Technical Sciences

UDC 004.4'22

**Andrusiv Andrii**
*Student of the Faculty of Informatics*
*and Computer Science of the*
*National Technical University of Ukraine*
*"Igor Sikorsky Kyiv Polytechnic Institute"*

# ROLE OF BUILD AUTOMATION TOOLS IN SOFTWARE DEVELOPMENT

***Summary.*** *The process of turning source code into executable application is a hard and timewasting experience. In big projects which use databases and a lot of external libraries it is nearly impossible to keep track of all dependencies there are and tests that code must pass trough. That's why build automation tools exist. The article overviews and analyzes popular methods and tools used for build automation.*

***Key words:*** *build automation, continuous integration, software testing, maven, gradle, gulp, bazel.*

**Introduction.** Build tools are utilities that automate all task needed to create executable application from code. Build automation cycle includes different tasks depending on the tool, but basic tasks are [3]:

- Downloading dependencies.
- Compiling source code.
- Packaging binary code into executable file.
- Running automated tests.

Optionally build cycle can also deploy ready-to-run project on a platform and create project documentation [3].

**Why do build tools exist?**

Really small projects that doesn't use any external libraries can be easily build without using any building tools, but as soon as some external libraries or media/text files are included, or tests must be executed, developer will start having troubles following all dependencies, including files in the right spot, and making sure that ready application will pass all the tests needed. Using a tool in big projects allows the build process to be executed in right sequence and with right dependencies, and in general to be more consistent, saving time, money, and developers from going insane.

**Types of building tools [3]:**

- **Build utility** – tool that generates build artifacts through tasks. The main function of this utility is automation of simple, repeatable tasks. Developer can go through whole building cycle as well as execute individual tasks if needed ( for example you can clean previous build or run changed tests on already existing build )
- **Build servers –** usually web based tools, use build utilities on a scheduled or individually triggered basis.

  They usually called **continuous integration** servers. **Continuous integration** is the idea of merging all working copies to a shared main branch, and then executing building on daily basis to discover problems as quickly as possible. The main idea of CI is to reduce number of mismatches caused by a lot of people committing to the same project. It does that by making a new build and running tests each time one of the developers commits. The longer main branch of code remains checked out, the greater risk of conflicts. So CI servers try to rebuild projects as frequently as possible.

Here some examples of building utilities:

1. **Apache Maven** – one of the most popular building utilitie, and at the same time one of the oldest. Maven took Apache Ant as a base and improved on it. It operates by following instructions written in a pom-file using XML language. POM file consists of build ( version, language, plugins etc. ) and dependency ( external libraries ) instructions. Maven has a few different build cycles ( default, site, clean ), but each task in a cycle can be run individually. The dependencies specified in pom are downloaded from repository, considering version and already considering other dependencies an external library can have.

Here is an example of Maven pom.xml file used in Java project [2]:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
    <groupId>Internauka</groupId>
    <artifactId>HelloWorld</artifactId>
    <version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
    <description> HelloWorld </description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

The downsides of Maven is that it is hard to learn at first and it is not as flexible as some other building tools due to pom file restrictions. And another downside is that if all dependencies are downloaded from a repository, some of newer open-source libraries could not yet be there, and in that case you have to ether wait or to add them to your project build manualy.

**2. Gradle** – is a building utility that was based of Apache Ant and Apache Maven but using it's own domain specific language unlike XML style instructions used by both Maven and Ant.

Gradle is a bit more flexible allowing developer to change build cycle of application almost completely. So Gradle can be used in large services and multi-projects builds. But while being able to work on more difficult projects, Gradle builds faster due to incremental builds, were it determines which parts of a project are updated, so parts of a build cycle that depend only on them don't have to be re-run.

Due to heavy influence of Maven, Gradle basic plugins are focused on Java. But a lot of other languages and projects are supported too.

Gradle is using build.gradle file as instruction which is based on Groovy instead of XML, so this leads to smaller files with less "garbage" which are easy to read and change.

Here is an example of build.gradle file [2]:

```
apply plugin: 'java'

repositories {
   mavenCentral()
}
jar {
   baseName = 'Internauka'
   version = '1.0'
}

dependencies {
   compile 'junit:junit:4.0'
}
```

**3. Bazel** – is a free building automation utility by Google. Bazel is a open-sourced part of a main build tool used by Google called Blaze. Bazel is quite new, its first release date is in Marh 2015 and currently is in beta state.

Bazel is similar to Gradle, it is based on Ant and Maven, and it builds applications using a set of rules which are not created using XML. Bazel uses Skylark language to create instuctions ( a subset of Python ) . Bazel downloads dependencies from a large pull of repositories[4]. And in the same way Gradle does it, Bazel re-builds only parts of the project that was changed and which will be affected by those changes. So overall Bazel is just a more refined Gradle for Google needs. It is easily extendable due to Skylark language, and can use projects build by differend utilities as a repository, so Bazel can use almost every bit of code that is out there.

**4. Front-end tools, Gulp and Grunt.**

Another, a bit different category of build utilities appeared recently, which is oriented on front-end development. They are commonly called JavaScript Task Runners.

For this overview I picked 2 popular front-end building tools which are Grunt and Gulp. Both of them are used as a command line tool for JavaScript objects. The same way as regular build tools are, they performs repetitive tasks, like compiling, testing etc. But they chase different goals then classic building tools.

The dependencies and file management is not the main problem of front end, so why do we need a building tool for JavaScript? Here is why:

a) Compile

Nowadays there is a lot of tools that need to be compiled.

b) HTTP request overhead.

Each file is loaded with a minimum of 20-100ms per request, and the file size does not metter.

c) The bigger the file is – the bigger the download time.

d) Downloading the same file twice.

All of this is lowering the performance and making the client wait.

That's where building tools come in, in front-end they are oriented to improve performance of a web page. So what exactly this tools do?

a) Compiling new syntax to old syntax.

b) Concatenating files. Front-end projects usually have a lot of files, even a basic one-page can have up to 5 JavaScript files plus CSS files. Each file takes time to load, so to reduce the waiting time building tools can join all JS files into one, so only one huge JS file will be loaded.

c) Uglify ( compress/minify ) JavaScript.

Developers need to have a particular way of structuring code and way of naming functions and numbers that are "must have", because it is a lot easier to read and understand code in this way. But for a machine, it does not matter if you name variable "A" or "SecondPageCartIncrementationFlag", and we can cut out all the spaces unnecessary for a machine, etc. So we can minimize that and decrease the file sizes.

d) Revision – optimiza for caching.

After loading a file this tools can generate a hash and add it to the files that are already loaded, so when next page starts loading it will see the hashes and load only new files.

The main differences between grunt and gulp is that gulp is the more recent one, meaning that it was developed considering all the up and downs of grunt. Tasks in Gulp work using streams instead of files, like grunt does, so it does not need to wait until one file finishes its work to work with another, so Gulp contacts with a file system only in the beginning and at the end of its work.

Another difference is in configuration files, while Grunt files look more like JSON then JS, Gulp's files look like a simple clean JS code, which is more understandable in big projects.

We overviewed a few major building tools that used in software development. And analyzed functionality they offer. And to summaries it all.

Co-founder of Gulp:

"Builds can be the most awful sinkhole for teams to waste their time with".

And that's exactly why using build automation tools is one of the most important things in software development. Why dig a hole yourself, when there is machines made to do it? So next time you start a project, start it with the right tool.

## References

1. Continuous Delivery in Java. Essential Tools and Best Practices for Deploying Code to Production. / By Abraham Marín-Pérez, Daniel Bryant – Retrieved from https://www.safaribooksonline.com/library/view/continuous-delivery-in/9781491986011/ch04.html
2. Ant vs Maven vs Gradle - Retrieved from http://www.baeldung.com/ant-maven-gradle
3. Build automation – Retrieved from https://en.wikipedia.org/wiki/Build_automation
4. Bazel documentation – Retrieved from https://docs.bazel.build/versions/master/bazel-overview.html