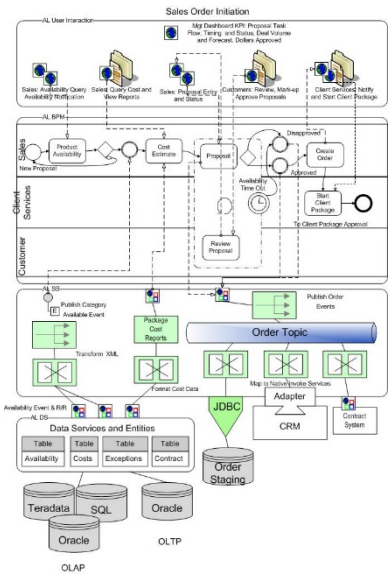
ttc
the testing consultancy
A Mosiac Group Company

Integration Testing White Paper

Software applications are commonly composed of a number of individual modules or components. These components need to interact and exchange information. This paper is an introduction to Integration Testing, a software testing discipline that focusses on testing the interfaces of the modules as well as their interactions with each other.

From monolithic applications to compositional systems

For some time, the software development process has been undergoing a transformation from building monolithic applications, by writing spaghetti code, to compositional modularised systems resulting from the assembly of prefabricated, shared, and independent code. As a consequence, modern applications include a higher number of integration points between their components. In addition, many of these components expose their interfaces and are available for testing as soon as they have been built.
The diagram on the right depicts a model of an even-driven architecture. The individual components are in turn composed of sub-modules in a hierarchical manner.



What is Integration Testing?

Integration Testing reflects the attitude of trying to understand the complexity of a software application and wanting to test its individual components as well as the communication of the components with each other. Integration Testing is more than just testing the communications going through the Enterprise Service Bus or any message queues. Literature on testing offers multiple definitions of Integration Testing, it can be described as:

- The phase in software testing in which individual software modules are combined and tested as a group. The purpose of Integration Testing is to verify functional, performance, and reliability requirements placed on major design items. Test cases are constructed to test whether all the components within assemblages interact correctly – definition as per International Software Testing Qualifications Board (ISTQB).
- Functional testing of components / services at lower levels via exposed internal interfaces to explicitly test their behaviour – also known as API testing.

At The Testing Consultancy (TTC), we define Integration Testing as a combination of the two activities above, starting with a breakdown of any application into its components, and functionally testing them, followed by testing groups of interacting components at an increasingly higher level. However, we distinguish Integration Testing from E2E or System testing by the fact that E2E tests try to verify detailed business requirements whereas integration tests verify the architectural solution and technical design of the system under test.

ttc
the testing consultancy
A Mosiac Group Company

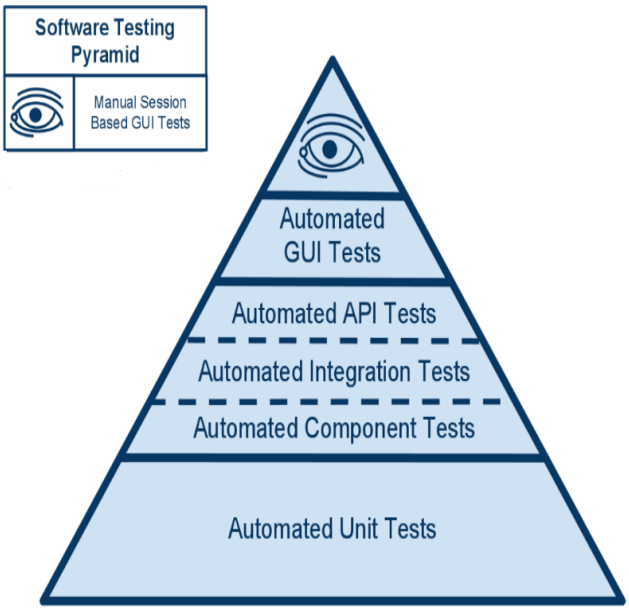
Why do Integration Testing?

- Premise: if internal workings are correct then the whole system will work as intended.
- Integration tests will be performed to confirm that:
- individual and assembled items work as designed;
- higher level components correctly choreograph usage of lower level services (e.g. order of call sequences, using tokens etc.);
- all scenarios, which all potential consumers would use, have been covered during API and component testing;
- consumers accept and process the received information to display expected behavior;

Additional benefits of Integration Testing

- Integration tests are often easier to design, build, execute, automate and maintain.
- It also enables targeted testing of perceived application weaknesses and bottlenecks.
- Tests can easily be prioritised according to the desired strategy (risk based, complexity based etc.).
- With testing smaller components it will be easier to replicate problems and analyse defects.
- Accountability for single components usually sits with individuals rather than large teams. Therefore defect turnaround times will be significantly reduced if defects are found earlier and at lower levels.

How does Integration Testing fit into an overall Test approach?



Integration tests confirm workings of individual parts with exposed interfaces, and of assembled parts (i.e. also test the difference between the whole and the sum of its parts).
API tests are functional tests of services / procedures / interfaces, utilising white box or black box testing techniques, and are based on technical specifications or contract definitions (tests are positive, negative and can include performance and security tests).
Component tests are functional tests of the assembled components, usually based on technical specifications.
For the benefit of this paper, API tests and component tests are considered to be forms of integration tests.
Unit tests confirm workings of individual parts without exposed interfaces (APIs).

ttc
the testing consultancy
A Mosiac Group Company

What industry trends are driving the shift towards performing more Integration Testing?

- **Moving to compositional systems or microservice architectural style:**
  - a single application as a suite of small services;
  - increased number of individual autonomous components with exposed and testable interfaces;
  - increased integration activities and integration testing effort;
  - components released and available as soon as they have been completed.

- **More frequent but smaller releases:**
  - **Continuous delivery** (possibly even Continuous deployment) based on Continuous integration and facilitated by a DevOps approach;
  - because of the frequent releases, it is advisable to focus tests on integrating the new parts as opposed to running full system regression tests;
  - automated builds, tests and deployments to test environments after changes are committed to mainline ➔ Continuous testing;
  - feasible only if most tests are automated.

- **More tests at lower level of testing pyramid:**
  - unit and API / integration tests to be executed extensively;
  - end to end (E2E) tests will be rarer, especially automated E2E tests – they are costly to build and costly and hard to maintain;
  - tests of UI as exploratory tests by manual testers (end user / client experience plus selected functional tests) – similar to UAT (User Acceptance Testing);
  - bottom-up integration testing is applied, i.e. lower level components are tested first or as soon as they become available ➔ “shift left” (in regards to time lines) to test early.

Which other topics are relevant to facilitate Integration Testing?

- **Tight collaboration between developers and testers:**
  - tests cases written (and automated) during development time – not after handover to QA;
  - test driven development (TDD);
  - bottom-up integration testing: development and testing of small units happen at the same time;
  - DevOps model applied.

- **Increased levels of test automation:**
  - decide what and when to automate: API and integration tests are easier to design and build;
  - costs to write and maintain automated tests;
  - test artefacts and application components stored together in version control system.

- **Increased amounts of technical testing:**
  - code reviews;
  - static code analysis;
  - walkthroughs with developer;
  - reviews of technical design and specification documents;
  - reviews of architecture solution documentation.

- **Service virtualisation:**
  - Service virtualisation can be used to simulate services or components that are not available yet or to copy service production interfaces into test environments.
  - Tools will be needed to create virtual services or virtual service environments.
  - Virtual services can be based on real ones or based on description of service (WSDL etc.).
  - Sophisticated service virtualisation tools allow for different levels of complexity in virtual services (data contents, messaging sequences).

What are some of the challenges associated with Integration Testing?

- Test coverage: components / services are often over-engineered. Therefore there is a need to find the appropriate test scope - one that is similar to the superset of all possible consumers of the API which exist in the application under test. In other words, there is no need to test all theoretically possible scenarios and scenarios intended for future proofing of the service: tests should only cover the functionalities that are required by the current application under test (unless services are shared across applications – see below).
- Test case design: usually no detailed requirements available at service / component level. Therefore it is recommended to analyse solution architecture documents and technical specifications (after having earlier confirmed that those reflect the detailed business requirements) to gain insight into service design and functionality, and to be able to design appropriate test cases.
- There is an increased need for QA personal to stay abreast of changes to service design and functionality because test cases need to be added / updated very frequently: tight collaboration is a prerequisite.
- Services / components are often shared and used in multiple applications; therefore changes to these services require API re-testing and also multiple (preferably automated) regression / integration tests.
- Integration Testing requires more technical know-how.

Summary

Integration Testing is a growing area, in importance and also in volume. This, along with moving to more frequent software releases, drives the need for integration test specialists and, even more so, the need to automate integration tests as much as possible.

ttc
the testing consultancy
A Mosiac Group Company

Resources

http://istqbexamcertification.com/what-is-integration-testing/
https://en.wikipedia.org/wiki/Integration_testing
http://www.soapui.org/testing-dojo/welcome-to-the-dojo/overview.html
http://watirmelon.com/2011/06/10/yet-another-software-testing-pyramid/
http://smartbear.com/all-resources/articles/what-is-service-virtualization/
http://katrinatester.blogspot.co.nz/2015/09/continuous-delivery-testing-pathway.html
http://katrinatester.blogspot.co.nz/2015/09/api-web-services-microservices-testing.html
http://martinfowler.com/articles/microservices.html
https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/SoftwareDevelopment-Fraunhofer_SIT.pdf