**Davison Bullock MSc Information Technology PT Year 2 2012/2013**

**Submission 24/12/2012 Deadline extended 27/12/2012 DESIRED WEIGHT Part A 90% Part B 10%**

**Part A - Build Tools – Ant and Maven Compared**

**1. Background**

During the software development process developers automate a wide variety of tasks including compiling source code into binary code, packaging binary code, running tests , deploying systems and producing documentation (Ref 7). The purpose of build tools is to provide something useful usually an executable from code (Ref 3).

Previously building software involved compiling and linking code by command line before GUI (Ref 7 and Ref 3). Using the command line was unsuitable where many modules existed (Ref 7) as projects were becoming more complex (Ref 3) due to large file structure, having many files and a large amount of precompiled libraries and dependencies (Ref 3 and Ref 8). The compiler is not able to see all files and libraries in one place (Ref 3).

MAKE scripting language was used to create a build script that would call a series of calls to compile and link steps required to build the software deployable object (Ref 7) and allowed source code dependency management. Build tools were being used for "orchestrating" compilation and executing tests for the actual deployment amongst other things (Ref 3). Build tools started to come with dedicated scripting language to tackle the complexity of builds by automating and making the builds process more efficient(Ref 3).

Recent years have seen build management solutions which makes the build automation process easier. Developers have added the ability for pre and post compile activities to build automation (Ref 7).

**1.1. Reasons for Build Tools**

Code dependencies resulted in having to compile all classes (super or sub classes) every time there was a change to one class was a problem as when changes was made. The ability to re-compile only the code or libraries that was changed is one of the reasons for creating build tools (Ref 3).

Other reasons for build tools was to allow portability of code. Having build tools that were not dependant on the IDE like Make scripts meant that when code was built it would not give different compilation results (Ref 3). The tools allowed the build process to be automated (Ref 3).

Build software is available as commercial or open source that creates automated build and workflow processing. Some solutions automate the pre and post steps calling the build scripts. Others extend this further by also focusing on streamlining the compile and linker calls without the need for much manual scripting. Build tools are useful in the continuous integration builds process where frequent calls to the compile process is required when incremental building (Ref 7).

Build automation has the advantage of improving product quality, minimizing bad builds, automating the compile and link processing, eliminating dependencies on key personnel and saving time and money (Ref 7).

**2. Apache Ant and Apache Maven build tools**

Ant and Maven are both build tools in Java(Ref 8 and Ref 1) based on different concepts and work in a different way to each other (Ref 3). An early release of Maven was based on Ant but a later release is standalone of Ant. Both are open source software for building a project and both are available as plug-ins in popular IDEs such as Eclipse. They both abstract away dependencies in the builds process and both are configured using XML (Ref 3).

**Table 1 - Ant and Maven High Level Features (Ref 5 and Ref 6)**

|  |  |  |
| --- | --- | --- |
|  | **Apache Ant** | **Apache Maven** |
| **Purpose** | Tool for automating software build process. | Project management and build automation tool for Java. Similar to Ant but Maven also used to build and manage projects written in C#, Ruby and Scala. |
| **How** | - Uses java language platform.  - Suited to building java projects.  - Used by most java development teams.  - Uses XML (build.xml) to describe its build process instead of MAKE file.  - Easier for developers to adopt TDD and extreme programming as it allows JUnit tests integration in builds process.  - Build file (XML) lists targets and actions that must be taken before each target is built.  - Actions maybe running in-built tasks e.g. mkdir, javac etc. | - Uses java language platform  - Suited to building Java projects and C#, Ruby and Scala projects.  - Uses XML Project Object Model (pom.xml) to configure its build process.  - POM file contains any dependencies, components, external modules, build directories, build order and required plug-ins.  - Tasks are predefined as targets in the POM file e.g. compiling and packaging code.  - Plug-ins do all the build work  - Other tasks include cleaning target directories, running unit tests and generating documentation.  - Based on plug-in architecture where plug-ins can be added to interface with build tools (compilers, unit test tools etc). |
| **Why** | - Ant built because previous way of building software was to use the Make shell commands which were platform specific and different commands were required to run for the platform used. This caused a portability problem. | - Maven helps users make the build process easier and to have uniform build environments between projects |
| **IDE Integration** | - Ant can be run in popular IDE's such as Eclipse and Netbeans simplifying its usage.  - Ant may be extended using widely available third party extensions | - Extensions integrate Maven with build and source editing mechanisms in the popular IDEs such as Eclipse, Netbeans, IntelliJ and JDeveloper.  - The actual POM file can be edited and used within the IDE to see where the dependencies are. |
| **Limitations** | - XML file may be hard to understand from human perspective  - Core tasks <javac>, <exec> and <java> are built on commands where default parameters have not changed between versions. Older Ant scripts would break if newer versions of tasks are used.  No persistence of state and limited fault handling. Can only be used for build and test process unlike Maven which can be used also as a workflow tool.  Ant uses verbose language, build files for larger and complex projects can be large. | - Not as easy to learn for new developers |

**2.1 Ant and Maven Build Concepts and Approach**

Ant is an older tool than Maven (Ref 8 and Ref 1) and is different to the powerful Make language. It uses a model that is extended by java classes rather than using shell commands like the Make language (Ref 8 and Ref 1). Configuration is done using files based on XML instead of running shell commands. Ant is user configurable to suit a project and is easy to use as new team members can study the build and then use as a basis for other builds. Each project can have a different build process . Ant is user configurable making it a powerful tool for the build process. (Ref 8).

Maven looks for a standard approach to build projects, what a project consists of, an easy way of publishing project information and distributing java JAR files across several projects (Ref 8). Maven encourages using best practices and development teams may prefer to use Ant instead. Maven is less flexible than Ant and using it may require users agreeing to make compromises (Ref 8). Maven is considered a more comprehensive tool than Ant (Ref 1).

**3. Ant and Maven Differences**

Maven is different to Ant and is not just a build tool like Ant. Maven allows software build as well as a structure and organization layer. Maven does all of what Ant can do because Ant can be set as the target in Maven (Ref 8).

**3.1 Why choose Maven over Ant or Ant over Maven ?**

Both tools are good build tools with Maven being the preferred choice over the older Ant tool (Ref 1).

The main reasons users would choose to migrate to Maven are projects are standardized, has clearly defined dependencies, works with the concept of artefacts which can be shared to other projects, reports and documents and remote and local repository concepts. Maven helps users make the build process easier and to have uniform build environments between projects (Ref 8).

The choice of tool is governed by the projects needs and where developers may want to concentrate on real work, carry out developments faster or be able to generate documentation easily. The choice of tool may also keep development managers happy if they can have a positive impact on reducing development costs, reduced time to market, and if reports can be produced frequently and presented to the senior managers (Ref 8).

**Table 1 - Main Advantage and Disadvantages of Ant and Maven (Ref 8)**

|  |  |
| --- | --- |
| **Maven** | **Ant** |
| Description of project | Development of a build script per project |
| Invocation of defined goals (targets) | Invocation of project specific targets |
| Project Knowledge | "just" the build process |
| Build lifecycle, standard project layout | Too complex scripts |
| Reusable plug-in, repositories | Scripts are not reusable |
| Moving fast forward | Development are slowed down |

Maven has the advantage of being very flexible because plug-ins including core ones can be retrieved from online repositories. Projects and sub projects can be related to each other using a single source of information.

Other advantages of using Maven include the automatic downloading of dependencies, standard conventions for naming and layout, code coverage, a wide variety of reports can be produced and JUnit tests can be integrated within the build process (Ref 8).

**3.2 Maven Compared With Ant**

**3.2.1. Design** - Maven projects have a certain structure and a set of supported task workflows compared to Ant (defined in the POM file). The design resembles how IDE’s handle projects providing benefits such as automatic integration with a Maven project with other development tools (Ref 6).

In Ant, projects do not exist (Ref 6) formal conventions to support this does not exist (Ref 1) . Informal conventions however have evolved and do exist (Ref 1). Ant instead chains together tasks and executes them based on Boolean conditions and inter-dependencies (Ref 6). Users have to look for sources and note where they output files (Ref 1). In Maven on the other hand users do not need to look for source code as Maven consists of conventions (Ref 1).

Ant is procedural and build tasks need to be specifically ordered compared to Maven which is declarative and where all files are taken care of if they are in directories and stored in the pom.xml file (Ref 1).

Ant does not use a life cycles but Maven does. In Ant a series of build tasks are assigned manually to each goal compared to Maven where a series of build tasks are executed until a lifecycle is reached. When a lifecycle is reached default plug-ins are run resulting in compilation and building of a JAR file (Ref 1).

**3.2.2. Ease of use** – A developer not familiar with Ant would be able to determine what an Ant script does and any dependencies by looking at the script. This is not the case with Maven (Ref 6).

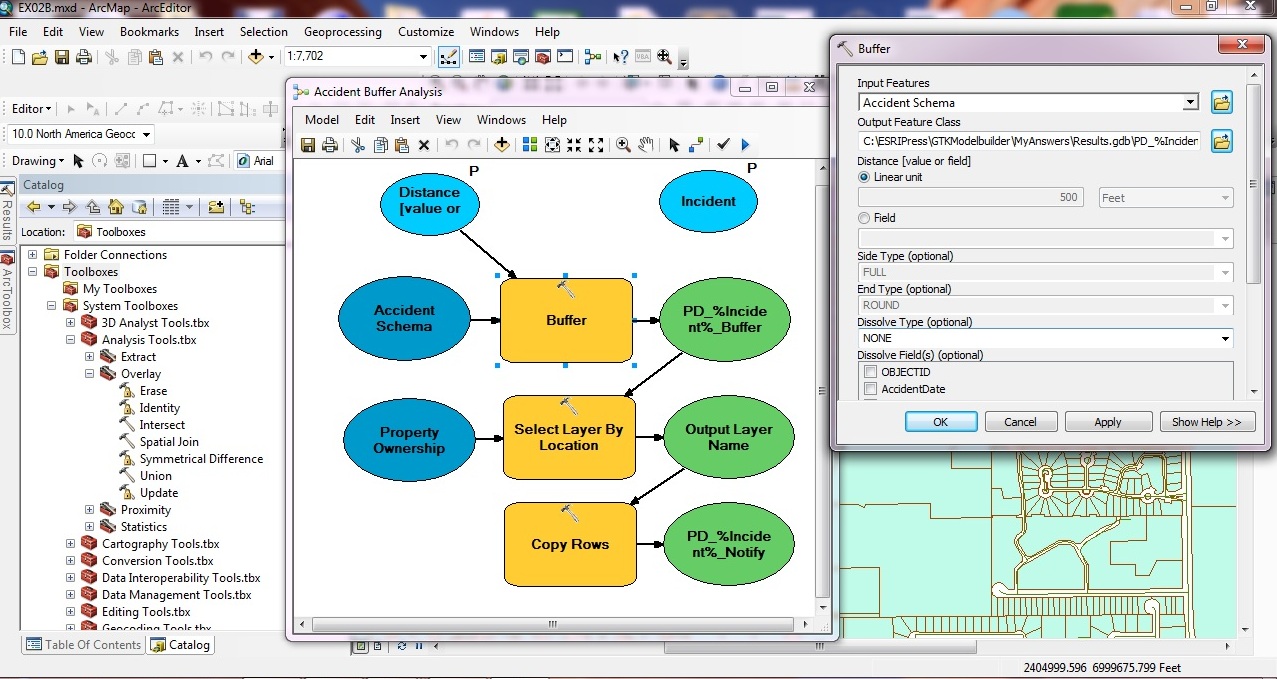
However an experienced developer would not be able to infer what the high level structure is for an Ant project without examining the script in detail. This is unlike a developer looking at a Maven project for the first time. The developer can quickly familiarise themselves with the structure of project and execute workflows that are already familiar with knowing what the expected outcome is (Ref 1).

Ant scripts are not reusable whereas Maven has reusable plug-ins which makes it more flexible and is a reason for why people may prefer to use it instead of Ant (Ref 1).

**Part B - A tool that has options that should be handled with care.**

**1. ArcGIS GIS v9.3 (Geographic Information System) software.**

ArcGIS desktop is a popular desktop suite of GIS applications that includes allowing users to create maps. ArcMap is a popular mapping/GIS tool in the ArcGIS suite of tools. ArcMap allows users to manipulate spatial data and carry out spatial analyses amongst other tasks. These tasks can be run from a model that can be created in the model builder interface where ArcGIS geo-processing tools (tools that do GIS processing work) can be dragged onto the model builder canvas and interconnected. When a model is run all the processes will be executed and data is saved to a file (dbf, txt, csv) or geo-database table (Ref 4).



**Fig 1 - Above: Geo-processing model in ArcMap showing user having to set outputs individually for each of these tools (in orange).**

By default users cannot overwrite outputs every time the same model is run without them

having to specify new outputs for the feature-class (data layers/tables) in the geo-database every time the model is run. This gives users an element of control when running a model . Users have an option where they can check and allow model output data to be overwritten every time the model is run (Ref 2).

**1.1 Why does option exist and when should it be used?**

The option to overwrite when used carefully allows users to re-run models quickly where they have to be run often without having the cost in having to interactively specify new output layers for each geo-processing task that is in the model (Ref 2 and Ref 4).

**1.2 Why is the option is dangerous and why would someone use it unintentionally?**

The option to overwrite is located in a nested dialog found on the applications options menu. This allows users to overwrite the output data every time a model is run without warning that the file will be overwritten (Ref 2).

New users may not be aware the overwrite option being selected and existing users are unlikely to check or may forget that they have it selected because the option is hidden especially when map documents might be shared by different users who are unaware that the option is ticked.

There is the likelihood of data being overwritten and lost when the map document is opened and the model is run before users realise that their data has been overwritten. There is a risk of rework.

**References**

1. differencebetween.net. (2012). *Difference Between Ant and Maven.*Available: http://www.differencebetween.net/technology/software-technology/difference-between-ant-and-maven. Last accessed 22nd December 2012.

2. ESRI. (2009). *Controlling tool results.* Available: http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Controlling%20tool%20results. Last accessed 22nd December 2012.

3. Lachish,O. (2012). *Session 8 - Build Tools.* Available: http://www.dcs.bbk.ac.uk/~oded/Tools2012-2013/Sessions/Session-8/Session-8.pdf. Last accessed 16th December 2012.

4. Ormsby T, Napolean E, Burke R, Groessl C, Bowden L. (2009). Geoprocessing. In: Environmental Systems Research Institute Inc.,U.S.; 2nd edition (31 Jan 2009) *Getting to Know ArcGIS Desktop: Basics of ArcView, ArcEditor, and ArcInfo, Second Edition Updated for ArcGIS 9.3*. USA: Environmental Systems Research Institute Inc..

5. Various. (2012). *Apache Ant.* Available: http://en.wikipedia.org/wiki/Apache\_Ant. Last accessed 16th December 2012.

6. Various. (2012). *Apache Maven.* Available: http://en.wikipedia.org/wiki/Apache\_Maven. Last accessed 16th December 2012.

7. Various. (2012). *Build Automation.* Available: http://en.wikipedia.org/wiki/Build\_automation. Last accessed 16th December 2012.

8. Various. (2012). *Maven vs Ant and Ant vs Maven?.* Available: http://www.javafaq.nu/java-article1168.html. Last accessed 16th December 2012.