Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Основы программной инженерии

Лабораторная работа №3

Вариант № 696969

Выполнил: студент группы P3208,
Васильев Н. А.

Преподаватель: Воронина Д. С.

Санкт-Петербург 2025

Текст задания

Написать сценарий для утилиты Apache Ant, реализующий компиляцию, тестирование и упаковку в jar-архив кода проекта из лабораторной работы №3 по дисциплине "Веб-программирование".

Каждый этап должен быть выделен в отдельный блок сценария; все переменные и константы, используемые в сценарии, должны быть вынесены в отдельный файл параметров; MANIFEST.MF должен содержать информацию о версии и о запускаемом классе.

Сценарий должен реализовывать следующие цели (targets):

1. **compile** -- компиляция исходных кодов проекта.
2. **build** -- компиляция исходных кодов проекта и их упаковка в исполняемый jar-архив. Компиляцию исходных кодов реализовать посредством вызова цели **compile**.
3. **clean** -- удаление скомпилированных классов проекта и всех временных файлов (если они есть).
4. **test** -- запуск junit-тестов проекта. Перед запуском тестов необходимо осуществить сборку проекта (цель **build**).
5. **xml** - валидация всех xml-файлов в проекте.
6. **doc** - добавление в MANIFEST.MF MD5 и SHA-1 файлов проекта, а также генерация и добавление в архив javadoc по всем классам проекта.
7. **music** - воспроизведение музыки по завершению сборки (цель **build**).
8. **native2ascii** - преобразование native2ascii для копий файлов локализации (для тестирования сценария все строковые параметры необходимо вынести из классов в файлы локализации).
9. **scp** - перемещение собранного проекта по scp на выбранный сервер по завершению сборки. Предварительно необходимо выполнить сборку проекта (цель **build**)
10. **team** - осуществляет получение из git-репозитория 4 предыдущих ревизий, их сборку (по аналогии с основной) и упаковку получившихся jar-файлов в zip-архив. Сборку реализовать посредством вызова цели **build**.
11. **env** - осуществляет сборку и запуск программы в альтернативных окружениях; окружение задается версией java и набором аргументов виртуальной машины в файле параметров.
12. **alt** - создаёт альтернативную версию программы с измененными именами переменных и классов (используя задание replace/replaceregexp в файлах параметров) и упаковывает её в jar-архив. Для создания jar-архива использует цель **build**.
13. **history** - если проект не удаётся скомпилировать (цель **compile**), загружается предыдущая версия из репозитория git. Операция повторяется до тех пор, пока проект не удастся собрать, либо не будет получена самая первая ревизия из репозитория. Если такая ревизия найдена, то формируется файл, содержащий

результат операции diff для всех файлов, изменённых в ревизии, следующей непосредственно за последней работающей.

14. **diff** - осуществляет проверку состояния рабочей копии, и, если изменения касаются классов, указанных в файле параметров выполняет commit в репозиторий svn.
15. **report** - в случае успешного прохождения тестов сохраняет отчет junit в формате xml, добавляет его в репозиторий git и выполняет commit.

Сценарий Apache Ant

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="web-lab3" default="build" basedir=".">

    <taskdef name="scp"
classname="org.apache.tools.ant.taskdefs.optional.ssh.Scp">
        <classpath>
            <pathelement location="lib/jsch-0.1.55.jar"/>
        </classpath>
    </taskdef>

    <property file="build.properties"/>

    <target name="compile">
        <echo message="Compiling Java sources..."/>

        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"
includeantruntime="false">
            <classpath>
                <fileset dir="${lib.dir}" includes="**/*.jar"/>
            </classpath>
        </javac>

        <echo message="Java sources compiled successfully."/>
    </target>

    <target name="build" depends="compile">
        <echo message="Building WAR file..."/>

        <mkdir dir="${build.dir}/WEB-INF/lib"/>
        <mkdir dir="${build.dir}/WEB-INF/classes"/>
        <mkdir dir="${dist.dir}"/>

        <copy todir="${build.dir}/WEB-INF/classes">
            <fileset dir="${classes.dir}"/>
        </copy>
        <copy todir="${build.dir}/WEB-INF/lib">
            <fileset dir="${lib.dir}" includes="**/*.jar"/>
        </copy>
        <copy file="${web.xml.file}" tofile="${build.dir}/WEB-INF/web.xml"/>
        <copy todir="${build.dir}">
            <fileset dir="${web.dir}" includes="**/*"/>
        </copy>
        <war destfile="${war.file}" webxml="${build.web.xml}"
basedir="${build.dir}"/>

        <echo message="WAR file created at: ${war.file}"/>
    </target>

    <target name="clean">
```

```
        <echo message="Cleaning up..."/>
        <delete dir="${build.dir}"/>
        <echo message="Clean up completed."/>
    </target>

    <target name="test" depends="build">
        <echo message="Compiling test sources..."/>
        <mkdir dir="${test.classes.dir}"/>

        <javac srcdir="${test.dir}" destdir="${test.classes.dir}"
includeantruntime="false">
            <classpath>
                <pathelement path="${classes.dir}"/>
                <fileset dir="${lib.dir}" includes="**/*.jar"/>
            </classpath>
            <compilerarg value="-processorpath"/>
            <compilerarg value="${lib.dir}/lombok-1.18.26.jar"/>
        </javac>


        <echo message="Running JUnit tests..."/>
        <mkdir dir="${junit.report.dir}"/>

        <junit printsummary="yes" haltonfailure="yes" fork="true">
            <classpath>
                <pathelement path="${test.classes.dir}"/>
                <pathelement path="${classes.dir}"/>
                <pathelement path="${junit}"/>
                <pathelement path="${hamcrest}"/>
            </classpath>
            <formatter type="plain"/>
            <batchtest todir="${junit.report.dir}">
                <fileset dir="${test.dir}">
                    <include name="**/TestResult.java"/>
                </fileset>
                <formatter type="xml"/>
            </batchtest>
        </junit>

        <echo message="JUnit tests completed."/>
    </target>

    <target name="xml">
        <echo message="Validating XML files..."/>
        <xmlvalidate failonerror="true" lenient="false">
            <fileset dir="${src.dir}">
                <include name="**/*.xml"/>
                <exclude name="**/resources/**"/>
                <exclude name="**/webapp/**"/>
            </fileset>
        </xmlvalidate>
        <echo message="XML validation completed successfully."/>
    </target>

    <target name="doc" depends="build">
        <echo message="Generating Javadoc..."/>
        <mkdir dir="${ant.dir}/docs/javadoc"/>

        <javadoc destdir="${ant.dir}/docs/javadoc" sourcepath="${main.dir}"
classpath="${lib.dir}" use="true" windowtitle="Project Javadoc">
            <classpath>
                <fileset dir="${lib.dir}" includes="**/*.jar"/>
            </classpath>
        </javadoc>
```

```
        <echo message="Generating MD5 and SHA-1 digests..."/>
        <checksum file="${dist.dir}/lab3.war" algorithm="MD5"
property="md5sum"/>
        <checksum file="${dist.dir}/lab3.war" algorithm="SHA-1"
property="sha1sum"/>

        <echo message="Creating custom MANIFEST.MF..."/>
        <mkdir dir="${ant.dir}/meta-inf"/>
        <echo file="${ant.dir}/meta-inf/MANIFEST.MF">
            Manifest-Version: 1.0
            Created-By: Ant Task
            MD5-Digest: ${md5sum}
            SHA1-Digest: ${sha1sum}
        </echo>

        <echo message="Updating WAR with javadoc and manifest..."/>
        <zip destfile="${build.dir}/../dist/lab3-doc.war" update="true">
            <zipfileset dir="${ant.dir}/docs/javadoc" prefix="javadoc"/>
            <zipfileset dir="${ant.dir}/meta-inf" includes="MANIFEST.MF"
fullpath="META-INF/MANIFEST.MF"/>
        </zip>

        <echo message="doc task completed. Output: ant/dist/lab3-doc.war"/>
    </target>

    <target name="music" depends="build">
        <echo message="Playing music after build..." />

        <exec executable="cmd.exe" osfamily="windows">
            <arg value="/c"/>
            <arg value="start wmplayer
&amp;quot;${basedir}/music/success.mp3&amp;quot;"/>
        </exec>

        <echo message="Music played successfully." />
    </target>

    <target name="native2ascii">
        <echo message="Converting localization files to ASCII format..."/>

        <mkdir dir="${build.dir}/locales-ascii"/>

        <native2ascii encoding="UTF-8" src="${native2ascii.resources}"
                      dest="${build.dir}/locales-ascii">
            <include name="**/*.properties"/>
        </native2ascii>

        <echo message="Localization files converted to ASCII and saved in
build/locales-ascii."/>
    </target>

    <target name="scp" depends="build">
        <echo message="Deploying project via SCP..."/>

        <scp file="${dist.dir}/lab3.war"
             todir="s366389@helios.cs.ifmo.ru:/home/studs/s366389/opi/lab3"
             port="2222"
             trust="true"
             password="Your Password"
             verbose="true"/>

        <echo message="Deployment done! File uploaded to server."/>
    </target>
```

```
    <target name="team">
        <echo message="Starting team build process..."/>
        <mkdir dir="${ant.dir}/team-builds"/>
        <mkdir dir="${ant.dir}/team-output"/>

        <antcall target="checkout-and-build">
            <param name="rev" value="HEAD~1"/>
            <param name="suffix" value="rev1"/>
        </antcall>

        <antcall target="checkout-and-build">
            <param name="rev" value="HEAD~2"/>
            <param name="suffix" value="rev2"/>
        </antcall>

        <antcall target="checkout-and-build">
            <param name="rev" value="HEAD~3"/>
            <param name="suffix" value="rev3"/>
        </antcall>

        <antcall target="checkout-and-build">
            <param name="rev" value="HEAD~4"/>
            <param name="suffix" value="rev4"/>
        </antcall>

        <zip destfile="${ant.dir}/team-builds/team-archives.zip">
            <fileset dir="${ant.dir}/team-output" includes="*.war"/>
        </zip>

        <echo message="Team build complete: ant/team-builds/team-
archives.zip"/>
    </target>

    <target name="checkout-and-build">
        <echo message="Checking out ${rev}..."/>
        <exec executable="git">
            <arg line="checkout ${rev}"/>
        </exec>

        <antcall target="build"/>

        <copy file="${dist.dir}/lab3.war"
              tofile="${ant.dir}/team-output/lab3_${suffix}.war"/>

        <echo message="Returning to latest revision..."/>
        <exec executable="git">
            <arg line="checkout master"/>
        </exec>
    </target>

    <target name="env">
        <echo message="Building under Java 17..."/>
        <antcall target="env-run">
            <param name="java.home" value="${java.home.17}"/>
            <param name="jvm.args" value="${jvm.args.17}"/>
            <param name="env.name" value="java17"/>
        </antcall>

        <echo message="Building under Java 21..."/>
        <antcall target="env-run">
            <param name="java.home" value="${java.home.21}"/>
            <param name="jvm.args" value="${jvm.args.21}"/>
            <param name="env.name" value="java21"/>
```

```
        </antcall>
    </target>

    <target name="env-run">
        <property name="java.executable" value="${java.home}/bin/java"/>
        <property name="javac.executable" value="${java.home}/bin/javac"/>

        <echo message="Using Java from: ${java.home}"/>
        <echo message="JVM Args: ${jvm.args}"/>

        <javac srcdir="${src.dir}" destdir="${classes.dir}" fork="true"
               executable="${javac.executable}" includeantruntime="false"
encoding="UTF-8">
            <classpath>
                <fileset dir="${lib.dir}" includes="**/*.jar"/>
            </classpath>
        </javac>
    </target>

    <target name="alt">
        <echo message="Alternative version of the project creating..."/>

        <delete dir="${alt.src.dir}"/>
        <mkdir dir="${alt.src.dir}"/>
        <copy todir="${alt.src.dir}">
            <fileset dir="${src.dir}"/>
        </copy>

        <replaceregexp match="\bResult\b" replace="AltResult" byline="true">
            <fileset dir="${alt.src.dir}" includes="**/*.java"/>
        </replaceregexp>

        <move file="${alt.src.dir}/java/web3/Result.java"
tofile="${alt.src.dir}/java/web3/AltResult.java"/>

        <replaceregexp match="\bResultsRepository\b"
replace="AltResultsRepository" byline="true">
            <fileset dir="${alt.src.dir}" includes="**/*.java"/>
        </replaceregexp>

        <move file="${alt.src.dir}/java/web3/database/ResultsRepository.java"
tofile="${alt.src.dir}/java/web3/database/AltResultsRepository.java"/>

        <replaceregexp match="new Result\(" replace="new AltResult("
byline="true">
            <fileset dir="${alt.src.dir}" includes="**/*.java"/>
        </replaceregexp>

        <replaceregexp match="Result\.class" replace="AltResult.class"
byline="true">
            <fileset dir="${alt.src.dir}" includes="**/*.java"/>
        </replaceregexp>

        <mkdir dir="${alt.classes.dir}"/>
        <javac srcdir="${alt.src.dir}" destdir="${alt.classes.dir}"
includeantruntime="false">
            <classpath>
                <fileset dir="${lib.dir}" includes="**/*.jar"/>
            </classpath>
        </javac>

        <mkdir dir="${alt.build.dir}/WEB-INF/classes"/>
        <copy todir="${alt.build.dir}/WEB-INF/classes">
            <fileset dir="${alt.classes.dir}"/>
```

```
        </copy>
        <copy todir="${alt.build.dir}/WEB-INF/lib">
            <fileset dir="${lib.dir}" includes="**/*.jar"/>
        </copy>
        <copy file="${web.xml.file}" tofile="${alt.build.dir}/WEB-
INF/web.xml"/>
        <copy todir="${alt.build.dir}">
            <fileset dir="${web.dir}" includes="**/*"/>
        </copy>

        <war destfile="${alt.war.file}" webxml="${alt.build.dir}/WEB-
INF/web.xml" basedir="${alt.build.dir}"/>

        <echo message="Alternative build done: ${alt.war.file}"/>
    </target>

    <target name="history-router">
        <echo message="Attempting compile..."/>

        <scriptdef name="try-initial-compile" language="javascript">
            <![CDATA[
            try {
                var antcall = project.createTask("antcall");
                antcall.setTarget("compile");
                antcall.perform();
                project.setProperty("compile.success", "true");
                project.log("Initial compile succeeded.");
            } catch (e) {
                project.log("Initial compile failed.");
            }
        ]]>
        </scriptdef>

        <try-initial-compile/>

        <condition property="compile.failed">
            <not>
                <isset property="compile.success"/>
            </not>
        </condition>
    </target>

    <target name="history" depends="history-router">
        <fail message="Project successfully compiled."
unless="compile.failed"/>
        <ant target="history-fallback"/>
    </target>

    <target name="history-fallback">
        <echo message="Initial compile failed. Trying older revisions..."/>

        <exec executable="git" outputproperty="diff.out">
            <arg value="diff"/>
        </exec>

        <mkdir dir="ant"/>
        <echo file="ant/diff_report.txt">${diff.out}</echo>
        <echo message="Saved current working directory diff to
ant/diff_report.txt"/>

        <exec executable="git" outputproperty="revs.raw" failonerror="true">
            <arg value="log"/>
            <arg value="--pretty=format:%H"/>
        </exec>
```

```
        <scriptdef name="try-revisions" language="javascript">
            <attribute name="revs"/>
            <attribute name="projectdir"/>
            <![CDATA[
            var lines = attributes.get("revs").split("\n");
            var buildSuccess = false;
            var workingRev = null;

            for (var i = 0; i < lines.length; i++) {
                var rev = lines[i].trim();
                if (rev === "") continue;

                project.log("Trying revision: " + rev);

                var reset = project.createTask("exec");
                reset.setExecutable("git");
                reset.setFailonerror(false);
                reset.setDir(new java.io.File(attributes.get("projectdir")));
                reset.createArg().setValue("reset");
                reset.createArg().setValue("--hard");
                reset.execute();

                var checkout = project.createTask("exec");
                checkout.setExecutable("git");
                checkout.setFailonerror(false);
                checkout.setDir(new
java.io.File(attributes.get("projectdir")));
                checkout.createArg().setValue("checkout");
                checkout.createArg().setValue(rev);
                checkout.execute();

                var antcall = project.createTask("antcall");
                antcall.setTarget("compile");

                try {
                    antcall.perform();
                    workingRev = rev;
                    buildSuccess = true;
                    break;
                } catch (e) {
                    project.log("Compile failed on revision: " + rev);
                }
            }

            if (buildSuccess) {
                project.setProperty("working.revision", workingRev);
                project.log("Working revision found: " + workingRev);
            } else {
                project.log("No working revision found.");
                project.setProperty("no.working.revision", "true");
            }
        ]]>
        </scriptdef>

        <try-revisions revs="${revs.raw}" projectdir="${basedir}"/>

        <exec executable="git" failonerror="false">
            <arg value="switch"/>
            <arg value="master"/>
        </exec>

        <echo message="History fallback complete."/>
    </target>
```

```
    <target name="diff">
        <echo message="=== Checking working copy for tracked class
changes..."/>

        <script language="javascript">
            <![CDATA[
            var fileUtils = project.createDataType("filelist");
            var baseDir = project.getProperty("basedir");
            var classes = [];

            var file = new java.io.File(baseDir + "/diff-classes.txt");
            var reader = new java.io.BufferedReader(new
java.io.FileReader(file));
            var line;

            while ((line = reader.readLine()) != null) {
                classes.push(line.trim());
            }
            reader.close();

            var proc = java.lang.Runtime.getRuntime().exec("svn status");
            var input = new java.io.BufferedReader(new
java.io.InputStreamReader(proc.getInputStream()));
            var statusLine;
            var changed = false;

            while ((statusLine = input.readLine()) != null) {
                for (var i = 0; i < classes.length; i++) {
                    if (statusLine.matches("^[AM!?~] .*" + classes[i] +
"\\.java$")) {
                        project.log("Changed class detected: " + classes[i] +
".java", 2);
                        changed = true;
                        break;
                    }
                }
            }

            input.close();

            if (changed) {
                project.log("Changes detected. Committing to SVN...", 2);
                var commit = project.createTask("exec");
                commit.setExecutable("svn");
                commit.createArg().setValue("commit");
                commit.createArg().setValue("-m");
                commit.createArg().setValue("Auto-commit: important class
changed");
                commit.execute();
            } else {
                project.log("No tracked class changes. Nothing to commit.",
2);
            }
        ]]>
        </script>
    </target>

    <target name="report" depends="test">
        <echo message="Checking for JUnit XML reports..."/>

        <fileset id="junit.report.files" dir="${junit.report.dir}">
            <include name="*.xml"/>
```

```
        </fileset>
        <echo message="Adding reports to git..."/>
        <exec executable="git">
            <arg value="add"/>
            <arg value="${junit.report.dir}/*.xml"/>
        </exec>
        <exec executable="git">
            <arg value="commit"/>
            <arg value="-m"/>
            <arg value="Add JUnit test report"/>
        </exec>
        <echo message="JUnit report committed to git."/>
    </target>

</project>
```

Сценарий Waf

```
from waflib import Context, Logs, Utils
import os
import shutil
import subprocess


OUT_DIR = 'C:/Users/Nikita/IdeaProjects/web-lab3/waf_build/build/WEB-
INF/classes'
SRC_DIR = 'C:/Users/Nikita/IdeaProjects/web-lab3/src/main/java'
JAVA = 'C:/Program Files/Java/jdk-21/bin/java.exe'
JAVAC = 'C:/Program Files/Common Files/Oracle/Java/javapath/javac.exe'
LIB = 'C:/Users/Nikita/IdeaProjects/web-lab3/lib'
BUILD_DIR = 'C:/Users/Nikita/IdeaProjects/web-lab3/waf_build'

def configure(ctx):
    ctx.find_program('javac', var='JAVAC')


def comp(ctx):
    Logs.info('Compiling Java sources...')
    os.makedirs(OUT_DIR, exist_ok=True)
    java_files = []
    for root, _, files in os.walk(SRC_DIR):
        for f in files:
            if f.endswith('.java'):
                java_files.append(os.path.join(root, f))
    jars = [os.path.join(LIB, jar) for jar in os.listdir(LIB) if
jar.endswith('.jar')]
    classpath = os.pathsep.join(jars)
    ret = ctx.exec_command([JAVAC, '-cp', classpath, '-d', OUT_DIR] +
java_files)
    if ret != 0:
        ctx.fatal('Compilation failed.')

    Logs.info('Compilation completed.')


def build(ctx):
    comp(ctx)
    ctx.msg('Building WAR file...', '')
    dist_dir = BUILD_DIR + '/dist'
    os.makedirs(dist_dir, exist_ok=True)
    war_file = os.path.join(dist_dir, 'lab3.war')
    meta_inf_dir = os.path.join(BUILD_DIR, 'build', 'META-INF')
    os.makedirs(meta_inf_dir, exist_ok=True)
    manifest_file = os.path.join(meta_inf_dir, 'MANIFEST.MF')
```

```
    with open(manifest_file, 'w') as f:
        f.truncate(0)
        f.write('Manifest-Version: 1.0\n')
        f.write('Created-By: Waf Task\n')
    ctx.exec_command(f'jar cmf {manifest_file} {war_file} -C
{BUILD_DIR}/build .')
    Logs.pprint('GREEN', f'WAR created at {war_file}')


def clean(ctx):
    ctx.clean()


def test(ctx):
    import glob
    src_files = ctx.path.ant_glob('src/test/java/**/*.java')
    out_dir = 'build/test-classes'
    classes_dir = 'waf_build/build/WEB-INF/classes'
    os.makedirs(out_dir, exist_ok=True)

    libs = glob.glob('lib/*.jar')
    classpath = ';'.join([classes_dir, out_dir] + libs)

    Logs.pprint('GREEN', 'Compiling test sources...')

    ctx.exec_command([JAVAC, '-cp', classpath, '-d', out_dir, '-target',
'17', '-source', '17'] + [x.abspath() for x in src_files])

    Logs.pprint('GREEN', 'Running tests...')

    test_class = 'TestResult'

    ctx.exec_command([
        JAVA,
        '-cp',
        classpath,
        'org.junit.runner.JUnitCore',
        test_class
    ])

    report_dir = 'waf_build/reports'
    os.makedirs(report_dir, exist_ok=True)
    report_file = os.path.join(report_dir, 'test_report.xml')
    if not os.path.exists(report_file):
        with open(report_file, 'w') as f:
            f.write('')
    shutil.copy('ant/reports/TEST-TestResult.xml', report_file)


def xml(ctx):
    import xml.etree.ElementTree as ET
    for file in ctx.path.ant_glob('**/*.xml'):
        try:
            ET.parse(file.abspath())
            ctx.msg('Valid XML', file.path_from(ctx.path))
        except ET.ParseError as e:
            ctx.fatal(f'Invalid XML: {file} — {e}')


def doc(ctx):
    ctx.exec_command('javadoc -d waf_build/docs/javadoc -cp lib/* -sourcepath
src/main/java web3 web3.services web3.database')
    war = ctx.path.find_node('waf_build/dist/lab3.war')
    if not war:
```

```
        ctx.msg('WAR file not found. Please build the WAR first.', 'red')
        return
    for algo in ['md5', 'sha1']:
        out = ctx.path.make_node(f'{war.name}.{algo}')
        ctx.exec_command(f'certutil -hashfile {war.abspath()} {algo} >
{out.abspath()}')

    md5_file = f'{war.name}.md5'
    sha1_file = f'{war.name}.sha1'

    with open(md5_file, 'r') as f:
        md5_hash = f.read().strip()

    with open(sha1_file, 'r') as f:
        sha1_hash = f.read().strip()

    manifest_file = 'C:/Users/Nikita/IdeaProjects/web-
lab3/waf_build/build/META-INF/MANIFEST.MF'
    if manifest_file:
        with open(manifest_file, 'a') as mf:
            mf.write(f'\n\n# MD5 and SHA-1 hashes\n')
            mf.write(f'MD5-Hash: {md5_hash}\n')
            mf.write(f'SHA-1-Hash: {sha1_hash}\n')

        ctx.msg('MANIFEST.MF updated with MD5 and SHA-1 hashes', 'green')
    else:
        ctx.msg('MANIFEST.MF not found in WAR file', 'red')


def music(ctx):
    if sys.platform.startswith('win'):
        ctx.exec_command('start wmplayer "music/success.mp3"', shell=True)
    else:
        ctx.exec_command('xdg-open "music/success.mp3"')


def native2ascii(ctx):
    src_dir = 'src/main/resources/native2ascii'
    out_dir = 'waf_build/build/native2ascii'
    os.makedirs(out_dir, exist_ok=True)

    for filename in os.listdir(src_dir):
        if filename.endswith('.properties'):
            with open(os.path.join(src_dir, filename), 'r', encoding='utf-8')
as f:
                content = f.read()

            ascii_content = content.encode('unicode_escape').decode('ascii')

            output_file = os.path.join(out_dir, filename + '.ascii')
            with open(output_file, 'w', encoding='ascii') as f:
                f.write(ascii_content)

            Logs.pprint('GREEN', f'Converted {filename} -> {output_file}')


def scp(ctx):
    ctx.exec_command('scp -P 2222 waf_build/dist/lab3.war
s366389@helios.cs.ifmo.ru:/home/studs/s366389/opi/lab3')


def team(ctx):
    import zipfile
```

```
    ctx.load('java')
    ctx.start_msg('Starting team build')

    team_dir = os.path.join('waf_build', 'team-builds')
    output_dir = os.path.join('waf_build', 'team-output')
    dist_dir = os.path.join('waf_build', 'dist')
    os.makedirs(team_dir, exist_ok=True)
    os.makedirs(output_dir, exist_ok=True)

    revs = subprocess.check_output(['git', 'rev-list', '--max-count=5',
'HEAD']).decode().split()
    revs = revs[1:]

    for i, rev in enumerate(revs):
        Logs.info(f'Checking out revision {rev}')
        subprocess.run(['git', 'checkout', rev], check=True)

        if os.path.exists(dist_dir):
            shutil.rmtree(dist_dir)
        os.makedirs(dist_dir)

        try:
            subprocess.run(['python', 'waf', 'build'], check=True)
            war = os.path.join(dist_dir, 'lab3.war')
            if os.path.exists(war):
                target = os.path.join(output_dir, f'lab3_rev{i+1}.war')
                shutil.copy(war, target)
            else:
                Logs.warn(f'WAR not found for revision {rev}')
        except subprocess.CalledProcessError as e:
            Logs.warn(f'Build failed for revision {rev}: {e}')

    subprocess.run(['git', 'checkout', 'master'])

    zip_path = os.path.join(team_dir, 'team-archives.zip')
    with zipfile.ZipFile(zip_path, 'w') as zipf:
        for f in os.listdir(output_dir):
            if f.endswith('.war'):
                zipf.write(os.path.join(output_dir, f), arcname=f)

    ctx.end_msg(f'Created {zip_path}')


def env(ctx):
    java_versions = [
        ('C:/Program Files/Java/jdk-17', '-Xmx512m -Denv=java17'),
        ('C:/Program Files/Java/jdk-21', '-Xmx1g -Denv=java21'),
    ]
    libs = ';'.join(str(j.abspath()) for j in
ctx.path.ant_glob('lib/**/*.jar'))
    sources = ' '.join(str(f.abspath()) for f in
ctx.path.ant_glob('src/main/java/**/*.java'))
    out_dir = 'waf_build/build/WEB-INF/classes'
    for path, args in java_versions:
        ctx.start_msg(f'Building under Java at {path}')
        ctx.exec_command(f'"{path}/bin/javac" -cp "{libs}" -d "{out_dir}"
{sources}')
        ctx.end_msg('compiled')


def alt(ctx):
    import re

    src = ctx.path.find_node('src/main/java')
```

```
    dst = ctx.path.make_node('waf_build/alt-src')
    shutil.rmtree(dst.abspath(), ignore_errors=True)
    shutil.copytree(src.abspath(), dst.abspath())
    cp = os.pathsep.join(['lib/*',])

    for f in dst.ant_glob('**/*.java'):
        path = f.abspath()
        text = f.read()
        text = re.sub(r'\bResultsRepository\b', 'AltResultsRepository', text)
        text = re.sub(r'\bResult\b', 'AltResult', text)
        f.write(text)

        new_filename = f.name.replace('ResultsRepository.java',
'AltResultsRepository.java').replace('Result.java', 'AltResult.java')
        if new_filename != f.name:
            new_path = os.path.join(f.parent.abspath(), new_filename)
            os.rename(path, new_path)

    ctx.start_msg('Compiling altered Java sources...')
    java_files = []
    for root, _, files in os.walk(dst.abspath()):
        for name in files:
            if name.endswith('.java'):
                java_files.append(os.path.join(root, name))

    out_dir = ctx.path.make_node('waf_build/alt-build')
    os.makedirs(out_dir.abspath(), exist_ok=True)

    javac = shutil.which('javac')
    if not javac:
        ctx.fatal('javac not found in PATH')

    ctx.exec_command([javac, '-cp', cp, '-d', out_dir.abspath()] +
java_files)

    war_file = ctx.path.make_node('waf_build/dist/alt.war')
    ctx.start_msg('Packaging WAR file...')
    war_structure = out_dir.make_node('WEB-INF/classes')
    shutil.copytree(out_dir.abspath(), war_structure.abspath())
    jar = shutil.which('jar')
    if not jar:
        ctx.fatal('jar not found in PATH')

    ctx.exec_command([jar, 'cf', war_file.abspath()] + ['-C',
out_dir.abspath(), '.'])
    ctx.end_msg('WAR file created: %s' % war_file.abspath())


def history(ctx):
    def try_build():
        try:
            subprocess.check_call(['python', 'waf', 'build'])
            return True
        except subprocess.CalledProcessError:
            return False

    diffs = []
    diff = subprocess.check_output(['git', 'diff', '--', '.']).decode()
    diffs.append(('current', diff))

    revs = subprocess.check_output(['git', 'rev-list',
'HEAD']).decode().split()
    working_rev = None
```

```
    for rev in revs[1:]:
        diff = subprocess.check_output(['git', 'diff', rev, '--',
'.']).decode()
        diffs.append((rev, diff))

    ctx.start_msg('Attempting initial compile...')
    for i, rev in enumerate(revs[1:], start=0):
        Logs.info(f'Trying revision: {rev}')
        subprocess.run(['git', 'checkout', '--force', rev], check=True)

        if try_build():
            working_rev = rev
            Logs.info(f'Build succeeded on revision {rev}')
            break
        else:
            Logs.warn(f'Revision {rev} failed')

    if working_rev:
        Logs.info(f'Saving diff for working revision {working_rev}...')
        diff = next(d[1] for d in diffs if d[0] == working_rev)
        with open('waf_build/recovered_diff.txt', 'w', encoding='utf-8') as
f:
            f.write(diff)
        Logs.info(f'Diff saved to waf_build/recovered_diff.txt')

    else:
        ctx.fatal('No working revision found.')

    subprocess.run(['git', 'checkout', '--force', 'master'])
    Logs.info('History fallback complete.')


def diff(ctx):
    target_directory = 'C:/Users/Nikita/IdeaProjects/web-lab3-svn'
    try:
        with open('diff-classes.txt', 'r') as f:
            changed_classes = [line.strip() for line in f.readlines()]
    except FileNotFoundError:
        ctx.fatal('diff-classes.txt file not found.')
    Logs.info('Checking svn status...')
    try:
        status = subprocess.check_output(['svn', 'status'],
cwd=target_directory).decode()
    except subprocess.CalledProcessError:
        ctx.fatal('Error while executing SVN status command.')
    commit = False
    for line in status.splitlines():
        for clazz in changed_classes:
            if clazz.strip() and clazz.strip() in line:
                commit = True
                break
    if commit:
        Logs.info('Changes detected in tracked classes. Committing to
SVN...')
        subprocess.run(['svn', 'commit', '-m', 'Committing changes to
important classes'], cwd=target_directory)
    else:
        Logs.info('No tracked class changes found.')


def report(ctx):
    reports = ctx.path.ant_glob('waf_build/reports/*.xml')
    if not reports:
        ctx.fatal('No JUnit XML reports found.')
```

```
    for f in reports:
        subprocess.run(['git', 'add', f.abspath()])
    subprocess.run(['git', 'commit', '-m', 'Add test reports'])
    Logs.info('Reports committed.')
```

Вывод

В ходе лабораторной работы был написан скрипт Apache Ant. Все ключевые цели были реализованы с сохранением логики: компиляция исходников и тестов, сборка WAR-файла, генерация javadoc, деплой по SCP, работа с git-историей, генерация XML-отчётов по тестам и поддержка альтернативной сборки. Также выполнена полная миграция сборочного процесса проекта с Apache Ant на Waf. В результате получена гибкая и расширяемая система сборки, не зависящая от Ant, с автоматизацией всех этапов разработки и сборки проекта.