**Top 11 slacknesses that can bite you as an experienced programmer/architect**

Production issues seek the attention of middle and top level management. Here are a few things that you must pay attention as a software developer or architect to prevent any future embarrassments. You can use this as a check list.

**#1: Not externalizing configuration values**in config (E.g: .properties, .xml, or .yaml) file(s). For example, not making the number of threads used in a batch job configurable via a *config* file. You may have a batch job that worked well in DEV to UAT (i.e user acceptance) environments, but when deployed to PROD takes a longer time to complete due to larger datasets. If the number of threads are configurable, the number of threads can be tweaked. This applies to all other configurable values like web service URLs, host names, port numbers, log levels, timeout values, etc.

**#2: Not testing the application with the right volume of data**. For example, testing your application with 1 to 3 accounts instead of 1000 to 2000 accounts, which is the typical scenario in the production environment. The performance tests need to be conducted with the real life data, and not cut down data. Not adhering to real life performance test scenarios can cause unexpected performance, scalability, and multi-threading issues. It is imperative that you test your application for larger volume of data to ensure that it works as expected and meets the SLAs  (i.e. **S**ervice **L**evel **A**greements) in the non-functional specification.

**#3: Naively assuming that external or other internal services that are invoked from your application are going to be reliable and always available**. Not allowing for proper service invocations timeouts and retries can adversely impact the stability and performance of your application. Proper outage testings need to be carried out. This is very crucial because the modern applications are distributed and service oriented with lots of web services. Indefinitely trying for a service that is not available can adversely impact your application. The load balancers need to be properly tested to ensure that they are functioning as expected by bringing each balanced node down.

**#4: Lack of due diligence relating to**timezones (E.g. UTC vs local zones), monetary calculations (E.g. use BigDecimal or Money class as opposed to float/double), not writing thread-safe code, not defining transactional boundaries, reinventing the wheel by writing your own logic when there are already well written and proven APIs and libraries are available, resource leaks, etc. This is discussed in detail [In your Java experience, what are some of the common mistakes developers, especially beginners make?](https://www.java-success.com/module-20/11-java-experience-qas/)

**#5:  Not adhering to the bare minimum security requirements**. As mentioned above, web services are everywhere, and web services can be easily exploited by the hackers for the denial of service attack. So, use of  SSL layer, basic authentication, and penetration testing with tools like Google skipfish are mandatory. Unsecured applications can not only adversely impact stability of an application, but also can tarnish an organization’s reputation due to data integrity issues like customer “A” being able to view customer “B’s” data.

**#6: Not performing cross browser compatibility testing**. Modern web applications are rich single page applications making use of JavaScript code and frameworks like angularjs. The sites you build need to use responsive designs to work well across devices and browsers. It is imperative that proper cross-browser and device compatibility testing is performed to ensure it works in all of them.

**#7: Not externalizing business rules that are likely to change often**. For example, tax laws, government or industry compliance requirements, classification laws, etc. Use business rules engines like Drools that allow you to externalize rules into database tables and excel spreadsheets. The business can take ownership of these rules, and can react quickly to changes to tax laws or compliance requirements with minimal changes and testing.

**#8: Not having proper documentation in the form of**

1. Unit tests with proper code coverage.
2. Integration tests.
3. A confluence or wiki page listing all the software artifacts like classes, scripts, configuration files that have been modified or newly created.
4. High level conceptual diagrams depicting all the components, interactions, and structures.
5. Basic documentation for developers on  “how to set up the DEV environment with data source  details.

Points 1 and  2 are the primary form of documentation in an**agile project** in addition to the COS (**C**ondition **O**f **S**atisfaction) created via tools like *MindMap*.

**#9**: Continuous integration and continuous delivery (**CI/CD**) process not properly implemented from the start. Continuous integration (**CI**) is a process in which developers and testers collaboratively validate new code. Continuous delivery (**CD**) is the process of continuously creating releasable artifacts. CI/CD is a process for continuous development, testing, and delivery of new code that enables organisations to release more often. Tools like Jenkins, Docker containers, Git Hub, etc can be used to automate the process.

**#10: Not having proper disaster recovery plans, system monitoring and archival strategies in place**. It is easy to get missed on these activities in a rush to get the application deployed to meet the tight deadlines. Not having proper system monitoring through Nagios and Splunk can not only impact the stability of the application, but also can hinder current diagnostics and future improvements.

**#11**: **Not designing Database tables with proper house keeping columns** like created\_datetm, update\_datetm, created\_by, updated\_by and timestamp,  and provision to logically delete records with columns like ‘deleted’ with ‘Y’ or ‘N’ values or record\_status like ‘Active’ or ‘Inactive’. Proper constraints are equally important to not corrupt the data. Not having a **version** column for the optimistic concurrency.

**#12**: **Not having proper system backout plan** to restore the system to its stable state before deployment if anything goes wrong. This plan needs to be properly reviewed and signed-off by the relevant teams. This includes backing out to previous versions of software artifacts, any data inserted into the database, properties file entries, etc.

**#13**: **Not performing proper capacity planning** at the beginning of the project.  Its no longer sufficient to simply say that you “need a Unix box, an Oracle database server and a JBoss application server” when specifying your platform. You need to be really precise about the

* specific versions of operating systems, JVMs, etc
* how much memory (including physical memory, JVM heap size, JVM stack size, and JVM perm gen space)
* CPU (number of cores)
* load balancer, number of nodes required, node types like active/active or active/passive and clustering requirements.
* file system requirements, for example, your application may archive generated reports and keep it for a year before archiving them. So, you need to have enough hard disk space. Some applications require to generate data extract files to be generated and temporarily stored to be picked up by the other system processes or data warehouse systems for multi dimensional reporting. Some data files are SFTP’ed from other internal or external systems, and need to be kept for a period like 12 to 36 months before archived.

Adding **#13**based on “David DeCesare’s” comments at “[java.dzone](http://java.dzone.com/articles/11-mistakes-come-back-bite)“,

**#14.** “Not using the best tool for the job”. Too often developers will use a tool or language in production systems that they want to learn but may not be the best choice. For example, using a NoSQL database when your data is actually relational. Remember, whatever tools you choose, you may have to support them for the next 3-5 years (or longer).

**#15.** Lack of good knowledge in some of the [**16 technical key areas**](https://www.java-success.com/what-are-the-16-technical-key-areas-of-java-programming-and-how-will-they-help-you-fast-track-your-java-career-as-a-developer/)like **identifying**, **reproducing**and **fixing 1).** “Concurrency issues” **2)**Transactional issues **3)**Performance issues **4)** Security considerations e.g. SSL certs, keystores, truststores, etc. In many job interviews, I have sold my skills in these 4 key areas to secure new contracts.