Both static and dynamic testing play an important role in the testing process. While both play an important role in producing effective and bug free software, the methods by which they check for and correct errors in a system are much different.

Static testing can be defined as any method of testing that is performed on a system without executing the underlying code. One of the main benefits of performing static testing is that it can be performed early and often to prevent system bugs before they happen. This in turn can save a great of time and money that would otherwise be spent debugging.

Static testing can be further divided into two main subgroups of testing methods, reviews and static analysis. In *Software Testing*, reviews are defined as being a means of testing “anything that is written or typed; this can include documents such as requirement specifications, system designs, code, test plans and test cases.” (Hambling et al. 62). Documents such as requirements and system designs are typically drafted well before any code are written, thus the importance of ensuring that these documents are accurate and complete cannot be understated. Static testing provides for this by having numerous reviews done over the course of the project on multiple forms of documentation.

Static analysis differs from reviews in that its primary function is to find errors in project code. Static analysis is typically aided by special automated tools that run on the source code to find bugs. After the tools are run on the software, all errors are reported to be fixed. Testing software in this way can help to point out small mistakes in the code that could have a much larger impact later in the design process.

Dynamic testing is a series of testing methods performed on a system through execution of source code. When we think of software, we often think of only a single program or application, but within that single application are a series of modules that come together to create the application. A single piece of software can have hundreds if not thousands of individual files or modules. Programming in this modular fashion has great advantages in terms of reusability and efficiency, however great care must be taken to ensure that the application does not break due to errors.

Unit testing can be seen as the first line of defense in that unit tests are devised to validate the logic of an individual application component. These tests are often written before the code in what is called test-driven development (TDD). Developing in this fashion allows us to place emphasis on achieving the desired outcomes before the code is ever submitted for further testing.

Integration testing takes the process a step further by ensuring that our individual components work together as expected. The components of a system often need to work together and share resources to achieve a desired task. Integration testing ensures that information flows smoothly between the different system components without errors.

Finally, system and acceptance testing focus on the finished product from a higher-level perspective. System testing focuses on ensuring that the system is behaving as expected, while acceptance testing is a means by which the development team ensures that the finished application is acceptable for the customer’s use.

Designing and implementing effective static and dynamic tests is an essential part of developing production level code. Customers expect a finished product that behaves as intended and is free of bugs, and as such, effective testing can save time and money for all parties involved.

**Sources:**

* Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 3.3 Reviews and the Test Process.* (pp. 62). BCS The Chartered Institute for IT. Retrieved from   
  https://app.knovel.com/hotlink/pdf/id:kt00UC2IE3/software-testing-an-istqb/reviews-test-process