**Design document for MusoPlan Software**

**Summary.**

The proposed Musoplan software is expected to be a tool to be employed by the music management team to measure how successful Globex’s elite musicians will be, and the cost associated with their deployment as a troupe.

**Program inputs and outputs.**

The command line interface (CLI) will be used to input and create data into the software. These inputs will include creating the profile of the musicians, classifying the musicians into troupes with adequate description of each troupe, calculation of cost of deployment of any troupe to an event with outputs available by writing the inputs to a file.

**Software Class and method definition.**

The software is going to be built using a modular approach where classes and functions will be defined in different scripts and called from the main index page. Among the classes to be defined are.

|  |  |  |  |
| --- | --- | --- | --- |
| S/No | Class | Method | Data |
| 1 | Musician | Display musician details. descibeMusician(musician\_name) {  If condition for validation based on name} | Instrument type eg. Guitarist, pianist etc., name, career years, hourly rate etc. |
|  | Bassist (Child class to Musician class) | displayInterestingFact() | “Everyone loves a bassist” |
|  | Flautist (Child class to Musician class) | displayInterestingFact() | “1989 heavy metal instrument of the year” |
|  | Percussionist (Child class to Musician class) | displayInterestingFact() | “Me drum” |
|  | Guitarist (Child class to Musician class) | displayInterestingFact() | “The more strings you have the better you are” |
| 2 | Troupe | Display troupe name, genre etc. descibeTroupe(troupe\_name) {  If condition for validation based on name}  Aggregation in Troupe | Troupe name, minimum duration, genre.  Musician array in troupe. |
| 3 | Add\_Musician |  |  |

**Version Control System:** Git will be used to monitor changes to the code for repeatability and version control. Two branches will be created to manage the codes and documents differently. Each commit message will follow industry standard and will be readable and meaningful to prospective users. Code commit will be done as each function is developed. A README.md file will be made available to explain the scope of the project and what is covered in it.

**Code testing:** code testing will be done manually, and a report appended.

**Version control**

The process of monitoring and managing changes to software code is known as version control, also commonly referred to as source control. Software technologies called version control systems assist software development teams in tracking changes to source code over time. Version control systems enable software teams to operate more swiftly and intelligently as development environments have increased. They are especially helpful for DevOps teams because they enable them to speed up successful deployments and cut down on development time.

Every change to the code is recorded by version control software in a particular form of database. If a mistake is made, programmers can go back in time and review prior iterations of the code to help fix it while causing the least amount of interruption to the entire team.

Without using any kind of version control, software development teams frequently encounter issues like not knowing which changes have been made are accessible to users or the generation of incompatible modifications between two unconnected pieces of work that must then be carefully untangled and revised. Developers that have never used version control may have added versions to their files, sometimes with suffixes like "final" or "latest," and then dealt with a new final version afterwards. You may have code blocks that are commented out because you wish to remove some functionality but keep the code in case you need it in the future. Version control offers a solution to these issues.

Version control software is a crucial component of the day-to-day professional activities of the modern software team. The enormous value version control provides them even on tiny solo projects is often recognised by individual software developers who are used to working with a capable version control system in their teams. Many developers wouldn't even think about working without version control systems for non-software tasks once they become acclimated to their potent advantages.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/No** | **VCS** | **Feature** | **Pros** | **Cons** |
| 1 | Git | * Significant support for non-linear development is offered. * Model of a distributed repository. * Compatible with current protocols and systems including HTTP, FTP, and ssh. * Able to handle small- to large-scale projects with efficiency. * Historical verification using cryptography. * Programmable merging techniques. * Framework-based design. * Periodic packing of explicit objects. * Garbage builds up until it is collected. | * Incredibly quick and effective performance. * Cross-platform * Tracking code changes is simple. * Sturdy and easily maintainable. * Offers the excellent git bash command-line tool. * Provides a GIT GUI where you can easily re-scan, modify the status of the project, sign off, commit, and push the code with only a few clicks. | * Understanding a complex and larger history log is challenging. * Does not allow the preservation of timestamps and keyword expansion. |
| 2 | Concurrent Versions System (CVS) | * Model for a client-server repository. * The same project might be worked on concurrently by several developers. * Only when an edit conflict arises will CVS client require user intervention to maintain the working copy of the file current. * Keeps a record of the project's past. * Anonymous access to read. * To keep local copies up to date, use the 'Update' command. * Can support several project branches. * Symbolic links are excluded to prevent security risks. * Use delta compression for effective storage. | * Fantastic platform support. * A robust and feature-rich command-line client enables strong scripting. * The wide CVS community's helpful assistance enables efficient source code repository web browsing. * It's a well-established, well-understood tool. * Fits the open-source community's collaborative character perfectly. | * There are no source code repository integrity checks. * supports neither atomic check-outs nor commits. * Poor distributed source control support. * does not support merging tracking or signed revisions. |
| 3 | Subversion (SVN) | * Model for a client-server repository. SVK does, however, allow for distributed branches in SVN. * Directory versions exist. * Operations like moving, deleting, copying, and renaming are also versioned. * Allows for atomic commits. * Signed links with versions. * Versioned metadata in free form. * Binary diff storage that uses less space. * Branching is a low-cost process that is independent of file size. * Merge tracking, complete MIME support, path-based authorisation, file locking, and solo server operation are further features. | * Benefits from having quality GUI tools like TortoiseSVN. * Supports directories that are empty. * Provide stronger Windows support than Git. * Easy to administrate and set up. * Easily integrates with Windows, top IDEs, and Agile tools. | * Does not keep track of when files were modified. * Does not handle filename normalisation correctly. * Not compatible with signed revisions. |
| 4 | Mercurial | * high scalability and performance. * High level of branching and merging. * Collaborative development that is entirely distributed. * Decentralized Supports both binary and plain text files well. * Features a built-in web interface. | * Great performance and scalability. * Lots of merging and branching. * A distributed, collaborative development process. * Decentralized supports plain text and binary files as well. * Has an integrated web interface. | * Python must be used for all add-ons. * Checkouts in parts are not permitted. * When used with extra extensions, it becomes quite difficult |
| 5 | Monotone | * Provides excellent support for internationalisation and localisation. * Integrity is prioritised over performance. * Created with distributed processes in mind. * Tracks file revisions and authentications using cryptographic primitives. * Able to import CVS projects. * Utilises netsync, a very effective and reliable proprietary protocol. | * Reduced maintenance requirements * Adequate documentation * Simple to learn * Modular construction * Excellent results when branching and merging * Secure GUI | * Performance difficulties were noticed for a few operations, the initial draw being the most obvious. * Can’t check-out or commit while behind a proxy (this is because of a non-HTTP protocol). |