# Shelby El-rassi, T3A1, Flextrack, MEL

## Q1 |Provide an overview and description of a standard source control process for a large project.

#### **What is source control?**

Source control (also known as version control) is the tracking and management of files/code and any changes made. The practice of source control is the tracking of any modifications and the ability to revert back to any previous versions in the event of mistakes or backtracking, a key feature is the ability to know which version is the current iteration. An excellent example of this is Git.

**Git**

Git is a free and open source version control system, it is designed for source code management, however it does have the ability to management other sets of files. Git is currently one of the most popular version control system for software development, as it is a distributed version control system, most Git operations are local, with every user/client having their owner server and full back up of the repository locally. It has the advantages of speed, feature branching, useful staging areas and the ability to run multiple workflows. Below is an overview of the git-flow branching model, which has become a popular version control model for software teams.

**Git-flow Workflow**

Git-flow workflow first published by Vincent Driessen at nvie. It describes a firm branching model based around a projects release. The hefty framework is ideal for management of larger projects.

**Overview of this workflow:**

* Getting started with this workflow starts with an actual toolset available for installation, after installation (which is available on multiple operating systems), utilising git-flow for a project involves executing *git flow init*. It essentially is an extension of the standard *git init* command, that creates branches.
* A central repository houses the ‘Master’ branch, which is for the core of the project reflecting a production-ready base, along with another branch named ‘Develop’, this branch runs parallel with the master and reflects a state of latest features for next release (also known as an integration branch). An important note is the ‘Develop’ branch will house the entire history of the project, whereas the ‘Master’ will only contain the latest version.
* When *git flow init* is utilised, this command will create the ‘Develop’ branch.
* When a new feature is started, a new branch is created for this feature. This can be done via git-flow extension by the command *git flow feature start feature\_branch.*
* Each developer pulls and pushes to the origin ‘Develop’ as the parent branch.
* Developers may also pull changes from other peer’s branches, which can be useful for working together in sub-teams.
* Each branch should have a clear descriptive name with a clear purpose.
* Each feature branch should also be pushed to the central repository branch ‘Develop’, this means the code can be shared for collaboration.
* When complete these feature branches get merged back into ‘Develop’, never interacting with the ‘Master’ branch directly.
* Once a feature branch is finalised, the command *git flow feature finish feature\_branch* can be executed.
* When the ‘Develop’ branch has accumulated the required features for release or a scheduled date is imminent, a ‘Release’ branch gets forked off ‘Develop’. This ensures the next release cycle beings, so new features cannot be added after this point, only fixes, documentation and other release centred tasks.
* Once this ‘Release’ branch is ready to go, it gets merged into ‘Master’ and marked with a release number.
* The usage of a ‘Release’ branch for initial preparation of the release ensures that the rest of the team is able to continue working on features for the succeeding release date.
* It is important that the ‘Release’ branch also gets merged back into ‘Develop’ as well as ‘Master’ for any updates possibly added which new features might need access too. Once this process is complete the current ‘Release’ branch can be deleted.
* Another type of branch operated in this flow are ‘Hotfix’ branches, which are in place to swiftly fix up production releases. They operate just like feature branches, however, are parented off the ‘Master’ branch. A ‘Hotfix’ branch is the only branch forked directly off the ‘Master’. Once the fix is fully complete, these branches get merged immediately into ‘Master’ and ‘Develop’, with the ‘Master’ tagged with an updated version number. Operating a dedicated branch just for branch is a key advantage to this workflow as it ensures issues can be addresses by the team, with total interruption of the rest of the workflow.

In summary, this workflow is great for the large projects operating on a release-based schedule, offering the specific branches enable this workflow management, minimising interruption.

#### [Sources]---

1. <https://www.wrike.com/project-management-guide/faq/why-should-i-use-version-control-in-project-management-software/>
2. <https://www.automationworld.com/home/blog/13316896/configuration-management-and-version-control>
3. <https://hackernoon.com/top-10-version-control-systems-4d314cf7adea>
4. <https://nvie.com/posts/a-successful-git-branching-model/>
5. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

## Q2 |What are the most important aspects of quality software?

Through the perspective of software engineering, software quality implications involve two related, but separate concepts, these being;

1. Software functional quality, being the base product quality model.  [ISO/IEC 25010](http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733) defines a set of eight software quality characteristics, measuring software through these aspects is to define what makes a product worthwhile amongst competitors and in relation to behaviour. These aspects are:
   1. Functional Suitability
   2. Performance Efficiency
   3. Compatibility
   4. Usability
   5. Reliability
   6. Security
   7. Maintainability
   8. Portability.
2. Software structural quality, which is the quality-in-use model. By extracting from ISO standards, CISQ (Consortium for Information & Software Quality) defined quality aspects at a source code level in order to support the functional level qualities, but also enabling measurement of quality in meeting these function requirements this determining if the software works as intended. These four important aspects are all derived from the original eight and involve them all, just at a more specified functional level. These aspects are:
   1. **Reliability:** This is referring to the presence of consistent behaviour of software and the overall stability of the program in the event of unforeseen conditions. Errors should either be correct handled with no effect on the user, or no error occurrence at all. The user should never question the execution of the program. To summarise, reliability is the characteristic of resiliency and structural solidity.
   2. **Performance Efficiency:** This is referring to how the software utilises resources and their execution time, amount of database calls and overall effect on program scalability, satisfaction and response times. The overall code architecture and design contributes to software efficiency. Additionally, is code compliance with best architecture and database practices. Overall, efficient code is fast, memory-efficient and consumes the fewest CPU cycles.
   3. **Security:** This is referring to protection in the case of security risks and how well this is executed. The likelihood of attackers breaching the software, interrupt processes and gain access to restricted information is a measure of the security quality and how well the code architecture was formed and put into practice. The measurement of this aspect is via ‘vulnerabilities’, which are known issues possibility resulting in a breach, the ability to detect them, the amount of these vulnerabilities found and the gravity of them indicate the overall security level. Other factors also include the resolution time taken to fix an issue, deployment of security updates and the actual amount of full system breaches as opposed to just vulnerability checks.
   4. **Maintainability:** This is referring to the ease in which software can be modified, utilised and adapted for alternative purposes and overall portability between environments and development teams. If compliance with software architecture practices and consistent high-quality code is produced across the board then easier streamlined maintenance is likely to be more achievable. Overall high-quality code is clean code that is consistent, easy to understand, well documented, is DRY, has good error handling and can be tested.

#### <https://www.it-cisq.org/standards/code-quality-standards/>

<https://www.it-cisq.org/coding-rules/index.htm>

<https://en.wikipedia.org/wiki/Software_quality>

#### <https://www.silasreinagel.com/blog/2016/11/15/the-seven-aspects-of-software-quality/>

<https://www.sealights.io/software-quality/measuring-software-quality-a-practical-guide/>

<https://www.altexsoft.com/blog/engineering/what-software-quality-really-is-and-the-metrics-you-can-use-to-measure-it/>

<https://asq.org/quality-resources/software-quality>

<https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality#:~:text=Code%20quality%20defines%20code%20that,%2C%20bad%20%E2%80%94%20is%20all%20subjective.&text=Code%20that%20is%20considered%20high,for%20a%20web%20application%20developer>.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q3 |Outline a standard high level structure for a MERN stack application and explain the components.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q4 |A team is about to engage in a project, developing a website for a small business. What knowledge and skills would they need in order to develop the project?

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q5 |With reference to one of your own projects, discuss what knowledge or skills were required to complete your project, and to overcome challenges.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q6 |With reference to one of your own projects, evaluate how effective your knowledge and skills were for this project, and suggest changes or improvements for future projects of a similar nature.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q7 |Explain control flow, using an example from the JavaScript programming language.

Control flow is the flow or order in which statements in a script are executed. Code is run line by line started from the first line then continuing until the whole code is implemented. Factors that alter the control flow are structures such as conditionals and loops. So overall control flow involves not just start to finish script reading, but the reading and analysing the overall structure of the program to determine order of execution dependant on how many instructions present that alternative the start to finish flow.

An example in JavaScript is when code execution reaches a control structure of a conditional *if/else* statement. A conditional statement is a set of instructions that executes if a specification is true. When the statement is reached the given expression is analysed and then chooses the direction the program takes based on if the logic is true or not. Else is an optional statement utilised to enable an alternate path. In the example below, is the condition is true, statement 1 is executed, if the condition is false, statement 2 will be executed.

<https://developer.mozilla.org/en-US/docs/Glossary/Control_flow>

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q8 |Explain type coercion, using examples from the JavaScript programming language.

Type Coercion is the automatic or implicit conversion process of converting a value from one type to another type. Such as a string to a number. Adjacent to type coercion is type conversion, which is explicit conversion of data types, which involves the explicit requiring of a conversion to take place rather than automatic as aforementioned.

An example of this is the toString() method, which can return a converting number to a string as shown below. In this snippet to code is specifically calling a method of conversion on the number 15 which then results in a string 15.

(15).toString() //=> “15”

Type Coercion has differences to Conversion. Coercion is the term used when unexpected type casting happens in JavaScript. Coercion of a data type is often a side effect of different operations. For example, like the code shown above of a number to a string, a type coercion version of the same example would be:

15 + “” //=> “15”

#### Adding a string to a number will always result in a string. A reverse of the above example, coercing a string to be a number is below. In this example using the “-“ operator means JavaScript will cast the values to be numbers.

“15” - 1 //=> 14

These snippets are in themselves simple examples of Type Coercion, by they do show the way JavaScript when it is executing the code will even out the datatypes and cast them automatically to enable to operator to work.

<https://www.w3schools.com/js/js_type_conversion.asp>

#### <https://hackernoon.com/understanding-js-coercion-ff5684475bfc>

https://developer.mozilla.org/en-US/docs/Glossary/Type\_coercion

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q9 |Explain data types, using examples from the JavaScript programming language.

#### Data types like in any programming language tell the characteristic of a piece of data; this is conveyed to the compiler so that is can perform the applicable operation. In JavaScript data types can include primitive, which means these data types are immutable or unchangeable, and non-primitive or objects which are collections of properties in data usually referenced by an identifier. Objects are not immutable as they can have properties added and removed. The value properties in an object can be of any type, also including other objects, so this enables the creation of complex data structures.

#### Primitives

* Boolean: This type is representative as a logical entity, either the value true or the value false. Other data type values such as NaN, null and undefined are by definition a falsey value along with false, 0 and an empty string. All other values truthy by nature.
* Null: This value is representative of empty, or the intentional absence of a value.
* Undefined: This value is utilised when a variable has not been assigned a value yet, by default it becomes undefined.
* Number: Number is representative of numeric values. JavaScript is different to other languages in that it cannot distinct automatically an integer and float. So technically the type only has one integer, this being 0 and all other numbers are represented as a double-precision 64-bit floating point format. Other languages will know to round to the ideal representation, this is not the case for JavaScript. That is why methods like .round() exist in JavaScript to enable the rounding of integers. The number type can also including +Infinitity, -infinity and NaN(not a number).
* BigInt: A BigInt is representative of an integer with arbitrary precision, it is also numeric in nature. This value is utilised to store and operator on large integers safely when they go beyond the double-precision floating-point number range.
* String: A string is representative of textual data and comes in the form of a string of characters. JavaScript strings are immutable in that once created; they are unmodifiable directly. However, utilising .substr() to pick individual letters or .concat() to concatenate two strings together are alternatives to ‘modifying’ a string.
* Symbol: A symbol is representative of a unique and immutable value, such as the key in a JavaScript object.

#### Objects (non-primitive)

* Object: A object in JavaScript is representing of key,value pairs which are a list of xero or more property names and corresponding values, which are enclosed in curly braces.
* Array: Arrays are also by definition objects, however they relationship differs in that the key is the integer set index paired with the length property of the array.

There are many other built-in JavaScript objects that come in the form of a standard library, an example of these are:

* Error
* Math
* Set

[*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data\_structures*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures)

[*https://en.wikipedia.org/wiki/Double-precision\_floating-point\_format*](https://en.wikipedia.org/wiki/Double-precision_floating-point_format)

[*https://www.programiz.com/javascript/data-types*](https://www.programiz.com/javascript/data-types)

## Q10 |Explain how arrays can be manipulated in JavaScript, using examples from the JavaScript programming language.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q11 |Explain how objects can be manipulated in JavaScript, using examples from the JavaScript programming language.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q12 |Explain how JSON can be manipulated in JavaScript, using examples from the JavaScript programming language.

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)

## Q13 |For the code snippet provided below, write comments for each line of code to explain its functionality. In your comments you must demonstrates your ability to recognise and identify functions, ranges and classes.

class Car {

constructor(brand) {

this.carname = brand;

}

present() {

return 'I have a ' + this.carname;

}

}

class Model extends Car {

constructor(brand, mod) {

super(brand);

this.model = mod;

}

show() {

return this.present() + ', it was made in ' + this.model;

}

}

let makes = ["Ford", "Holden", "Toyota"]

let models = Array.from(new Array(40), (x,i) => i + 1980)

function randomIntFromInterval(min,max) { // min and max included

return Math.floor(Math.random()\*(max-min+1)+min);

}

for (model of models) {

make = makes[randomIntFromInterval(0,makes.length-1)]

model = models[randomIntFromInterval(0,makes.length-1)]

mycar = new Model(make, model);

console.log(mycar.show())

}

#### [Sources]---

1. [label](https://github.com/Shelby219/T3A1_workbook/blob/master/link)