**Guidelines for shared code**

* All engineers must reuse existing libraries, unless they have very good reason not to based on a project specific need
* Shared code must be reusable and self contained as possible

Chromium and Android are open source

**Interviewing SETs**

Implement the function account(void \* s) that returns count of the number of As in a String

What is function used for?

Why are we building it?

Does the function prototype even look correct?

How to validate right behaviors?

What is the encoding of the incoming strings : ASCII, UTF-8

The function name is weak, should be CamelCased, ,more descriptive or are there other standard naming practices to be followed here?

What is return type?

Void\* is dangerous. We should give it an appropriate type like char \*

What counts as a A? Should it count lowercase As too?

Does this function already exist in standard library

Think about scale – may be return type should be 64 bit integer

Think about re-use – Why does this function count only As? It is probably good idea to parameterize this so that an arbitrary character can be counted?

Think about safety – Are these pointers coming from trusted sources

Optimization – can we assume the data is coming in is already sorted, What is the texture of the input data? Is it most often all As, is It ,most often a mix of all characters, or is it only A and spaces.

Safety – add a length parameter to help ensure the code doesn’t walk off the end of string. Check the length parameter’s value for sanity. Should we be doing this check in try – catch block. If there is a possibility that buffer can be modified by some other thread while this function executes. There may be thread safety issue.

**Good candidate will do following –**

* Have little trouble with basics of coding this solution? While doing so, they do not have trouble rewriting or fumbling over basic syntax issues?
* Show no sign of misunderstanding pointers or allocating anything unnecessarily
* Perform some input validation upfront to avoid pesky crashes
* Understand the runtime or Big O19 of their code. Anything other than linear here shows creativity
* Produce code is clear and easily readable by others
* Walk through their code with single test input of A or null
* Consider big integer for counters and avoid overflow when someone uses this function to count A in an insanely long string
* Write code to shared/distribute the counting computation. Some candidates unfamiliar with MapReduce can come with their own variant
* Walk through their code with many different inputs and fix every bug that they find.
* Test their function before being asked
* Continue trying to optimize the solution until asked to stp
* Focus on generating interesting test data. Consider how to run large tests and where to get real world test data
* Want to spin up concurrent threads executing this function to look for cross talk, deadlock and memory leaks
* Build long running tests & ensure they continue to work over the long haul
* Remain interested in coming up with test cases and interesting approaches to test data generation, validation and execution
* Where are weak points in software
* What are security, privacy, performance, reliability , usability , compatibility , globalization or other concerns
* Do all primary user scenarios work as expected? Do they work for all international audiences?
* Does the product interoperate with other products (hardware and software)?
* In the event of problem, how good are the diagnostics?

**ACC**

It has three views of a product corresponding to

1. *Adjectives* and *adverbs* that describe the product’s purpose and goals
2. *Noun* that identify the various parts & feature of the product
3. *Verbs* that indicate what product actually does

**A is for “Attribute”**

Attributes are the adjectives of the system. They are qualities and characteristics that promote the product and distinguish it from competitor?

* From a testing perspective, we just need to capture and label these things so we can ensure they are accounted for when we test it.
* Chrome for example is held to be fast, secure, stable and elegant.
* We want to get to a point where we can attach test cases to these labels so that we know how much testing we have done to show that chrome is fast, secure.

When starting test planning or ACC, it is important to first identify why the product is important to users and to the business?

Why are we building this thing?

What core values does it deliver?

Why is it interesting to customer?

Few tips on coming up with attributes for your own projects

* **Keep it simple:** if it takes more than an hour or two, you are spending too long on this step
* **Keep it accurate :** Make sure it comes from a document or marketing information that your team already accepts as a truth
* **Keep it moving :** Don’t worry if you missed something – if it’s not obvious later, it probably wasn’t that important anyway
* **Keep it short :** No more than a dozen is good target

Attributes are used to figure out what the product does to support the core reason for the product’s existence and to surface these reason to testers so they can be aware of how the testing they do impacts the application’s ultimate reason for existence.

Attributes are there for the taking. If you have trouble enumerating them in anything more than a few minutes, then you don’t understand your product well enough to be an effective tester

**C is for “Component”**

This is noun of the system and are building blocks that together constitute the system. They are the shopping card and the checkout feature of an online store. They are the formatting and printing features of a word processor. They are core chunks of the code that make the software what it is. Indeed, they are the very things that testers are tasked with testing.

* Components are generally easy to identify and often already cast in a design document. For large system, they are big boxes in an architectural diagram and often appear in labels in bug database or called explicitly in project pages and documents.
* For smaller projects, they are the classes and objects in the code. In every case, go & ask each developer: “What component are you working on? And you will get list without having to do much else.
* If you are struggling coming up with components, then you seriously lack familiarity with your product and you should spend some time using it to get to the level of power user quickly.

**C is for “Capability”**

This is verb of the system, they represent the actions the system performs at the command of the user. They are responses to input, answer to queries and activities accomplished on behalf of the user.

* Chrome for example, has the capabilities of rendering a web page and playing a flash file. Chrome can synchronize between client and download a document. All these capabilities and these and many more represents the full set of capabilities of the chrome web browser.
* A shopping app, on other hand has the capability of performing a product search and completing a sale. If an application can perform a task, then this task is labeled as one of its capabilities.
* Capabilities lie at the intersection of attributes and components. Components perform some function to satisfy an attribute of the product and the result of this activity is providing a capability to a user. Chrome renders a web page fast. Chrome plays a flash file securely.
* If your product does something that isn’t covered by the intersection of an attribute and a component, it probably does not matter and warrants raising the question as to why we are bothering to have it in the product.
* A capability that does not serve a core value to the product sounds very much like fat that can be trimmed and represents potential points of failure with little to be gained.
* If any single engineer engineer on a project understands his product value propositions to its user, that someone is a tester.

Some example of capabilities for an online shopping site –

* **Add/Remove items from the shopping cart :** This is a capability of the cart component when trying to meet the intuitive UI attribute.
* **Collect Credit card and verification data :** this is a capability of the cart component when trying to meet the convenient attribute and the integrated(for instance integrated with payment system) attribute
* **Process monetary transactions through HTTPS :** This is capability of the cart component when trying to meet the secure attributes
* **Provide suggestion to shoppers based on the product they are viewing :**  This is capability of the search component when trying to meet the convenient attribute.
* **Calculate Shipping cost :** Thisis acapability of the UPS Integration component when trying to meet the Fast and Secure attribute.
* **Display available Inventory :** This is a capability of the search component when trying to meet the convenient and accurate attribute
* **Defer a purchase for a later date :** This is a capability of the card component when trying to meet the convenient attribute
* **Search for items by keyword, SKU and category :** This is a capability of the search component when trying to satisfy the convenient and accurate attribute. In general, we prefer to treat each search category as a separate capability.

The entire idea is to list, quickly and succinctly , the most important capabilities of the system that need to be verified to be in working order.

Capabilities are user oriented and written to convey a user’s view of what the system does. They are also far more numerous than either attributes or components.

It should in sum total describe everything the system is capable of doing and therefore be greater in number based on the feature richness and complexity of the application in question.

Capabilities are testable, this is the primary reason we write them in an active voice. They are verbs because they require action on our part, specifically that we will write test cases to determine whether each capability is implemented correctly and that the user will find the experience useful.

Capabilities are meant to be atomic actions. As single capability can describe any number of actual user cases.

The capability is that user can shop. The test case specifies what they shop for. This is general concept of actions the software can take or a user can request. They imply tests and values but are not tests themselves.

Writing good capabilities requires some discipline. Here are some properties we have found to be useful guidance for writing capabilities –

1. A capability should be written as an action and covey the sense of the user accomplishing some activity using the application under test
2. A capability should provide enough guidance for a tester to understand what variables are involved in writing test cases for the behavior it describes. For example , process monetary transactions using https requires that the tester understand what types of monetary transactions the system is capable of performing and a mechanism to validate whether the transaction the system is capable of performing and a mechanism to validate whether the transaction occur over https. Don’t fall into the trap of trying to document everything as capabilities, they are supposed to be abstract. Leave it to the test cases or exploratory testers themselves to provide that level of details.
3. A capability should be composed with other capabilities. In fact, a user story or use case should be describable with a series of capabilities. If a user story cannot be written with only the existing capabilities, then there are missing capabilities or the capabilities are at too high a level of abstraction.

**Guidelines for translating capabilities to test cases**

* Every capability should be linked to at least on test case. If capability is important enough to document, it is important enough to test.
* Many capabilities requires more than one test case. Whenever there is variation in the inputs, input sequences, system variables, data used and so on, multiple tests cases are required.
* Not all capabilities are equal, some are more important than others. How risk is associated with the capabilities to distinguish their importance.

**Example: Determining Attributes, Components and capabilities for Google+**

ACC can be performed quickly in a document, spreadsheet or even on a napkins.

Here is abbreviated example of ACC for Google+

* **Google+ Attributes (derived exclusively by watching an executive discuss Google+)**

Social**:** Empowers users to share information and what they are upto

Expressive: Users can express themselves through features

Easy: Intuitive. Easy to figure out how to do what you want to do.

Relevant: Shows only information the user care about.

Extensible: Capable of integrating with Google properties and third party

sites and applications

Private: user data won’t be shared

* **Google+ Components (derived from architecture documents)**

Profile: Information and preferences for the logged in user.

People: Profiles that the user has connected with.

Stream: A ranked stream of posts, comments, notifications, photos and so on.

Circles: Group to put contacts into friends , co-workers and so on.

Notification: Indicators for when you are mentioned in a post.

Interest or +1: Indication for user likes

Posts: Buzz posts from the users and their contacts.

Comments: Comments on posts, photo, video and so on.

Photos: photos uploaded by the users and their contacts.

* **Google+ Capabilities**

**Profile:**

Social: Share profiles and preferences with friends and contacts.

Expressive: Users can create an online version of themselves.

Expressive: Personalize your experience with Google+

Easy: Easy to enter and update information and have it propagate.

Extensible: Serves profile information to applications with appropriate access.

Private: Enables a user to keep private data private.

Private: Share data only with approved and appropriate parties

**People:**

Social: Users can connect with users’s friends, coworkers

Expressive: Profiles of other users are personalized and easily distinguishable

Easy: Provides tools to easily manage a user’s contacts.

Relevant: Users ca filter their contacts based on criteria for relevance

Extensible: Serves contact data to authorized services and applications.

Private: Keeps data about a user’s contacts private to approved parties

**Stream:**

Social: Groups contacts into circles based on social context.

Expressive: New circles can be created based on the context of the user

Easy: Facilitates adding, updating and removing contacts to and from circles.

Easy: Facilitates creating and modifying circles

Extensible: Serves data about circles for use to services and applications.

**Hangout:**

Social: Users can invite their circles to hang out

Social: Users can open hangouts to the public

Social: Others are notified of hangouts they access in their stream

Easy: Hangouts can be created and participated in within a few clicks

Easy: Video and audio inputs can be disabled with a single click

Easy: Additional users can be added to an existing hangout.

Expressive: Before joining a hangout, users can preview how they will appear to others.

Extensible: Users can chat through text while in a hangout

Extensible: Videos from youtube can be added to a hangout

Extensible: Users without a webcam can participate in hangouts through audio

Private: Only invited guests can access a hangout

Private: Only invited guests are notified of a hangout

**Posts:**

Expressive: Expresses the thoughts of the user through Buzz.

Private: Posts are restricted to the intended audience

**Comments:**

Expressive: Expresses the thoughts of the user through comments.

Extensible: Posts data on comments for use by other services and applications

Private: Posts are restricted to the intended audiences

**Photos:**

Social: Users can share their photos with their contacts and friends

Easy: Users can easily upload new photos

Easy: Users can easily import photos from other sources

Extensible: Integration with other photo services

Private: Photos are restricted so that they are only visible to the intended audiences

**Risk Analysis**

For software testing, we follow a common sense process for understanding risk :

* + What events are we concerned about?
  + How likely are these events?
  + How bad would they be to the enterprise?
  + How bad would they be for customers?
  + What mitigations does the product implement?
  + How likely is it that these mitigations would fail?
  + What would be the cost of dealing with the failure?
  + How difficult is the recover process?
  + Is the event likely to recur or be a one time problem?

When I join a project first few weeks are spent listening and not talking. It is crucial to get an understanding of what’s going on and to learn the architecture of the product and dynamics of the team.

Formula to work in new project/Company

1. Learn first
2. Build street credentials
3. Start looking for ways to innovate