

Glossary for Key Approaches

A

- acceptance criteria [[Where to start?; Write acceptance criteria](#)]
 - The criteria that must be fulfilled to satisfy a user story and be accepted by a project stakeholder
- access control [[Authorization and access control issues](#)]
 - Rules specifying who can access which parts of a database
- ACID [[SOAP overview](#)]
 - An acronym that refers to the requirement that database transactions be atomic, consistent, isolated, and durable
- ACL [[Authorization and access control issues](#)]
 - Access control list—lists which users have what kinds of access to system resources
- adapter design pattern [[The Adapter pattern defined](#)]
 - A pattern that is used to convert the interface of a class into another interface that clients expect
- @Afterall [[Test structure](#)]
 - in JUnit, an annotation for code that is to be executed after all tests are completed
- @AfterEach [[Test structure](#)]
 - In JUnit, an annotation for code that is to be executed after every test
- aggregate object [[Understanding the Iterator pattern](#)]
 - In the iterator design pattern, a collection of objects, like an array or arrayList
- agile [[TDD and agile](#)]
 - A software development methodology characterized by a collaborative approach with continuous improvement and delivery of usable software
- allocation, memory [[Why do we need to manage the memory?; Allocating memory](#)]
 - Reserving a chunk of memory for a certain bit of information
- API [[Web services, APIs, and microservices](#)]
 - Application programming interface—the communication and data sharing mechanism between two different applications or systems; these are typically more lightweight than a web service and may not depend on a protocol such as SOAP.
- arena (Python) [[Memory management in Python](#)]
 - The largest bits of memory, containing sequential blocks of memory

- argument (GraphQL) [[The structure of GraphQL queries](#)]
 - A set of key-value pairs attached to a specific field
- assert statement [[Writing test cases](#)]
 - A statement in a programming language that ensures that a certain condition is true at a given point in a program's execution; if the condition is not true, the program generates an error.
- assertEquals [[Assertions](#)]
 - An assert statement that checks that the values of an actual and expected object are the same
- assertions, type-specific [[Assertion frameworks](#)]
 - Assertions that allow the use of use tests that are specific to a particular data type; for example, when testing a String data type, one can use a test such as endsWith.
- AssertJ [[TDD tools and frameworks](#)]
 - A third-party assertion framework that extends core xUnit libraries
- assertEquals [[Assertions](#)]
 - An assert statement that checks that the actual and expected object refer to the same place in memory
- assertThrows [[Testing exceptions](#)]
 - An assert statement that checks that a given exception occurs
- attack tree [[Embrace security in testing](#)]
 - A diagram or flowchart showing how a resource might be attacked
- authentication [[Secure web services](#)]
 - Validating the identity of a client who is attempting to call a web service that accesses secure data
- authorization [[Secure web services](#)]
 - Deciding which data an authenticated user can access or modify
- automated [[Make a test plan](#)]
 - Part of a test plan that tells whether a test is automated or not
- automatic memory management [[Why do we need to manage the memory?](#)]
 - A system of managing memory that is built into the programming language and runtime; Python, Java, and JavaScript take care of memory management for you.
- automatic variables (C language) [[The C way: Allocating memory](#)]

- Another name for local variables; they are automatically freed when no longer needed.

B

- base object [[Extending behavior with composition](#)]
 - In the decorator design pattern, the object that has other objects “wrapped around it” by composition
- @BeforeAll [[Test structure](#)]
 - in JUnit, an annotation for code that is to be executed before the test class instance is created
- @BeforeEach [[Test structure](#)]
 - In JUnit, an annotation for code that is to be executed before every test
- bind variables [[Database issues](#)]
 - Variables whose values are inserted into an SQL statement (as opposed to concatenating with user input strings)
- black box testing [[Box testing](#)]
 - A form of box testing where you know the input and output but have no knowledge about the internals of an application
- blacklisting [[Input validation issues](#)]
 - Looking for key patterns (such as <script> tags in HTML) and filtering them out
- block (Python) [[Memory management in Python](#)]
 - The actual memory addresses that are going to be assigned when doing memory management
- block starting symbol (C language) [[The C way: Allocating memory](#)]
 - Block starting symbol—a special area of memory in the C runtime for static and global variables that are un-initialized at compile time
- body (SOAP) [[SOAP overview](#)]
 - A required part of a SOAP message that contains the actual XML data that the server transmits
- bottom-up approach [[What is test-driven development \(TDD\)?](#)]
 - An approach to building software that involves taking small, incremental steps that are joined together to create the product
- box testing [[Box testing](#)]
 - Testing performed so that there is a holistic approach
- broken cryptographic routine [[Cryptography issues](#)]

- A cryptographic algorithm that has been cracked, theoretically can be cracked, or has a flaw that has been identified
- brute-force attack [[Authentication and password issues](#)]
 - An attack in which the malicious actor continuously submits many passwords with the hope of finding one that works
- BSS [[The C way: Allocating memory](#)]
 - Block starting symbol—a special area of memory in the C runtime for static and global variables that are un-initialized at compile time
- buffer overflow vulnerability [[Memory management issues](#)]
 - An attack that writes, for example, a 10-byte buffer with 13 bytes of data, which may cause code to be executed
- bug bash [[Have bug bashes](#)]
 - A week in which team members focus entirely on fixing bugs; it is recommended to have these two to four times a year.
- bug book [[Document what you understand](#)]
 - A document describing security bugs found in a system
- bug priority [[Triage bugs](#)]
 - How fast a bug should be fixed
- bug severity [[Triage bugs](#)]
 - How impactful a bug is to the business
- bug triage [[Triage bugs](#)]
 - Classification of an application bug (problem) by its severity and the priority for fixing it
- build phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where the software is actually constructed

C

- call stack [[Stack memory](#)]
 - A stack that keeps track of the order of function calls
- certificate [[Communication channel issues](#)]
 - An electronic document that verifies the identity of its owner
- changing functional declaration [[Refactor to improve the design](#)]
 - In TDD, changing the name of a method to make it easier to understand what it does
- CI/CD [[Embrace security in testing](#)]

- Continuous integration/continuous deployment—frequent rebuilding and delivery of a software system
- client-side validation [[Input validation issues](#)]
 - Validating input in the browser or client application; this should be done in addition to, but not as a substitute for, server-side validation.
- code smell [[Refactor to improve the design](#)]
 - Valid code that somehow doesn't seem right; it doesn't pass the “smell test”
- compacting [[Actual removing or sweeping](#)]
 - Closing gaps in heap memory when removing objects
- composition, vs. inheritance [[Why HAS-A is better than IS-A](#)]
 - A design principle that says, if you have a choice, use composition rather than inheritance because typically, composition leads to a more flexible design
- concurrency [[Heap vs. stack memory](#)]
 - Multiple things happening at the same time
- configuration [[Configuration issues](#)]
 - The feature flags, options, and other data elements that an application needs to run but that can be optionally modified at startup or runtime
- containerized [[Embrace security in deployment](#)]
 - A situation in which software is packaged with all its dependencies and configuration information
- convention [[Writing test cases](#)]
 - An unwritten rule that software developers are expected to follow; for example, in unit testing, a convention is to keep code and test cases separate. Another example: when writing Java, variable names by convention begin with lowercase letters; class names begin with uppercase letters.
- copy, deep [[Best practices with memory](#)]
 - A copy of the object and its attributes, and their attributes, and so on; the data itself is copied.
- copy, shallow [[Best practices with memory](#)]
 - A copy of an object reference; the data referred to is not copied.
- CPU [[What is memory?](#)]
 - Central processing unit of a computer
- cross-site scripting [[Input validation issues](#)]

- An attack in which a bad actor injects a script (in a programming language) that is executed in the browser
- CVE [[Dependency issues](#)]
 - Common Vulnerabilities and Exposures—a catalog of publicly disclosed security vulnerabilities
- CWE [[Dependency issues](#)]
 - Common Weakness Enumeration—a listing of common software and hardware weaknesses with security ramifications (from <https://cwe.mitre.org>)
- cyclic reference [[Garbage collection](#)]
 - Two objects that reference each other

D

- dangling pointer [[The C way: Deallocating and reallocating memory](#)]
 - A reference to a memory area that has been deallocated
- data segment (C language) [[The C way: Allocating memory](#)]
 - Data segment—an area of memory in the C runtime for initialized global and static variables
- DBA [[Database issues](#)]
 - Database Administrator -- the person who manages a database. This may include configuration, security, updates, etc.
- deallocation, memory [[Why do we need to manage the memory?; Deallocating memory](#)]
 - Releasing memory when information is no longer needed
- decorator design pattern [[Understanding the Decorator pattern](#)]
 - A design pattern in which additional responsibilities are added to an object dynamically
- deep copy [[Best practices with memory](#)]
 - A copy of the object and its attributes, and their attributes, and so on; the data itself is copied.
- define phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where requirements are defined for the developers; writing of specifications and acceptance criteria
- DELETE operation (REST) [[REST overview](#)]
 - An operation to remove a resource
- dependency graph [[Dependency issues](#)]

- A document showing which software dependencies are vulnerable to an attack
- dependent object [[The Observer pattern defined](#)]
 - In the observer design pattern, the “many” objects in a one-to-many relationship; for example, one publisher has many subscribers, and the subscribers are dependent objects.
- deploy phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where the product is released to the customer
- design pattern, adapter [[The Adapter pattern defined](#)]
 - A pattern that is used to convert the interface of a class into another interface that clients expect
- design pattern, decorator [[Understanding the Decorator pattern](#)]
 - A design pattern in which additional responsibilities are added to an object dynamically
- design pattern, factory method [[The Factory Method pattern](#)]
 - A pattern that allows programmers to decouple the process of creating objects from the clients that use those objects; it defines an interface for creating an object but lets subclasses decide which classes to instantiate.
- design pattern, iterator [[Understanding the Iterator pattern](#)]
 - A design pattern that provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- design pattern, observer [[The Observer pattern defined](#)]
 - A pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- design pattern, strategy [[What are design principles?](#)]
 - A pattern that separates out an object’s behavior in a flexible and extensible way
- design phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where the team builds mockups and prototypes
- design principle, open-closed [[Understanding the open-closed principle](#)]
 - A principle that says classes should be open for extension but closed for modification
- design principle, single responsibility [[The single responsibility principle](#)]
 - A principle that says a class should have only one reason to change
- design principles [[What are design principles?](#)]
 - General guidelines for developing programs, as opposed to design patterns, which are specific design solutions for common object-oriented programs

- development sprint [[Getting bugs fixed](#)]
 - A specific block of time in which developers are to complete a set of goals; this is part of the agile development methodology.
- DevOps [[Roles and responsibilities](#)]
 - Practices for integrating and automating development and information technology operations
- DevSecOps [[Break what you build](#)]
 - A methodology for integrating and automating a system's development, security, and operations
- DoS [[Security testing](#)]
 - Denial of service (DoS)—an attack in which a bad actor uses automated scripts to flood a server with traffic so it cannot handle legitimate requests
- DS [[The C way: Allocating memory](#)]
 - Data segment—an area of memory in the C runtime for initialized global and static variables
- dummy object [[Test doubles](#)]
 - A test double object created just to make code compile
- dynamic analysis [[Implement best practices](#)]
 - Analysis of run-time performance of a system: time and memory usage, etc.
- dynamic memory allocation [[Why do we need to manage the memory?; Allocating memory](#)]
 - Memory allocation that occurs during runtime

E

- endurance testing [[Performance testing](#)]
 - A form of performance testing done to make sure an application can handle the expected load over a long period of time
- envelope [[SOAP overview](#)]
 - An XML file that contains the multiple parts of a SOAP message
- error messaging, information disclosure through [[Error handling issues](#)]
 - A situation in which an error message provides information useful to an attacker, such a stack trace or an authentication error that tells that a username is valid but the password is not
- error, out of memory [[Out of Memory error](#)]
 - An error that occurs when the system cannot allocate a requested memory block anymore

- error, stack overflow [[Stack memory](#)]
 - An error that occurs when a program exceeds the size limit of stack memory
- escaping references [[Best practices with memory](#)]
 - The situation when an object is accessed unintentionally through another function
- expected result [[Make a test plan](#)]
 - Part of a test plan that describes what the outcome will be for a given result
- extract function refactoring [[Refactor to improve the design](#)]
 - The technique of moving out a certain functionality to make a method's implementation align with its intention

F

- factory method design pattern [[The Factory Method pattern](#)]
 - A pattern that allows programmers to decouple the process of creating objects from the clients that use those objects; it defines an interface for creating an object but lets subclasses decide which classes to instantiate.
- fake object [[Test doubles](#)]
 - A test double method that provides some functionality but is not production ready—it is better than having nothing at all.
- faking it [[TDD and agile](#)]
 - In TDD, writing code only to pass the test and no more
- fault (SOAP) [[SOAP overview](#)]
 - An optional part of a SOAP message containing information about any errors that might occur during message processing
- field name (GraphQL) [[The structure of GraphQL queries](#)]
 - Each field represents a unit of data you are asking for.
- fragmentation, heap [[Deallocating memory](#)]
 - The situation where gaps occur in the heap when memory is allocated and deallocated
- free (C language) [[The C way: Deallocating and reallocating memory](#)]
 - A function that deallocates memory
- functional declaration, changing [[Refactor to improve the design](#)]
 - In TDD, changing the name of a method to make it easier to understand what it does

G

- garbage collection [[Deallocating memory](#)]

- A process by which the runtime system keeps track of which items in memory are no longer needed and deletes them
- GET operation (REST) [[REST overview](#)]
 - An operation to retrieve a resource
- given [[Write acceptance criteria](#)]
 - Part of an acceptance criteria that specifies the precondition or beginning state
- GraphQL [[GraphQL overview](#)]
 - A query language for APIs; it is a syntax that describes how to ask for data and is generally used to load data from a server to a client.
- gray box testing [[Box testing](#)]
 - A form of box testing where scenarios examine the interaction between the outside and inside of the box; this includes such things as integration testing.

H

- Hamcrest [[TDD tools and frameworks](#)]
 - A third-party assertion framework that extends core xUnit libraries
- happy path scenario [[Manual testing](#)]
 - A scenario with a successful result
- hasNext method [[Using the Iterator pattern](#)]
 - A method in the iterator design pattern that determines whether there are more items to iterate over in the aggregate object
- HATEOAS [[HATEOAS overview](#)]
 - Hypermedia as the Engine of Application State—the principle that specifies that a RESTful API should provide enough information to the client to interact with the server.
- header (SOAP) [[SOAP overview](#)]
 - An optional part of a SOAP message that contains message attributes
- heap fragmentation [[Deallocating memory](#)]
 - The situation where gaps occur in the heap when memory is allocated and deallocated
- heap memory [[Heap memory](#)]
 - A part of RAM used for storing values that need to be accessed throughout the entire application; it also holds larger values that might not fit in the limited stack memory.

I

- infected file upload [[File and I/O issues](#)]

- A file that a malicious actor uploads; the file contains some form of malware.
- information disclosure through error messaging [[Error handling issues](#)]
 - A situation in which an error message provides information useful to an attacker, such a stack trace or an authentication error that tells that a username is valid but the password is not
- inheritance [[Revisiting inheritance](#)]
 - A process for passing on attributes and methods from other existing classes to a new class
- inheritance, vs. composition [[Why HAS-A is better than IS-A](#)]
 - A design principle that says, if you have a choice, use composition rather than inheritance because typically, composition leads to a more flexible design
- initialization vector [[Cryptography issues](#)]
 - A random (or pseudorandom) number used to prevent malicious actors from repeating or mimicking previous valid transactions
- injection attack [[Input validation issues](#)]
 - An attack in which a bad actor introduces input that is executed as code
- input validation issues [[Input validation issues](#)]
 - Security problems caused by improper validation of input, allowing malicious input into the system
- integration testing [[Integration testing](#)]
 - Testing that focuses on the interaction between components at lower layers of the application, seeing how the system reacts to certain actions; these tests have some knowledge of how the system works internally.
- interface (object-oriented) [[Trying interfaces](#)]
 - A structure that declares a set of methods for a class to implement; the methods are not implemented—the method signatures specify a capability for a class to implement. Every class that implements the interface must provide a body for all the methods.
- iteration pattern as language feature [[The Iterator pattern as language feature](#)]
 - Language features that hide an iterator design pattern behind the scenes and make it easy to iterate through collections of values; in Java, this is the enhanced for; in Python, it is the for/in statement; in JavaScript, it is the for/of statement.
- iterator design pattern [[Understanding the Iterator pattern](#)]
 - A design pattern that provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- iterator object [[Understanding the Iterator pattern](#)]

- In the iterator design pattern, an object that knows how to iterate over an aggregate object

J

- Java EE [[Create web service](#)]
 - Java Enterprise Edition—a version of Java with specifications for enterprise capabilities such as web services and distributed computing.
- JAX-WS [[Create web service](#)]
 - Java API for XML-Based Web Services
- Jira [[Report bugs](#)]
 - A tool used for tracking features, user stories, tasks, and bug reports
- JSON [[Benefits of REST](#)]
 - JavaScript Object Notation. The json.org website describes it as “a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language...”
- JUnit [[xUnit and jUnit](#)]
 - The xUnit framework, translated to Java

L

- language transparency [[Advantages of web services](#)]
 - The capability to have a web service in one language with the client written in a totally different language; they communicate through a common language, such as XML or JSON.
- latency [[Considerations of web services](#)]
 - The amount of time it takes once a request is made to receive a response
- latest result [[Make a test plan](#)]
 - Part of a test plan that gives the most recent result of executing a test (pass or fail)
- load testing [[Performance testing](#)]
 - Testing that checks an application’s ability to perform under anticipated user loads; the objective is to identify the maximum operating capacity of an application.
- log aggregation attack [[Logging and output issues](#)]
 - An attack in which someone with access to system logs can find sensitive information in error logs
- loose coupling (REST) [[Benefits of REST](#)]
 - Systems designed so that changes and enhancements to web services don’t break clients that are already using them

- loosely coupled [[Understanding the observer pattern](#); [The Observer pattern and loose coupling](#)]

- A situation in which objects interact with one another but don't know a lot about each other

M

- main memory [[What is memory?](#)]
 - Analogous to human short-term memory; when you shut down a computer, the main memory gets erased. This is called volatile.
- malloc (C language) [[The C way: Allocating memory](#)]
 - A function that allocates heap memory
- man in the middle [[Communication channel issues](#)]
 - An attack in which a bad actor intercepts, and possibly alters, the communication channel between two parties
- manual memory management [[Why do we need to manage the memory?](#)]
 - A system of managing memory in which the developer has to allocate and deallocate memory; this is the way C and C++ work.
- manual testing [[Manual testing](#)]
 - Testing that follows the steps as a user performing workflows in the application; its goal is to uncover any issues in application functionality and usability.
- matcher [[Assertion frameworks](#)]
 - In Hamcrest, a predicate (condition) that returns true if an object matches what the condition specifies
- Maven [[What you should know](#)]
 - A framework for building Java applications
- memory allocation [[Why do we need to manage the memory?](#); [Allocating memory](#)]
 - Reserving a chunk of memory for a certain bit of information
- memory allocation, dynamic [[Why do we need to manage the memory?](#); [Allocating memory](#)]
 - Memory allocation that occurs during runtime
- memory allocation, static [[Why do we need to manage the memory?](#); [Allocating memory](#)]
 - Memory allocation that occurs before a program is executed
- memory deallocation [[Why do we need to manage the memory?](#); [Deallocating memory](#)]
 - Releasing memory when information is no longer needed
- memory, heap [[Heap memory](#)]

- A part of RAM used for storing values that need to be accessed throughout the entire application; it also holds larger values that might not fit in the limited stack memory.
- memory leak [[Garbage collection](#)]
 - A situation where memory should be deallocated but has not and therefore cannot be reached anymore
- memory management [[Why do we need to manage the memory?](#)]
 - The allocation and releasing of memory
- memory management, automatic [[Why do we need to manage the memory?](#)]
 - A system of managing memory that is built into the programming language and runtime; Python, Java, and JavaScript take care of memory management for you.
- memory management, manual [[Why do we need to manage the memory?](#)]
 - A system of managing memory in which the developer has to allocate and deallocate memory; this is the way C and C++ work.
- memory profiler [[Best practices with memory](#)]
 - A program that lets you see how memory is used when a program runs
- memory, stack [[Stack memory](#)]
 - A stack that stores the variables created by functions; when entering a function, memory is allocated on the stack. When the function is done, the stack memory is deallocated.
- microservice [[Web services, APIs, and microservices](#)]
 - Fully contained, individual components that communicate with each other in calling clients
- mob testing session [[Have bug bashes](#)]
 - A one-hour interval when team members get together to test particular features in an application
- mock [[Mocking](#)]
 - An object that simulates the expected behavior of some external dependency (as opposed to its state)
- mocking framework [[Mocking](#)]
 - A framework that complements xUnit and can be integrated into the regular TDD setup
- Mockito [[Mocking](#)]
 - An API that is the most popular mocking framework among Java developers
- multi-threading [[Memory management in Python](#)]
 - An approach to writing code that executes multiple threads (tasks) concurrently

- mutation (GraphQL) [[The structure of GraphQL queries](#)]
 - A GraphQL operation for modifying server-side data

N

- next() method [[Using the Iterator pattern](#)]
 - A method in the iterator design pattern that retrieves the next item in the aggregate object
- nonce [[Cryptography issues](#)]
 - A random (or pseudorandom) number used to prevent malicious actors from repeating or mimicking previous valid transactions

O

- observer design pattern [[The Observer pattern defined](#)]
 - A pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- on-the-wire attack [[Communication channel issues](#)]
 - An attack on a communication channel in which the bad actor has access to the network of either the client or server
- open-closed design principle [[Understanding the open-closed principle](#)]
 - A principle that says classes should be open for extension but closed for modification
- OpenAPI [[Document an API](#)]
 - A standard, language-agnostic interface to HTTP APIs
- out of memory error [[Out of Memory error](#)]
 - An error that occurs when the system cannot allocate a requested memory block anymore
- OWASP [[Configuration issues](#)]
 - Open Worldwide Application Security Project —described by their website as “a nonprofit foundation that works to improve the security of software.”

P

- partial failure [[Considerations of web services](#)]
 - The condition in which a component like a server or network fails to respond to a request
- password hashing algorithm [[Authentication and password issues](#)]
 - An algorithm that encrypts a password
- pen test [[Break what you build](#)]
 - penetration test—a simulated attack on a system to evaluate its security

- performance testing [[Performance testing](#)]
 - Testing to benchmark how a system performs under load
- plan phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where business requirements and use cases are gathered from the customer
- pointer (C language) [[The C way: Allocating memory](#)]
 - A reference to a memory address where a value is stored
- POJO [[Create web service](#)]
 - Plain old Java object—an object that has no special restrictions beyond those imposed by the Java programming language.
- polymorphism, replace conditional with [[Refactor to improve the design](#)]
 - In TDD, a more complex refactoring technique requiring changes at the class level
- pool (Python) [[Memory management in Python](#)]
 - A subdivision of an arena containing blocks of memory of a specific same size
- POST operation (REST) [[REST overview](#)]
 - An operation to create a resource
- Postman [[Consume a RESTful API via Postman](#)]
 - A tool used to test an API
- profiler, memory [[Best practices with memory](#)]
 - A program that lets you see how memory is used when a program runs
- PUT operation (REST) [[REST overview](#)]
 - An operation to update a resource

Q

- QA [[What is quality assurance?](#)]
 - Quality assurance—a systematic process used to determine whether a product meets specifications
- QA deliverables [[Create a test strategy](#)]
 - Part of the test strategy that describes what QA will provide to the team; this may include test plans for each feature, issues reported for bugs, and a release process document.
- quality assurance [[What is quality assurance?](#)]
 - Quality assurance—a systematic process used to determine whether a product meets specifications

- query document [[The structure of GraphQL queries](#)]
 - A string you use in GraphQL that is sent to the server to process and request data
- query (GraphQL) [[The structure of GraphQL queries](#)]
 - Part of the GraphQL type system that allows you to get information about specific fields from objects

R

- rainbow table [[Cryptography issues](#)]
 - A precomputed table of results of a cryptographic hash algorithm
- RAM [[What is memory?](#)]
 - Random-access memory; see main memory.
- range-act-assert pattern [[Test structure](#)]
 - A TDD test structure that arranges the elements required to test, acts on the elements, and then asserts that the result is as expected
- realloc (C language) [[The C way: Deallocating and reallocating memory](#)]
 - A function that re-allocates memory
- red-green-refactor cycle [[The iterative red-green-refactor cycle](#)]
 - The process of taking a failing test case to writing the code to pass the test case; this may involve refactoring to, for example, get rid of test cases that are no longer necessary.
- refactoring [[What is test-driven development \(TDD\)?](#); [Refactor to improve the design](#)]
 - Improving the internal structure of code to make it easier to understand or modify without changing its external behavior
- refactoring, extract function [[Refactor to improve the design](#)]
 - The technique of moving out a certain functionality to make a method's implementation align with its intention
- reference counter [[Garbage collection](#); [Memory management in Python](#)]
 - A counter that tells how many references exist to an object on the heap
- references, escaping [[Best practices with memory](#)]
 - The situation when an object is accessed unintentionally through another function
- replace conditional with polymorphism [[Refactor to improve the design](#)]
 - In TDD, a more complex refactoring technique requiring changes at the class level
- resolver (GraphQL) [[The structure of GraphQL queries](#)]
 - Part of the GraphQL type system that is responsible for getting data to the client

- REST [[Web services overview](#); [REST overview](#)]
 - Representational State Transfer—a service that uses a web protocol, HTTP (HyperText Transfer Protocol), to access resources
- RESTful API [[REST overview](#)]
 - An API that uses REST for communicating between client and server
- risk register [[Document what you understand](#)]
 - A document detailing a system’s security risks and how to mitigate those risks
- ROM [[What is memory?](#)]
 - Read-only memory—slower than RAM but is non-volatile and doesn’t disappear when the computer is shut down; usually used for the logic needed to start a computer and operating system

S

- sad path scenario [[Manual testing](#)]
 - A scenario that returns errors or does not have results
- salt [[Cryptography issues](#)]
 - Random data that is part of a cryptographic hash; this prevents bad actors from building tables of cryptographic hash results.
- sandboxing [[File and I/O issues](#)]
 - Running a program in its own environment, separate from other programs
- scenario [[Make a test plan](#)]
 - Part of a test plan that explains the steps or actions that will be executed
- scenario, happy path [[Manual testing](#)]
 - A scenario with a successful result
- scenario, sad path [[Manual testing](#)]
 - A scenario that returns errors or does not have results
- schema (GraphQL) [[The structure of GraphQL queries](#)]
 - Part of the GraphQL type system that defines a set of types
- scope of testing [[Create a test strategy](#)]
 - Part of the test strategy that describes what types of tests exist for this project, which tools are used to write each type, and who owns them
- scripting attack [[Input validation issues](#)]

- An attack in which a bad actor injects a script (in a programming language) that is executed in the browser
- scrum [[Embrace security in design](#)]
 - A team collaboration approach to software design
- SDLC [[Embrace security in design](#); [Get involved throughout the SDLC](#)]
 - Software development life cycle—a process that produces high-quality software in the shortest amount of time; it includes detailed steps for how to effectively develop, change, and maintain a software system.
- secondary storage [[What is memory?](#)]
 - Analogous to human long-term memory. This storage persists even when the computer is shut down. An example is files and programs on your hard drive. This is called non-volatile.
- secret storage service [[Internal data management issues](#)]
 - A way to store passwords or other authentication information securely
- security testing [[Security testing](#)]
 - Testing performed to reveal flaws or vulnerabilities that can be exposed by users, causing an app to behave in unexpected ways or stop it from working
- selection set [[The structure of GraphQL queries](#)]
 - A set of key-value pairs attached to a specific field
- server-side validation [[Input validation issues](#)]
 - Validating user input when it is received by the server; this should always be done, and be done before the input is processed further.
- service, denial of [[Security testing](#)]
 - Denial of service (DoS)—an attack in which a bad actor uses automated scripts to flood a server with traffic so it cannot handle legitimate requests
- session [[Session management issues](#)]
 - Data that contains application state; it may be necessary when using HTTP, which is a stateless protocol.
- session fixation attack [[Session fixation](#)]
 - An attack in which a bad actor uses a pre-authentication token that was not updated to a secure token after logging in
- session hijacking [[Session hijacking](#)]
 - A vulnerability caused when a bad actor can predict a session token and use it to access a user's session data

- set up-test-tear down pattern [[Test structure](#)]
 - A TDD test structure that sets up a fixture, does the test, and then destroys any older objects or re-initializes any other variables modified in previous tests; this can sometimes make test cases run faster.
- shallow copy [[Best practices with memory](#)]
 - A copy of an object reference; the data referred to is not copied.
- single responsibility design principle [[The single responsibility principle](#)]
 - A principle that says a class should have only one reason to change
- soak testing [[Performance testing](#)]
 - A form of performance testing done to make sure an application can handle the expected load over a long period of time
- SOAP [[Web services overview](#); [SOAP overview](#)]
 - Simple Object Access Protocol—a service that uses XML (Extensible Markup Language) to send messages
- SoapUI [[Consume a SOAP web service via SoapUI](#)]
 - A tool for testing SOAP web services
- source control [[Implement best practices](#)]
 - Software that helps keep track of changes and updates to the source code for a project
- spoofing attack [[Communication channel issues](#)]
 - An attack in which a bad actor presents false data on the network; for example, giving a false IP (Internet Protocol) address
- spy [[Test doubles](#)]
 - A test double variable used to check that, for example, some event has occurred or count how many times something has happened
- SQL injection [[Input validation issues](#); [Security testing](#)]
 - A technique in which a user enters a value that is part of an SQL command to alter the query that should be running and change how it works
- SSL [[Communication channel issues](#)]
 - Secure Sockets Layer—a now-deprecated predecessor of TLS
- stack, call [[Stack memory](#)]
 - A stack that keeps track of the order of function calls
- stack memory [[Stack memory](#)]

- A stack that stores the variables created by functions; when entering a function, memory is allocated on the stack. When the function is done, the stack memory is deallocated.
- stack overflow error [[Stack memory](#)]
 - An error that occurs when a program exceeds the size limit of stack memory
- stateless [[REST overview](#)]
 - The condition in which a server will not remember or store any state about the client that made the call
- static analysis [[Implement best practices](#)]
 - Analysis of the source code for such things as performance, memory usage, unused variables, etc.
- static memory allocation [[Why do we need to manage the memory?; Allocating memory](#)]
 - Memory allocation that occurs before a program is executed
- strategy design pattern [[What are design principles?](#)]
 - A pattern that separates out an object's behavior in a flexible and extensible way
- stress testing [[Performance testing](#)]
 - A form of performance testing that tests an application under extreme workloads; the objective is to identify the breaking point of an application.
- stub [[Test doubles](#)]
 - A test double method that returns a value, feeding desired inputs into the tests rather than reflecting real behavior
- subject object [[The Observer pattern defined](#)]
 - In the observer design pattern, the “one” object in a one-to-many relationship; for example, one publisher has many subscribers, and the publisher is the subject object.
- subscription (GraphQL) [[The structure of GraphQL queries](#)]
 - A GraphQL operation that allows for notification of changes to data in real time
- sweeping [[Actual removing or sweeping](#)]
 - The actual removal of objects on the heap during garbage collection

T

- TDD [[Small steps to great things](#)]
 - Test-driven development
- test-code-refactor [[What is test-driven development \(TDD\)?](#)]

- A development cycle in which a programmer first writes a failing test case and then writes the code that passes that test case
- test double [[Test doubles](#)]
 - A proxy to represent an external dependency when doing tests in TDD
- test-driven approach [[What is test-driven development \(TDD\)?](#)]
 - An approach to building software in which, given requirements, a programmer writes a set of unit test cases that will drive the rest of development
- test fixture [[Test structure](#)]
 - In TDD, a block of code that represents a certain state of the system for test cases to run
- test management [[Create a test strategy](#)]
 - Part of the test strategy that describes what resources are needed to carry out testing in terms of tooling, environments, supported platforms and versions, and test data
- test phase [[Get involved throughout the SDLC](#)]
 - The part of the SDLC where QA has the major role: testing that the software meets specifications
- test plan [[Make a test plan](#)]
 - A QA plan for testing a feature
- test strategy [[Create a test strategy](#)]
 - A plan that describes how a product will be tested; it usually consists of an introduction, references (relevant project links), QA deliverables, test management, and scope of testing.
- testing, black box [[Box testing](#)]
 - A form of box testing where you know the input and output but have no knowledge about the internals of an application
- testing, box [[Box testing](#)]
 - Testing performed so that there is a holistic approach
- testing, endurance [[Performance testing](#)]
 - A form of performance testing done to make sure an application can handle the expected load over a long period of time
- testing, gray box [[Box testing](#)]
 - A form of box testing where scenarios examine the interaction between the outside and inside of the box; this includes such things as integration testing.
- testing, integration [[Integration testing](#)]

- Testing that focuses on the interaction between components at lower layers of the application, seeing how the system reacts to certain actions; these tests have some knowledge of how the system works internally.
- testing, load [[Performance testing](#)]
 - Testing that checks an application's ability to perform under anticipated user loads; the objective is to identify the maximum operating capacity of an application.
- testing, manual [[Manual testing](#)]
 - Testing that follows the steps as a user performing workflows in the application; its goal is to uncover any issues in application functionality and usability.
- testing, performance [[Performance testing](#)]
 - Testing to benchmark how a system performs under load
- testing, scope of [[Create a test strategy](#)]
 - Part of the test strategy that describes what types of tests exist for this project, which tools are used to write each type, and who owns them
- testing, security [[Security testing](#)]
 - Testing performed to reveal flaws or vulnerabilities that can be exposed by users, causing an app to behave in unexpected ways or stop it from working
- testing, soak [[Performance testing](#)]
 - A form of performance testing done to make sure an application can handle the expected load over a long period of time
- testing, stress [[Performance testing](#)]
 - A form of performance testing that tests an application under extreme workloads; the objective is to identify the breaking point of an application.
- testing, UI automation [[UI automation testing](#)]
 - Testing of a user interface (UI) accomplished by running scripts rather than manually entering input
- testing, white box [[Box testing](#)]
 - A form of box testing that focuses on the internals of the application and what is happening at the code or system level
- text segment (C language) [[The C way: Allocating memory](#)]
 - An area of memory for compiled code that is being stored
- then [[Write acceptance criteria](#)]
 - Part of an acceptance criteria that specifies the expected outcome of a scenario

- thread [[Heap vs. stack memory](#)]
 - A part of execution within a concurrent application; every thread has its own stack.
- TLS [[Communication channel issues](#)]
 - Transport Layer Security—a cryptographic protocol on the transport layer of the communications network
- token [[Session management issues](#)]
 - A unique identifier generated at the start of a session
- top-down approach [[What is test-driven development \(TDD\)?](#)]
 - An approach to building software that starts from the “big picture” and breaks it down into smaller steps
- triage, bug [[Triage bugs](#)]
 - Classification of an application bug (problem) by its severity and the priority for fixing it
- Truth [[Assertion frameworks](#)]
 - A third-party assertion framework that extends core xUnit libraries
- two-way certificate pinning [[Internal data management issues](#)]
 - Having a client and server authenticate each other
- type-specific assertions [[Assertion frameworks](#)]
 - Assertions that allow the use of use tests that are specific to a particular data type; for example, when testing a String data type, one can use a test such as endsWith.

U

- UI automation testing [[UI automation testing](#)]
 - Testing of a user interface (UI) accomplished by running scripts rather than manually entering input
- unit test [[What is test-driven development \(TDD\)?](#)]
 - Code that tests a specific part of a program
- unit testing [[Implement best practices](#)]
 - Tests for individual units (parts) of source code
- update method [[The Observer pattern defined](#)]
 - In the observer design pattern, this is a method that the subject object calls to update its dependent objects.
- URI [[REST overview](#)]
 - Uniform Resource Identifier, accessed by web links

- use case [[Where to start?](#)]
 - An informal description, written in a natural language, that describes a feature of a software product
- user persona [[Embrace security in testing](#)]
 - A fictional character representing a user of a system
- user story [[Where to start?](#)]
 - An informal description, written in a natural language, that describes a feature of a software product

V

- virtual memory [[What is memory?](#)]
 - Secondary storage used to manage data that cannot fit into main memory

W

- waterfall [[Embrace security in design](#)]
 - A top-down approach to software design
- web service [[Web services overview](#)]
 - A service that allows different systems to talk to each other over the internet
- when [[Write acceptance criteria](#)]
 - Part of an acceptance criteria that specifies the input or action of a scenario
- white box testing [[Box testing](#)]
 - A form of box testing that focuses on the internals of the application and what is happening at the code or system level
- whitelisting [[Input validation issues](#)]
 - Ensuring that input matches a specific pattern before being accepted
- WSDL [[SOAP overview](#)]
 - Web Services Description Language—a language used to give the client information on what services a web service can offer

X

- XSS [[Input validation issues](#)]
 - An attack in which a bad actor injects a script (in a programming language) that is executed in the browser
- xUnit [[xUnit and jUnit](#)]
 - A framework for unit testing

Z

- zero-day [[Memory management issues](#)]
 - A vulnerability that was previously unknown; the developers have had “zero days” to fix it before it is exploited.