the thread that threw it. I’ll be demoing this later.
   4. !Threads will display all of the managed threads in the application
   5. !EEStack will print the full stack trace of every thread
   6. !DumpHeap –stat will give you a summary of all the objects on the stack. When looking into high memory usage you’ll use this to find what’s using the most memory.
   7. !GCRoot can be using to find what is holding onto a particular object
8. Configuration
   1. If you haven’t set up your symbol path, WinDbg will give this error and you won’t be able to debug through a dump.
9. Demos
   1. First demo the crash, show what the event log looks like, then show how to take a crash dump using ADPlus
      1. Open the dump in WinDbg and show !analyze –v
      2. Can also show !Threads and !EEStack here
   2. Next demo the memory leak
      1. Show how to take the dump using Task Manager
      2. Open the dump in WinDbg to show !DumpHeap –stat
      3. Open the dump with DebugDiag to show memory diagnostics
   3. If there’s time left demo a hang
      1. Show stack traces in Process Explorer
      2. Take the dump with ADPlus –hang
      3. Show !EEStack in WinDbg
      4. Show the dump in Visual Studio
   4. If there’s time left show Process Monitors