**Advantages of using Spring**

1. Dependency injection :- Given an application context xml or by providing @Configuration, classes and the services within it can be instantiated. This reduces boiler plate code, supports coding to Interface, reduces coupling, unit testing is made simple. Factory method design pattern can be replaced (like the handler instantiation that I used in the solution).
2. Spring can be integrated with many different frameworks like hibernate eg by using hibernateTemplate
3. Spring supports Aspect Oriented Programming where in the security, logging and transaction are abstracted there my removing boiler plate code. Eg we currently use TransactionProxyFactoryBean for transaction management along with hibernate
4. Spring enables plain old java programming

**When would you use Automated testing**

1. During development every method should be unit tested
2. This helps in making sure any changes made does not break the existing code / functionality.
3. Helps in refactoring the code. Example if we want to refactor a big chunk of code, we can refactor bit by bit and run Junit tests in parallel to make sure we do not break the existing functionality.
4. Good measure of code quality (Code coverage).

**You just received the root access of your project's freshly installed linux box. Describe the main steps you'll take to run the war on it.**

1. Topology of an app should have firewalls, load balancers, Tomcat servers where the app is deployed.
2. Assuming firewalls and load balancers are taken care, I will go ahead and install tomcat server
3. Configure tomcat’s global configurations found in conf directory.
4. Mainly the server.xml.
   1. Configure JNDI – Java Naming and Directory Interface resources that allows Java software clients to discover and look up data and objects via a name.
   2. Set up jvm route to support load balancing
   3. Configure Host to unpack war with the app base name and set auto deploy to true.
   4. Clustering can be configured.
5. Have all the configurations of an app (like MQ settings, Kafka, Tibco, DB, springs applicationContext.xml etc ) in resources directory.
6. Having all these done I will go ahead and deploy the app in tomcat.
7. Which will be deployed in webapps directory and war file would be unpacked.
8. Now restart the tomcat server.

**Once deployed, your users start whining about slow pages, timeouts, failed uploads,etc. As a developer, how would you handle this ?**

1. First look at the logs to get clues
2. Timeout can be increased in Web.xml under <session-config>
3. For failed uploads I would increase the <max-file-size> and make sure there are no directory access permission issues.
4. Slow pages are valid as servers load the servlets and action on first heat. So it will be slow on first access.
5. <loadonstartup> can be configured in web.xml which will load required files on server start up.