Task 1: Comparison of implementation techniques

a) Versions can be variants or revisions. A variant describes intentional variations that exist in parallel (2013 Apel et.al).

Don’t Starve (Game, please refer to Steam for further information): The basic single-player game has several expansions called ‘Shipwrecked’ (SW) and ‘Reign of Giants’(RoG). These both variants can only be activated independently, so I can only choose to include the contents of RoG or SW, not both, on top of my basic game. These two can be considered as two variants of the software. Each patch of the software can be seen as a revision (not only the releases the customer sees). Both the expansions and the patches are called versions.

b)

(I) VCS can be used for SPL by creating new release branches from other branches (usually the central development branch). The separate release branch is then used to deploy a modified product to the customer, maybe including additional requested features.

(II) We can use a build system with two approaches. First, using the parameter approach to use conditional builds during compile-time. Second, using custom build scripts, eg. when using gradle every customer has an own build.gradle file that builds only the needed feature during compile-time.

(III) Preprocessors are used prior to compilation. The source code is annotated with a specific syntax not interfering with the syntax of the programming-language, before compilation the not selected feature code is commented or deleted depending on the preprocessor.

c) VCS would be preferred for products with few variants because there is a risk of the different branches to diverge otherwise, additionally we can use VCS “on-the-fly” to create variants very quickly since only a low preplanning effort is necessary.

Build-tools are best suited for features that can be created cohesive since the granularity is on file-level, hence we can also use it to compile very large software systems (eg. linux).

Preprocessors allow a fine granularity, but annotating large parts of the code can decrease readability. Programs that are small or have only a small amount of features are better use cases. Also preprocessors are considered simple to apply.

	VCS	Build-systems	preprocessors
Performance	Least amount of boilerplate code in the best case, since only the code parts needed are in the repository. Hard to reuse code, since source code is scattered all over different branches.	Maybe we need to compile entire not needed interfaces/ abstract classes. The runtime performance should be fine.	Excluding unused code should bring a good runtime, for the compile-time I cannot know since I never tried that
Collaboration among different	Well understood tool, but the overhead when	Work can be split and features can be integrated	With disciplined annotations it should

collaborators	maintaining a large amount of variants might lead to confusion	separately if the composition is good. Large build scripts can become hard to analyze.	be fine. In other cases this method is prone to errors.
Acquisition of third party software	Painful, since this changes need to be spread over all different repositories	+ dependencies are easily added	I guess this is like the usual process, because the annotations fluently included in the natural source code.
Fine-grained extension of code-base	- Branching because a few line of code increases maintenance effort, this changes also lead to bad feature traceability.	- Only file level, copying entire files for one new line of code is the only solution.	+ Easy to do, allows extensions on statement level.