The goal of **Stress testing** is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn’t crash under crunch situations.

The main purpose of stress testing is to make sure that the system recovers after failure which is called as recoverability.

**Stress testing is also extremely valuable for the following reasons:**

* To check whether the system works under abnormal conditions.
* Displaying appropriate error message when the system is under stress.
* System failure under extreme conditions could result in enormous revenue loss
* It is better to be prepared for extreme conditions by executing Stress Testing

**Following are some technical reasons behind performing Stress testing:**

* To verify the system behavior under abnormal or extreme load condition.
* To find the numerical value of users, requests etc., after which the system may break.
* Handle the error graciously by showing appropriate messages.
* To be well prepared for such conditions and take precautionary measures like code cleaning, DB cleaning, etc.
* To verify data handling before the system breaks i.e. to see if data was deleted, saved or not etc.
* To verify security threat under such breaking conditions etc.

**Strategy for Stress Testing**

* Following are some pointers that would help you to strategize your testing process:
* Identify the scenarios, functionalities etc., that will be accessed the most and may tend to break the system. Like for a financial app, the most commonly used functionality is transferring money.
* Identify the load that the system can experience on a given day i.e. both maximum and minimum.
* Create a separate [test plan](https://www.softwaretestinghelp.com/how-to-write-test-plan-document-software-testing-training-day3/), scenario, test case and test suite.
* Use 3-4 different computer systems for testing with different memory, processor etc.
* User 3-4 different browsers for web apps with different versions.
* Ideally, find the value below the breakpoint, at the breakpoint and the value after the breakpoint (when the system will not respond at all), create a test bed and data around these.
* In the case of web apps, try to stress test with a slow network too.
* Don’t jump to the conclusion of tests in just a round or two, execute the same tests for at least 5 rounds and then conclude your findings.
* Find the ideal response time of the web server and what is the time is at the breakpoint.
* Find the app behavior at the breaking point at different points of the app like while simply launching the app, logging in, performing some action post login etc.

**Stress Testing for Mobile Apps**

Stress testing for native mobile apps is a little different from that of web apps. In native apps, a stress test is done for the commonly used screens by adding huge data.

**Following are some verification that is done as a part of this testing for native mobile apps:**

* The app doesn’t crash when huge data is shown. Like for an emailing app, around 4-5 lakhs of received email cards, for shopping apps, the same amount of item cards etc.
* Scrolling is glitch free and the app doesn’t hang while scrolling up or down.
* The user should be able to view the details of a card or perform some action on the card from the huge list.
* Sending lakhs of updates from the app to the server like marking an item as ‘Favourite’, adding an item to the shopping cart, etc.
* Try loading the app with huge data on a 2G network, when the app hangs or crashes, it should show an appropriate message.
* Try an end to end scenario when there are huge data and a slow 2G network etc.

**Following should be your strategy for testing on mobile app**

* Identify the screens which have cards, images etc., so as to target those screens with huge data.
* Similarly, identify the functionalities that will be used most commonly.
* While creating the test bed, try to use medium and low-end phones.
* Try to test simultaneously on parallel devices.
* Avoid this testing on emulator and simulators.
* Avoid testing on Wifi connections as they are strong.
* Try to run at least one stress test out in the field etc.

More:<https://www.softwaretestinghelp.com/stress-testing/>