Chapter 26
Glossary of Terms

9’s: See **n nines**.
Acceptance testing: Testing by an end user, customer, or other independent personnel to verify that the system can be accepted for use.

Accessibility error : An error resulting from a user employing a non-standard input or output device, where the system is not usable for those who do not access the system using “standard” devices.

Active attack: In **security testing**, an attack on a system which causes some changes to the system, such as adding a program or modifying data in a database.

Ad hoc testing : A (sometimes considered pejorative) term for **exploratory testing**. **All-pairs testing**: Another term for **pairwise testing**.

Alpha testing: “Real-world” testing by a small group of test engineers or some other small group of technically proficient personnel. Often succeeded by beta testing.
Application under test: The system which the tester is testing.

Assertion : In a unit test, a statement which states that a certain condition must hold. If the condition does not hold, the test is considered to have failed. For example, assertEquals(2, Math.sqrt(4));.

Availability : In **performance testing**, what percentage of the time the system is available to the user (not in a failure mode, unresponsive, etc.) In **security testing**, one of the elements of the **InfoSec Triad**, an attribute that refers to the ability of authorized users to access a system.

Bacteria: A kind of **malware** which consumes an excess amount of system resources, perhaps taking up all file descriptors or disk space.
Bad data error: An error resulting from the system receiving malformed, corrupt, or otherwise invalid data.

Base case : A test case for the basic expected functionality of a system, or an interior value in an equivalence class. For example, when testing a calculator, a base case might be a user adding 2 and 2 together.

Baseline test : A kind of **load test** where a bare minimum amount of events, perhaps even none, are processed, to provide a “baseline” to show what minimal load on the system looks like. **Bathtub curve**: A generalized function of failure rates, which start out high (as poor components fail soon after being implemented), stay low throughout much of the life of the system, then start increasing as the system nears it end of life. So named because it looks like the outline of a bathtub as viewed from the side.

Beta testing: “Real-world” testing by a subset of the actual user base prior to release of the system. Often preceded by alpha testing.
Black-box testing: Testing the code as a user would, with no knowledge of the codebase.

Most manual tests are black-box tests.

Blocked : A status for a test case in a test run. It indicates that the test case cannot be executed at this time, for reasons outside the tester's control. For example, the functionality it tests is not yet at a testable state, perhaps because it has not yet been developed.

Blocker: A defect of the highest severity, where the system cannot reasonably be released without fixing it or providing a **workaround**.
Bot network: A collection of **zombies** controlled by a master.

Boundary value : A value which is "on the boundary" between equivalence classes. For example, a system that has two equivalence classes, between 0 and 19, and 20 or higher, would have boundary values at 19 and 20.

Bounds checking: Run-time checking that data is not being written outside of a properly allocated array.
Branch coverage: What percentage of branches in the code are tested, usually by unit tests. **Brownfield development**: Writing software which must interact with already-existing software in production, thus limiting potential design, solutions, architecture, etc.
Buffer overrun: A **vulnerability** where more data can be written than has been allocated for it. This can cause system crashes or unauthorized access.
Bug: Another term for **defect**.
Chaos Monkey: A **stochastic testing** tool developed by Netflix which tests distributed systems by shutting off random servers in the system.
CIA Triad: Another term for the **InfoSec Triad**.

Code coverage : How much of the codebase is actually tested, usually via unit tests. Although there are different kinds of code coverage, the majority of the time that non-specialists use it, they are referring to **statement coverage**.

Combinatorial testing: Testing in such a way so as to ensure that various combinations of variables will work as expected.
Complete (requirements): The property of having the requirements specify the entirety of the system.
Confidentiality: An attribute of a system, that only authorized users may read data. An element of the **InfoSec Triad**.
Configuration error: An error resulting from a misconfiguration of the system, as opposed to an error in the code which comprises the system itself.

Consistent (requirements) : The property of having requirements which can all be followed without a paradox (e.g., "the system shall display the message 'Hello' upon startup" and "the system shall not display any message upon startup" are not consistent).
Corner case: A test case for functionality of a system which is unlikely to occur, or is beyond the realm where a user will likely reproduce it. By analogy with **edge case** (a corner is where multiple edges intersect).

Covering array: An array which covers all possible combinations of variable values.
Cracker: An unauthorized person attempting to access and/or modify a system or data using underhanded techniques.
Critical: A defect of the second-highest level of severity, which severely impacts how a

user could use the system.
DDos: See **Distributed Denial of Service**.

Defect : A flaw in a system which causes it to behave in an unexpected or incorrect manner, or does not meet the requirements of the system. Much of software testing is involved in finding defects in a system.

Denial of Service: A method of attacking **availability** by sending so many unauthorized packets or other events to a computing resource that no authorized users have access to it.

Dependency injection : Passing dependencies of a method in as parameters, as opposed to having them be hard-coded. This helps with testing as they can easily be replaced with test doubles or fakes.

Deterministic: Something for which the causal behaviors are entirely known and reproducible. **Disk I/O Error**: An error resulting from a fault in reading or writing to long-term local storage (usually, but not always, a disk).
Display error: An error where the correct value was computed, but it was not displayed correctly.

Distributed Denial of Service : A **denial of service** attack which consists of many different sources of the unauthorized packets, so as to increase the number of events the system must process as well as help disguise the ultimate source. Often abbreviated as **DDos**.

Distributed system error: An error arising as a consequence of the system being distributed, as opposed to running entirely on one computer.
Dogfood, eating your own: Another term for **dogfooding**.
Dogfooding: Using your own software while developing it. For example, running the operating system that you are developing on your own computer.

DoS : An acronym for **denial of service**.
DoS tools: Tools which enable denial of service attacks.

DRY : Don’t Repeat Yourself. A tenet of writing good, testable code which states that code should not be repeated, for example by having two different methods which do the same thing, or copy/pasting code from one part of the codebase to another instead of making it into a callable method.

Dumb monkey: A **stochastic testing** method in which random data is sent in to a system.
Dynamic testing: Testing the system by executing it. Examples would be unit testing or black-box testing.

Edge case : A test case for functionality of a system which can be expected to happen, but will be rare and may require special work to handle appropriately from a development point of view. For example, when testing a calculator, an edge case might be ensuring that trying to divide by zero provides a correct error message.
Efficiency-oriented indicator: A performance metric related to the efficiency of use of the computational resources available to the system.

Enhancement: A requested modification or additional functionality which was not originally specified in the requirements.

Equivalence class: A group of input values which provide the same, or similar type, of output. **Equivalence class partitioning**: Separating a specific functionality into distinct equivalence classes based on input values.
Error: A status for a test case in a test run. It indicates that there is an issue with the test case itself, and the test cannot be run. For example, the preconditions indicate an impossible condition. **Error of assumption**: An error resulting from a developer or other person making an incorrect assumption about how the system should work.
Evil monkey: A **stochastic testing** method whereby malicious code or data is sent in to a system. Simulates an attacker trying to gain access or cause damage to a system.
Execution steps: The actual steps that the test will execute after ensuring that all preconditions hold.

Expected behavior : What the system is expected to do under certain circumstances. For example, after typing in "2 + 2 =" on a calculator, the expected behavior is that the system will display "4" on the screen.

Explicit boundary value : A **boundary value** explicitly called out by the system requirements. For example, requirements for an automated thermometer may state the system will turn on the DANGER light when the registered temperature is 102 degrees or over. The explicit boundary values would be 101 and 102. Contrast with **implicit boundary value**.

Exploit: A program or piece of data which takes advantage of a vulnerability. A vulnerability is "strictly theoretical" until somebody develops a way to exploit it.
Exploratory testing: An informal style of testing, where the goal is often both to learn about the system by testing it as well as find defects.

Externally consistent (requirements) : The property of having the system be consistent with requirements of other systems or of the universe. For example, having a system which mandates that the system will be able to communicate with a base on Pluto instantaneously would require faster-than-light communication, and would thus be inconsistent with the laws of this universe.

Fabrication: In **security testing**, an attack on **integrity**, which deliberately adds data, such as an attack which allows a user to create an entirely new bank account.

Failed : A status for a test case in a test run. It indicates that, while the test itself was executed without error, the system has not met at least one postcondition or expected output value, or some other unexpected behavior has taken place. In other words, the observed behavior was not the expected behavior.

Failure case : A kind of test where the expected behavior of the system is to fail in a certain way. For example, sending in a negative number to a square root function which does not support complex numbers may be expected to throw an exception. Compare to **success case**.

Fake : A kind of **test double** in which the behavior of the double is handled by the object itself, as opposed to being referenced by stub methods. These act as simpler, faster versions of the actual object, in order to make tests run faster or reduce dependencies.
Falsifying the invariant: Showing an example where an **invariant** does not hold, such as an invariant for arithmetic method that adding two positive integers should always result

in a number greater than one of the numbers, and showing that 1 + 1 = 0.

Feasible: In regard to requirements, possible to test with a realistic timeframe and allocation of resources.
Field testing: Testing that a system works while actually operating with real users.
Fishing: Catching marine animals for food or enjoyment. Has nothing to do with **security testing**. If you think it did, you are probably thinking of **phishing**.
Floating-point error: An error caused by a floating-point number being rounded or being inaccurate due to the incomplete mapping between actual decimal numbers and floating-point values. **Fork bomb**: A special kind of **bacteria** which continually forks itself, causing all CPU resources to be used up creating more copies of the fork bomb.
Fragile: A description of a test case or suite which is easily broken by small modifications to the codebase or environment.

Functional requirement : A requirement that specifies exactly what a system should do under certain circumstances. For example, "the system shall display ERROR on the console if any parameter is negative." Contrast with **non-functional requirement**.

Fuzz testing: A form of **stochastic testing** whereby random but possible data is passed in to a system to see how it responds.
Greenfield development: Writing software from scratch, and thus being able to design the entire system without worrying about previous solutions.

Grey-box testing : Testing the code as a user would, but with knowledge of the codebase in order to understand where errors might be hiding. A mixture of **white-box testing** and **black-box testing**.

Hacker : According to the Jargon File, "[a] person who enjoys exploring the details of programmable systems and how to stretch their capabilities". Often used in modern times to mean the same as **cracker**.

Happy path: The easiest path a user will take through the system, when the system is operating properly, without attempting to perform anything that is an **edge case** or **corner case**. **Identifier**: A number or string which uniquely identifies a test case, defect, or anything else.

-ility requirement : Another term for **non-functional requirement**, so named because many of these requirements use words ending in "-ility" to describe them (e.g., usability, scalability, maintainability).

Impact: When reporting defects, how the defect will affect users of the system.

Implicit boundary value : A **boundary value** that is not explicitly called out by the system requirements, but may have an impact on the system's operation. For example, code written using 32-bit signed integers (such as int in Java) has an implicit boundary at 2,147,483,647, the maximum integer size. Adding one to this will cause the number to overflow.

Impure: The opposite of **pure**, a method or function which produces at least one **side effect**. **Information Security**: The field of ensuring that computer systems successfully exhibit all three aspects of the **InfoSec Triad**.
InfoSec Triad: The three criteria that indicate a secure system—**confidentiality**,

integrity, and **availability**.
Injection attack: A kind of attack where the malicious user tries to get the victim's computer to execute arbitrary code.
Injection error: An error where the system accidentally allows arbitrary code to be executed. Leaves the system vulnerable to an **injection attack**.

Input value : A particular value which will be passed in to a test case. The distinction between this and precondition can be hazy in manual or other black-box testing; in white-box testing, values that you are passing in to the method under test (e.g. as arguments or parameters) are input values.

Integration: Connecting multiple systems or subsystems to work together.
Integration error: A type of error resulting from incompatibilities or other problems at the boundary between different systems or subsystems.
Interface error: An error where the interface to another system is defined incorrectly, or the system accessing it does not do so correctly.
Integrity: An attribute of a system, that only authorized users may write data. An element of the **InfoSec Triad**.
Interception: In **security testing**, an attack on **confidentiality**, such as eavesdropping on a network with a **packet analyzer** or on a computer with a **keylogger**.
Interior value: A value which is not a boundary value in its equivalence class.
Intermittent failure: A test failure which does not occur all of the time, but only on certain runs. Often, the reason for the intermittent failure is unknown, or it would have been fixed already.

Internally consistent (requirements) : The property of having no requirements contradict each other. For example, a requirements specification which has a requirement which states "The system shall always leave the red light on" and another requirement which states that "The system shall turn off the red light if SWITCH1 is enabled" would not be internally consistent.

Interruption: In **security testing**, an attack on **availability**, such as a **DDoS attack** or pulling the plug from a network switch.

Invariant : In **property-based testing**, a property that should always hold for a function or method. For example, a sorting method which accepts an unsorted array should always return an array with the same number of elements as the unsorted original.

Keylogger: Software which stores all keys that were pressed by the user, usually to be transmitted to, or retrieved by, an attacker.
Key Performance Indicator: A **performance indicator** that is considered of primary importance in the development of the system.
KPI: Abbreviation for **Key Performance Indicator**.
Legacy code: Code which is running in production, and was written without using modern software engineering techniques and/or has substandard automated test coverage.
Linter: A static analysis tool which informs the user of potential issues with code. **Load testing**: Running a full system with a specified amount of demand (e.g., a certain number of users or events) in order to determine how the system operates under realistic conditions. **Logic bomb**: Code within a program which executes an unauthorized

function, such as deleting all data on the first day of the month.
Logic error: An error in a program due to faulty logic being programmed.
Major: A defect of the third-highest level of severity, which indicates a severe problem but still allows the user to use the system.
Malware: Software which has pernicious and deliberate effects to the user of the software, such as a computer virus or key logger.
Mean time between failures: In **availability testing**, the mean (average) amount of time between failures on a system. Often abbreviated as **MTBF**.
Mean time to repair: In **availability testing**, the mean (average) amount of time it takes to repair a failure. Often abbreviated as **MTTR**.
Media testing: Ensuring that the media that the system is stored on (e.g., a CD-ROM or a server's hard drive) is operating correctly and has all of the data in the correct place.
Minor: A defect of a lower level of severity than **normal**, which causes only a very small problem for use of the system.

Missing data error : An error resulting from the system not receiving necessary data.
Mock: A particular kind of **test double** which keeps track of which methods on it have been called.

Modification: In **security testing**, an attack on **integrity**, which deliberately modifies data, such as an attack which allows a user to arbitrarily change the balance on their bank account.

Monkey testing : Another term for **stochastic testing**.
MTBF: An acronym for **mean time between failures**.
MTTR: An acronym for **mean time to repair**.

n nines : A way of showing what percentage of the time the system is available, based on the number of nines in that number, and assuming nines are the only significant digit. For example, a system that is available 99.92% of the time has 3 nines' availability, while a system that is available 99.999% of the time has 5 nines' availability.

Negative test case: See **failure case**.

Network error : An error which results when network connectivity is suboptimal or missing altogether. For example, an application which freezes if Internet connectivity is lost in the middle of a transaction.

Nines: See **n nines**.
Non-deterministic: A test failure which does not occur all of the time, but only on certain runs, and for unknown reasons.

Non-functional requirement : A requirement that specifies how the system should operate, without specifying specific behavior. For example, "the system shall be extensible, allowing for the addition of plug-ins" or "the system shall be usable by personnel with less than one hour of training". Contrast with **functional requirement**.

Normal: A defect of a severity which is noticeable but does not strongly hamper the user's use of the system.
Null pointer error: An error resulting from the code trying to dereference a null pointer, or access a null object.

Observed behavior : What the system actually does under certain circumstances. For example, if I type in “2 + 2 =” on a calculator, the **expected behavior** may be that I see “4”, but if I instead see “WALLA WALLA”, then “WALLA WALLA” is the observed behavior.
Off-by-one error: A specific kind of **logic error** where a program does something wrong because a value is off by one unit.

Operational testing: Testing that a system works under real-world conditions.

Output value : A particular value which will be output by a test case. The distinction between these and postconditions in manual or other black-box testing can be hazy; in white-box testing, values that are directly returned from a method are output values.

Packet analyzer: A tool which allows you to view individual packets that are being transmitted over the network.
Packet sniffing: Using a **packet analyzer** or similar software to view data being transmitted over a network.
Pair programming: Two people working at the same time on a problem on a single computer. Can be a white-box tester and developer, looking at the code together.
Pairwise testing: A particular form of combinatorial testing where you are testing for all two-way interactions.
Partitioning: See **equivalence class partitioning**.

Passed : A status for a test case in a test run. It indicates that the test was executed without error, and that the system has met all postconditions and/or expected output values—the observed behavior was equal to the expected behavior.

Passive attack: In **security testing**, an attack on a system which causes no changes to the system, such as eavesdropping on network traffic.
Pathological case: Another term for **corner case**.
Paused: A status for a test case in a test run. It indicates that the tester has started to run the test, but it is temporarily on hold for external reasons (e.g., the tester went out to lunch). **Penetration testing**: Testing the security of the system by attempting to compromise it as an unauthorized user would.
Performance indicator: A quantitative measure that indicates the level of performance of the system. For example, response time or memory usage.

Performance target : In **performance testing**, the target value for a **performance indicator**. If the indicator value meets or exceed the target, then the system has met the target. Contrast with **performance threshold**.

Performance testing: Testing that a system meets the **performance indicators** designated for it.

Performance threshold : In **performance testing**, an absolutely minimal value for a given performance indicator for the system to be considered releasable. Contrast with **performance target**.

Performant: A program which provides a high level of performance, which is usually measured by it meeting or exceeding its **KPIs**.
Phishing: A common **attack** which attempts to get personal or other sensitive information

via email or other communications.
Phone phreak: A person who explores the telephone system, usually without the permission of relevant authorities. Famous phone phreaks include John Draper and Joe Engressia.
Phreaker: Another term for **phone phreak**.

Pinning test : An automated test (usually a unit test) which checks for the current response of a system, in order to modify code without modifying existing behavior. Note that pinning tests check for the current response, *not* the correct response. Often used when refactoring or adding to **legacy code**.

Positive test case: See **success case**.

Postcondition : All conditions which need to hold true after the execution steps in order for the test to pass. For example, a postcondition after editing your user name may be that the Account Information page shows the new user name.

Precondition : All conditions which need to hold true before the execution steps of a test case can begin. For example, when testing the Account Information of a website, a precondition may be that the user is already logged in.

Principle of Least Privilege : The principle that states that users should have the minimal amount of access to the system necessary to do their jobs. For example, a developer should not (in general) have access to payroll data, and HR personnel should not have access to source code.

Profiler: A tool which allows you to measure the resource utilization and internal events (such as method calls or instantiation of objects) of a running program.

Property-based testing : A method of testing, usually automated, where many values, often pseudorandomly generated, are given as inputs, and properties of the output are tested as opposed to checking for specific values or behavior.

Pure: A method or function which does not produce any **side effects**, but simply returns the result of a computation.
QA: See **quality assurance**.
Qualitative: Not able to be expressed numerically, such as “the system shall be *extremely* cool”. Contrast with **quantitative**.
Quality assurance: Ensuring the quality of software, by a variety of methods. Software testing is an important, but not the only, part of quality assurance.
Quality attribute: Another (probably better) term for **non-functional requirement**.
Quantitative: Able to be expressed numerically, such as “the system shall respond within 500 milliseconds”. Contrast with **qualitative**.
Ransomware: A kind of **malware** which performs an unwanted action (e.g., encrypting your hard drive) and asks for money or other compensation in order to undo it.

Real time : The actual amount of time (the same kind of time as measured by a clock) taken for a process to perform some task. Also referred to as **wall clock time**. Not to be confused with “real-time system”.

Reflection: A way to determine class and method structure at run-time.
Regression failure: A failure of a previously-working piece of functionality that is caused

by (seemingly) unrelated additional functionality or defect fixes.
Regulation acceptance testing: Testing that a system meets legal or other regulatory requirements.

Report : The act of filing a defect according to the agreed-upon system for the project.
Reproduction steps: The steps necessary for reproducing a defect. Often included in defect reports so that readers of the defect will understand what causes the defect and how to reproduce it.

Requirement: A statement of what the system under development needs to do in order to be considered complete and correct.
Requirements specification: A list of requirements for a given system. The system is expected to meet all of the requirements in the requirements specification.
Response time: In **performance testing**, how quickly a system responds after user input or other event.
Rounding error: An error in a program caused by the system rounding a number.
Running: A status for a test case in a test run. It indicates that the test is currently being executed, but has not yet completed.
Sanitization: "Cleaning up" user input so that it does not contain code that would be executed by the running program or other content that would cause harm to the system.
Scripted testing: Testing with a rigid script, such as a test plan, where steps are well-defined and ordered. Contrast with **unscripted testing**.
Seam: A place in the codebase where you can modify behavior without modifying the code itself. **"Seat of your pants" testing**: Testing where the expected behavior is known to the tester through experience with the software or its domain, instead of being formally specified.
Security testing: Testing that the system meets the criteria of the InfoSec Triad, that the system is safe from unauthorized tampering and/or access.
Service-oriented indicator: A performance metric related to how the user will interact with the system.
Service Level Agreement: An agreement by a service provider, which often includes a guarantee of **availability**.
Severity: The degree to which a particular **defect** is of concern to system designers, developers, and other stakeholders.
Side effect: Anything which is not strictly the returned result of a computation, such as displaying a message or setting a variable value.
SLA: An abbreviation for Service Level Agreement.

Smart monkey : A **stochastic testing** method whereby the data passed in mimics how an actual user would use the system. For example, a smart monkey test for a word processing system might type some letters, format them, and save the file (as an actual user might), as opposed to clicking buttons purely at random.

Smoke test : A small subset of tests which is used as a gateway for further testing. **Soak test**: Another term for **stability test**.

Social engineering : Manipulating people to underhandedly cause them to perform actions that put security of a system at risk. For example, an attacker calling an administrative assistant falsely claiming that they are from the IT department and need to

know the user's password.

Spear phishing : A specific kind of **phishing** which is aimed at highly targeted individuals and is customized for them. For example, a regular phishing email may be "Dear user, please to reset your email password for linked here" whereas a spear phishing email would be "Dear Mr. Jones, please reset your SuperDuperEmail password here. Sincerely, Jane Smith, SuperDuperCo. Vice President of Security".

Spyware: A kind of **malware** which surreptitiously monitors the actions of the user of the system. **SQL injection attack**: A specific, common kind of **injection attack** where the malicious user attempts to have the system execute arbitrary SQL commands.
Stability test: A kind of **load test** where a small number of events are processed, but over a long period of time, in order to determine how stable the system is for non-trivial time periods. **Stakeholder**: Any person who has a direct interest in the successful completion, execution, or release of the system, such as customers, developers, and project managers.
Statement coverage: What percentage of statements in the code are tested, usually by unit tests. **Static testing**: Testing the system without executing any of its code. Examples would be code analysis or modeling tools.

Stochastic testing : Testing a system through the use of randomized inputs. These inputs do not have to be entirely random; for example, they can be a probability distribution of values or generated strings. Also called **monkey testing**.

Stress test : A kind of **load test** where a very high number of events are processed in a small amount of time, in order to determine how the system deals with periods of time where the system is "stressed".

Strict partitioning: Partitioning equivalence classes such that there is no overlap between the input values for any of them.
Stub: A "fake method" which can be used in unit testing to limit dependencies on other methods and focus testing on the method under test.
Success case: A kind of test case where the expected behavior of the system is to return the correct result or do the correct thing. Compare to **failure case**.
Summary (defect reporting): A brief description of the **defect** when filing a defect report. **System testing**: Testing the system as a whole, as the user (as opposed to a developer) would interact with it. Usually done in a black-box or grey-box manner.

System time : The amount of time that kernel code executes while a system performs a task. **System under test**: The system that is actually being tested.
Target: In **performance testing**, another term for **performance target**.

Tautological test case: A test case which was written so that it will always pass, for example boolean foo = true; assertTrue(foo);. This is usually done inadvertently.
Test artifact: A document or other byproduct of the testing process, such as test plans or results. **Test case**: The smallest unit of a test plan, these are the individual tests which describe what is to be tested and a check for the expected behavior of a system.
Test coverage: A general term for which aspects or parts of the system are covered by tests.

Test double : A "fake object" which can be used in unit testing to limit dependencies on other classes, which may have problems of their own, not yet be developed, or simply be

time- or resourceintensive to actually instantiate.
Test fixture: A procedure or program which puts a system into a state ready for testing.

Test hook: A “hidden” method which allows input to, or output from, the system in order to make it easier to test.

Test plan : A list of related test cases that are run together.
Test run: An actual iteration (run-through) of a test plan.
Test suite: A grouping of related test plans.

Testability: A quality of a system that specifies how easy it is to test—having well-designed and coherent methods, pure functions where possible, allowance for dependency injection, etc. **Testable code**: Code which can be easily tested in an automated fashion at multiple levels of abstraction.

Test-Driven Development (TDD) : A particular software development methodology which embraces **test-first development** along with several other tenets, such as continuous refactoring and expectation of change.

Test-First Development: Any software development methodology in which tests are written before the code that makes them pass.
Test-Unfriendly Construct: A part of the structure of the code which is difficult to test, such as a constructor, finalizer, or private method.
Test-Unfriendly Function: A functional aspect of a program which is difficult to test, such as communicating over a network or writing to disk.
Threshold: In **performance testing**, another term for **performance threshold**.
Throughput: In **performance testing**, the number of events or tasks the system can handle in a given time frame.
Total time: The total amount of time that code (user or kernel) executes, without taking into account other factors (such as time spent waiting for input). Compare to **real time**.
Traceability matrix: A two-dimensional matrix displaying test cases and requirements and indicating which test cases test which requirements.

Trapdoor : A program or piece of a program which provides secret access to a system or application. **Triage**: A meeting or other way (e.g. email discussion) of prioritizing defects.

Trivial: A defect of a severity which is not noticeable, or hardly noticeable, and only causes the smallest of issues for a user of the system.
Trojan horse: A kind of **malware** which pretends to be another kind of program in order to trick users into installing and executing it.

TUC : See **Test-Unfriendly Construct**.
TUF: See **Test-Unfriendly Function**.

Unambiguous (requirements): The property of having requirements which can be read and understood in one and only one way.

Undefined : In regards to a system specification, any area where the behavior is not specified. For example, if my entire system specification is “The first light shall turn on if the input value is less than 4”, the behavior for how the system should behave if the input value is 7 is undefined. **Unit testing**: Testing the smallest individual units of code, such as methods or functions, in a white-box manner.

Unscripted testing : Testing without a rigid script, where the tester has broad latitude to use their own volition and knowledge to determine the quality of the system or find defects. Contrast with **scripted testing**.

User acceptance testing : A particular kind of **acceptance testing** where the tester is the user of the software. It ensures that the system under test is acceptable to the user by meeting their needs.

User testing: Having an actual user of the system attempt to perform tasks, often without instruction, in order to determine how users interact with the system.
User time: The amount of time that user code executes while a system performs a task.
Utilization: In **performance testing**, the relative or absolute amount of a given computing resource (e.g., RAM, processor instructions, disk space) that is used under certain circumstances. **Validation**: Ensuring that the system meets the needs of the customer. Checking that you built the right *system*.
Verification (category of testing): Ensuring that the system operates correctly, does not crash, provides correct answers, etc. Checking that you built the system *right*.
Verification (in unit testing): Checking that a particular method was called on a **mock** object.

Virus : A kind of **malware**, often small, that replicates itself with human intervention. This intervention could be something such as clicking on a link or running a program sent to you as an attachment.

Vulnerability: A potential defect that would allow a user to compromise or otherwise gain unauthorized access to a system.
Wall clock time: Another term for **real time**.
White-box testing: Testing the code directly and with full knowledge of the code under test. **Unit testing** is an example of white-box testing.

Workaround : A method for avoiding a known **defect** while still being able to use the system. **Worm**: A kind of **malware**, often small, that replicates itself without human intervention.

YAGNI: You Ain’t Gonna Need It. A tenet of **Test-Driven Development** which states that you should not do work or add features which are not immediately necessary.

Zombie : A computer with software installed which allows unauthorized users access to it to perform unauthorized functionality. For example, a system might have a mailer program built in which will allow other users to send spam from your machine, so that the original users cannot be tracked.

Chapter 27
Acknowledgments

Thanks to everybody who has helped make this book a reality or made it into a better reality, especially:
Ross Acheson, Wil l Engler, Jake Goulding, Joel McCracken, Robbie McKinstry, Ryan Rahuba, Steve Robbibaro, Ed Wiancko, and Sheridan Zivanovich.
Special thanks go to:
Patrick Keane, who did an extremely thorough editing job and provided valuable perspective on the QA field as a whole.

Carol Nichols , who not only did an excellent job finding defects in my code, but also consistently amused me with her commit messages. She is also one of the people who has violated one of the principles of TDD with whom I have spent time, and definitely has not applied testing skills to test the integrity of this book by editing her own acknowledgment.

Tim Parenti, who found more typos and errors than I ever would have thought possible, and even better, did an excellent job fixing many of them.