6.1 Types of Test Tools (K2) — *45 minutes*

Terms
Configuration management tool, coverage tool, debugging tool, dynamic analysis tool, incident management tool, load testing tool, modeling tool, monitoring tool, performance testing tool, probe effect, requirements management tool, review tool, security tool, static analysis tool, stress testing tool, test comparator, test data preparation tool, test design tool, test harness, test execution tool, test management tool, unit test framework tool

6.1.1 Tool Support for Testing (K2)

Test tools can be used for one or more activities that support testing. These include:
1. Tools that are directly used in testing such as test execution tools, test data generation tools and result comparison tools
2. Tools that help in managing the testing process such as those used to manage tests, test results, data, requirements, incidents, defects, etc., and for reporting and monitoring test execution
3. Tools that are used in reconnaissance, or, in simple terms: exploration (e.g., tools that monitor file activity for an application)
4. Any tool that aids in testing (a spreadsheet is also a test tool in this meaning)

Tool support for testing can have one or more of the following purposes depending on the context:
- Improve the efficiency of test activities by automating repetitive tasks or supporting manual test activities like test planning, test design, test reporting and monitoring
- Automate activities that require significant resources when done manually (e.g., static testing)
- Automate activities that cannot be executed manually (e.g., large scale performance testing of client-server applications)
- Increase reliability of testing (e.g., by automating large data comparisons or simulating behavior)

The term “test frameworks” is also frequently used in the industry, in at least three meanings:
- Reusable and extensible testing libraries that can be used to build testing tools (called test harnesses as well)
- A type of design of test automation (e.g., data-driven, keyword-driven)
- Overall process of execution of testing

For the purpose of this syllabus, the term “test frameworks” is used in its first two meanings as described in Section 6.1.6.

6.1.2 Test Tool Classification (K2)

There are a number of tools that support different aspects of testing. Tools can be classified based on several criteria such as purpose, commercial / free / open-source / shareware, technology used and so forth. Tools are classified in this syllabus according to the testing activities that they support.

Some tools clearly support one activity; others may support more than one activity, but are classified under the activity with which they are most closely associated. Tools from a single provider, especially those that have been designed to work together, may be bundled into one package.

Some types of test tools can be intrusive, which means that they can affect the actual outcome of the test. For example, the actual timing may be different due to the extra instructions that are executed by the tool, or you may get a different measure of code coverage. The consequence of intrusive tools is called the probe effect.

Some tools offer support more appropriate for developers (e.g., tools that are used during component and component integration testing). Such tools are marked with "(D)" in the list below.

6.1.3 Tool Support for Management of Testing and Tests (K1)

Management tools apply to all test activities over the entire software life cycle.

Test Management Tools
These tools provide interfaces for executing tests, tracking defects and managing requirements, along with support for quantitative analysis and reporting of the test objects. They also support tracing the test objects to requirement specifications and might have an independent version control capability or an interface to an external one.

Requirements Management Tools
These tools store requirement statements, store the attributes for the requirements (including priority), provide unique identifiers and support tracing the requirements to individual tests. These tools may also help with identifying inconsistent or missing requirements.

Incident Management Tools (Defect Tracking Tools)
These tools store and manage incident reports, i.e., defects, failures, change requests or perceived problems and anomalies, and help in managing the life cycle of incidents, optionally with support for statistical analysis.

Configuration Management Tools
Although not strictly test tools, these are necessary for storage and version management of testware and related software especially when configuring more than one hardware/software environment in terms of operating system versions, compilers, browsers, etc.

6.1.4 Tool Support for Static Testing (K1)

Static testing tools provide a cost effective way of finding more defects at an earlier stage in the development process.

Review Tools
These tools assist with review processes, checklists, review guidelines and are used to store and communicate review comments and report on defects and effort. They can be of further help by providing aid for online reviews for large or geographically dispersed teams.

Static Analysis Tools (D)
These tools help developers and testers find defects prior to dynamic testing by providing support for enforcing coding standards (including secure coding), analysis of structures and dependencies. They can also help in planning or risk analysis by providing metrics for the code (e.g., complexity).

Modeling Tools (D)
These tools are used to validate software models (e.g., physical data model (PDM) for a relational database), by enumerating inconsistencies and finding defects. These tools can often aid in generating some test cases based on the model.

6.1.5 Tool Support for Test Specification (K1)

Test Design Tools
These tools are used to generate test inputs or executable tests and/or test oracles from requirements, graphical user interfaces, design models (state, data or object) or code.

Test Data Preparation Tools
Test data preparation tools manipulate databases, files or data transmissions to set up test data to be used during the execution of tests to ensure security through data anonymity.

6.1.6 Tool Support for Test Execution and Logging (K1)

Test Execution Tools
These tools enable tests to be executed automatically, or semi-automatically, using stored inputs and expected outcomes, through the use of a scripting language and usually provide a test log for each test run. They can also be used to record tests, and usually support scripting languages or GUI-based configuration for parameterization of data and other customization in the tests.

Test Harness/Unit Test Framework Tools (D)
A unit test harness or framework facilitates the testing of components or parts of a system by simulating the environment in which that test object will run, through the provision of mock objects as stubs or drivers.

Test Comparators
Test comparators determine differences between files, databases or test results. Test execution tools typically include dynamic comparators, but post-execution comparison may be done by a separate comparison tool. A test comparator may use a test oracle, especially if it is automated.

Coverage Measurement Tools (D)
These tools, through intrusive or non-intrusive means, measure the percentage of specific types of code structures that have been exercised (e.g., statements, branches or decisions, and module or function calls) by a set of tests.

Security Testing Tools
These tools are used to evaluate the security characteristics of software. This includes evaluating the ability of the software to protect data confidentiality, integrity, authentication, authorization, availability, and non-repudiation. Security tools are mostly focused on a particular technology, platform, and purpose.

6.1.7 Tool Support for Performance and Monitoring (K1)

Dynamic Analysis Tools (D)
Dynamic analysis tools find defects that are evident only when software is executing, such as time dependencies or memory leaks. They are typically used in component and component integration testing, and when testing middleware.

Performance Testing/Load Testing/Stress Testing Tools
Performance testing tools monitor and report on how a system behaves under a variety of simulated usage conditions in terms of number of concurrent users, their ramp-up pattern, frequency and relative percentage of transactions. The simulation of load is achieved by means of creating virtual users carrying out a selected set of transactions, spread across various test machines commonly known as load generators.

Monitoring Tools
Monitoring tools continuously analyze, verify and report on usage of specific system resources, and give warnings of possible service problems.

6.1.8 Tool Support for Specific Testing Needs (K1)

Data Quality Assessment
Data is at the center of some projects such as data conversion/migration projects and applications like data warehouses and its attributes can vary in terms of criticality and volume. In such contexts, tools need to be employed for data quality assessment to review and verify the data conversion and

migration rules to ensure that the processed data is correct, complete and complies with a pre-defined context-specific standard.

Other testing tools exist for usability testing.

6.2 Effective Use of Tools: Potential Benefits and Risks (K2) — *20 minutes*

Terms
Data-driven testing, keyword-driven testing, scripting language

6.2.1 Potential Benefits and Risks of Tool Support for Testing (for all tools) (K2)

Simply purchasing or leasing a tool does not guarantee success with that tool. Each type of tool may require additional effort to achieve real and lasting benefits. There are potential benefits and opportunities with the use of tools in testing, but there are also risks.

Potential benefits of using tools include:
- Repetitive work is reduced (e.g., running regression tests, re-entering the same test data, and checking against coding standards)
- Greater consistency and repeatability (e.g., tests executed by a tool in the same order with the same frequency, and tests derived from requirements)
- Objective assessment (e.g., static measures, coverage)
- Ease of access to information about tests or testing (e.g., statistics and graphs about test progress, incident rates and performance)

Risks of using tools include:
- Unrealistic expectations for the tool (including functionality and ease of use)
- Underestimating the time, cost and effort for the initial introduction of a tool (including training and external expertise)
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool (including the need for changes in the testing process and continuous improvement of the way the tool is used)
- Underestimating the effort required to maintain the test assets generated by the tool
- Over-reliance on the tool (replacement for test design or use of automated testing where manual testing would be better)
- Neglecting version control of test assets within the tool
- Neglecting relationships and interoperability issues between critical tools, such as requirements management tools, version control tools, incident management tools, defect tracking tools and tools from multiple vendors
- Risk of tool vendor going out of business, retiring the tool, or selling the tool to a different vendor
- Poor response from vendor for support, upgrades, and defect fixes
- Risk of suspension of open-source / free tool project
- Unforeseen, such as the inability to support a new platform

6.2.2 Special Considerations for Some Types of Tools (K1)

Test Execution Tools
Test execution tools execute test objects using automated test scripts. This type of tool often requires significant effort in order to achieve significant benefits.

Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of automated test scripts. A captured script is a linear representation with specific data and actions as part of each script. This type of script may be unstable when unexpected events occur.

A data-driven testing approach separates out the test inputs (the data), usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data. Testers who are not familiar with the scripting language can then create the test data for these predefined scripts.

There are other techniques employed in data-driven techniques, where instead of hard-coded data combinations placed in a spreadsheet, data is generated using algorithms based on configurable parameters at run time and supplied to the application. For example, a tool may use an algorithm, which generates a random user ID, and for repeatability in pattern, a seed is employed for controlling randomness.

In a keyword-driven testing approach, the spreadsheet contains keywords describing the actions to be taken (also called action words), and test data. Testers (even if they are not familiar with the scripting language) can then define tests using the keywords, which can be tailored to the application being tested.

Technical expertise in the scripting language is needed for all approaches (either by testers or by specialists in test automation).

Regardless of the scripting technique used, the expected results for each test need to be stored for later comparison.

Static Analysis Tools
Static analysis tools applied to source code can enforce coding standards, but if applied to existing code may generate a large quantity of messages. Warning messages do not stop the code from being translated into an executable program, but ideally should be addressed so that maintenance of the code is easier in the future. A gradual implementation of the analysis tool with initial filters to exclude some messages is an effective approach.

Test Management Tools
Test management tools need to interface with other tools or spreadsheets in order to produce useful information in a format that fits the needs of the organization.

6.3 Introducing a Tool into an Organization (K1)	*15 minutes*

Terms
No specific terms.

Background
The main considerations in selecting a tool for an organization include:
- Assessment of organizational maturity, strengths and weaknesses and identification of opportunities for an improved test process supported by tools
- Evaluation against clear requirements and objective criteria
- A proof-of-concept, by using a test tool during the evaluation phase to establish whether it performs effectively with the software under test and within the current infrastructure or to identify changes needed to that infrastructure to effectively use the tool
- Evaluation of the vendor (including training, support and commercial aspects) or service support suppliers in case of non-commercial tools
- Identification of internal requirements for coaching and mentoring in the use of the tool
- Evaluation of training needs considering the current test team's test automation skills
- Estimation of a cost-benefit ratio based on a concrete business case

Introducing the selected tool into an organization starts with a pilot project, which has the following objectives:
- Learn more detail about the tool
- Evaluate how the tool fits with existing processes and practices, and determine what would need to change
- Decide on standard ways of using, managing, storing and maintaining the tool and the test assets (e.g., deciding on naming conventions for files and tests, creating libraries and defining the modularity of test suites)
- Assess whether the benefits will be achieved at reasonable cost

Success factors for the deployment of the tool within an organization include:
- Rolling out the tool to the rest of the organization incrementally
- Adapting and improving processes to fit with the use of the tool
- Providing training and coaching/mentoring for new users
- Defining usage guidelines
- Implementing a way to gather usage information from the actual use
- Monitoring tool use and benefits
- Providing support for the test team for a given tool
- Gathering lessons learned from all teams

References
6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999