Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра Системного Программирования

Щербаков Александр Владимирович

Реализация инструмента для непрерывной интеграции для IBM z/OS с поддержкой средства управления конфигурациями IBM SCLM

Дипломная работа

Допущена к защите.
Зав. кафедрой:
д-р. физ.-мат. наук, профессор Терехов А. Н.

Научный руководитель:
д-р. физ.-мат. наук, профессор Нестеров В. М.

Рецензент:
спец., рук.гр. Хлебников П. Н.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Department of Software Engineering

Aleksandr Shcherbakov

Implementation of continuous integration tool for IBM z/OS with support of IBM SCLM (IBM Software Configuration and Library Manager)

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terekhov

Scientific supervisor:
professor Vyacheslav Nesterov

Reviewer:
team lead Khlebnikov Pavel

Saint-Petersburg
2015

Оглавление

Введение

Мейнфреймы [41] появились более 50 лет назад[1], но, не смотря на это, платформа продолжает оставаться актуальной и активно развивается[2]. Дело в том, что мейнфреймы предоставляют пользователям колоссальную надёжность и безопасность, поэтому на них работают многие отрасли, в которых крайне важна эти параметры.

Например, простои могут быть связаны с огромными материальными потерями, поэтому дешевле оказывается заплатить за более надёжную платформу, чем впоследствии терпеть убытки из-за какой-либо ошибки. Такими требовательными отраслями являются банки, а также государственные, коммуникационные и фармацевтические компании.

Существует огромный запас работающего ПО для мейнфреймов, которое было написано много лет назад и относится к так называемому классу "legacy code" [35]. Поддержка этого кода дорога, но стоимость его реинжиниринга или перехода на другую платформу значительно выше, поэтому многие компании продолжают выпускать новые версии ПО для мейнфреймов.

Для того, чтобы увеличить скорость разработки ПО и уменьшить её стоимость, а также повысить качество продукта, применяются различные подходы, одним из которых является использование гибких методологий разработки [38].

При разработке ПО для мейнфреймов также применяются гибкие методологии, например, Scrum [36, 29]. К сожалению, практически нет инструментов для автоматизации некоторых практик в таких методологиях. Одной из таких "обделённых" практик можно назвать непрерывную интеграцию [42]. Не существует инструментов, находящихся в открытом доступе, которые позволяли бы настроить непрерывную интеграцию с использованием SCM[3] SCLM[4].

[1]Датой рождения платформы принято считать выпуск IBM System/360 7 апреля 1964 года [33].

[2]В начале 2015 года компания IBM анонсировала новую линейку мейнфреймов, первым представителем которой стал z13 [8].

[3]Система управления конфигурациями [40]. В данном случае,интересующая функциональность во многом совпадает с системами управления версиями [45].

[4]IBM Software Configuration and Library Manager [15, 16, 17].

Целью данной работы является создание инструмента для непрерывной интеграции, который можно применять при разработке ПО для мейнфреймов с использованием SCLM в качестве системы управления изменениями, а также анализ влияния использования непрерывной интеграции на процесс разработки такого ПО.

Для достижения этой цели были поставлены следующие задачи:

- Создание прототипа инструмента для непрерывной интеграции для ОС IBM z/OS с поддержкой SCLM.
- Внедрение разработанного прототипа и практики непрерывной интеграции в процесс разработки ПО для мейнфреймов.
- Анализ результатов разработки инструмента и его внедрения.

Для разработки инструмента был выбран SCLM, так как он поставляется в комплекте с ISPF[5], входящим в пакет базовой поставки IBM z/OS.

Использование непрерывной интеграции в проектах, связанных с разработкой ПО для мейнфреймов, оказывает своё влияние как на процесс разработки, так и на процесс тестирования ПО. В данной работе будет представлено описание изменений, которым подвергаются процессы разработки и тестирования ПО для мейнфреймов при использовании непрерывной интеграции на примере разработки ПО для IBM z/OS.

В частях, посвящённым инструментальной поддержке (разделы 2, 3), будет дано описание двух прототипов, помогающих настроить непрерывную интеграцию на IBM z/OS, а также результатов, которых можно добиться при использовании этих инструментов.

[5]Interactive System Productivity Facility [16, 17].

1. Обзор решений

Настроить непрерывную интеграцию на мейнфреймах можно несколькими способами:

1. Использование готового средства автоматизации.
2. Разработка собственного инструмента для регулярного и настраиваемого запуска пакетных заданий.
3. Создание расширения для уже имеющегося средства автоматизации непрерывной интеграции.

1.1. Использование готового средства автоматизации

Первоначально было проведено исследование уже существующих решений, способных помочь настроить непрерывную интеграцию для SCLM. В результате, были исследованы доступные средства автоматизации, а также средства, имеющие поддержку SCLM.

Существует довольно много известных средств автоматизации для мейнфреймов, например: IBM Tivoli NetView for z/OS [21], VitalSigns for Network Automation and Control (VNAC) for z/OS [27], Operator Dynamic Dialog Subsystem (ODDS) [28], BMC MainView Automation [3], однако, эти средства не интегрированы с SCLM.

С другой стороны, есть совсем немного ПО, способного взаимодействовать с SCLM. Наиболее известными являются RDz [22] и SCLM Developer Toolkit [23]. К сожалению, оба эти инструмента не являются средствами автоматизации, а только позволяют использовать SCLM в качестве системы хранения версий, поэтому не применимы для решения поставленной задачи.

С учётом того, что все эти инструменты обладают закрытым исходным кодом и проприетарной лицензией, было принято решение разрабатывать собственный инструмент.

1.2. Самостоятельная разработка инструмента для непрерывной интеграции

Одним из вариантов решения поставленной задачи являлась самостоятельная разработка необходимого инструмента.

Ключевыми возможностями нового инструмента должны были стать:

- Расписание – Возможность регулярно запускать проверку SCLM на наличие изменений и тестирование нового кода.
- Гибкая настройка окружения – Возможность указать для каждого отслеживаемого проекта в SCLM что, где и когда должно запуститься.
- Отчётность – Возможность создания и хранения отчётов о проделанной работе. Также желательна возможность оперативного оповещения разработчиков о состоянии тестирования (например, отправка электронного письма).

При исследовании этого подхода был создан планировщик запуска пакетных заданий, с функциональностью, похожей на Cron (см. раздел 2).

1.3. Расширение для существующего средства автоматизации непрерывной интеграции

При выборе способа реализации инструмента для непрерывной интеграции одним из вариантов естественно стало расширение функционала существующего инструмента.

Сравнивались такие инструменты как:

- Jenkins CI [25]
- GitLab CI [7]
- Go [30]

При выборе инструмента для расширения было проведено их сравнение по функциональности, открытости, гибкости, расширяемости и документации. В результате, выбор пал на Jenkins CI:

- Функциональность – В сравнении[6] ПО для непрерывной интеграции Jenkins CI является продуктом, обладающим одним из самых широких списков функций и поддерживаемых платформ.

- Открытость – Jenkins CI обладает открытой лицензией[7], что позволяет любому как улучшать саму систему, так и расширять её возможности посредством дополнений.

- Гибкость – Благодаря своим преимуществам Jenkins CI получил огромную поддержку среди разработчиков ПО с открытым исходным кодом. В частности, для Jenkins CI написано множество дополнений, позволяющих настроить непрерывную интеграцию практически в любом процессе разработки ПО.

- Расширяемость – Jenkins CI является, по своей сути, основой для настройки пользовательского процесса непрерывной интеграции. Для того, чтобы Jenkins CI гармонично вписался в пользовательский процесс, необходима либо установка подходящих расширений, либо разработка собственных. Во втором случае новое расширение можно выложить в библиотеку расширений для Jenkins CI, где оно станет доступно и для других пользователей Jenkins CI.

- Документация – Одним из важнейших параметров при выборе платформы стала хорошая документация, позволяющая быстро найти ответы на вопросы, возникающие во время разработки дополнений, а также содержащая подробные инструкции для тех, кто только начинает разрабатывать под Jenkins CI.

[6]Comparison of continuous integration software [31].

[7]Jenkins CI распространяется по лицензии MIT.

2. Решение №1: Serv

При решении поставленных задач было создано два прототипа, помогающих автоматизировать работу, связанную с непрерывной интеграцией. Одним из этих решений является приложение Serv [46], написанное с помощью C и Rexx [9, 13] и полностью работающее в среде ОС IBM z/OS.

Параллельно с Serv шла разработка IBM zOS Connector – дополнения к популярному серверу для непрерывной интеграции – Jenkins CI. При сравнении результатов, достигнутых после месяца разработки, было принято решение приостановить разработку Serv, так как функциональность, которую удалось реализовать за одно и то же время, отличалась в разы. Более подробно дополнение IBM zOS Connector рассмотрено в разделе 3.

2.1. Описание

В своей текущей реализации Serv представляет собой планировщик запуска пакетных заданий, написанных на JCL[8], обладающий следующими возможностями:

1. Регулярно запускать пакетные задания (по одному), написанные на JCL и упомянутые в файле настройки (пример использования и файла настройки будет в разделе 2.4)

2. Поддержка MVS[9]-команд /STOP (сокращённо – /P) и /MODIFY (сокращённо – /F) (подробнее команды MVS рассмотрены в [14]):

   - /P – Останавливает Serv сразу после завершения текущего работающего пакетного задания, если такое есть.

[8] Job Control Language [12, 11].

[9] Multiple Virtual Storage - ОС для System/360 и System/370, разработанная компанией IBM. В дальнейшем своём развитии эта ОС была переименована в z/OS, однако часть функций всё ещё ссылаются на старое имя.

- /F *jobname*,APPL=REFRESH – заставляет Serv перечитать файл настроек с последующим перестроением списка запускаемых пакетных заданий.

В целом, реализованная функциональность мало чем отличается от функциональности планировщика Cron [37], доступного в IBM z/OS в среде UNIX System Services [20].

2.2. Архитектура

Serv написан на C и Rexx и состоит из трёх основных модулей:

1. SERVMAIN – Написан на C, отвечает за основной функционал:
   - Отправление пакетных заданий на запуск.
   - Ведение журнала запусков.
   - Считывание настроек и перестроение списка запускаемых пакетных заданий.
   - Отклик на команды MVS.
2. JOBLIST – Написан на C, отвечает за операции с пакетными заданиями с точки зрения C:
   - Создание и удаление внутреннего представления пакетного задания.
   - Ведение списка пакетных заданий.
   - Передача пакетного задания для исполнения модулю SUBRC.
3. SUBRC – Написан на Rexx, отвечает за работу с пакетным заданием:
   - Непосредственный запуск пакетного задания.
   - Ожидание завершения пакетного задания.
   - Возвращение кода завершения (CC[10]) пакетного задания модулю JOBLIST.

[10]Condition Code.

2.3. Описание работы

Serv предназначен для периодического запуска пакетных заданий. После завершения очередного пакетного задания Serv пишет запись в журнал, содержащий информацию о том, какое конкретно пакетное задание было запущено, в какое время, а также с каким кодом завершения оно закончило свою работу.

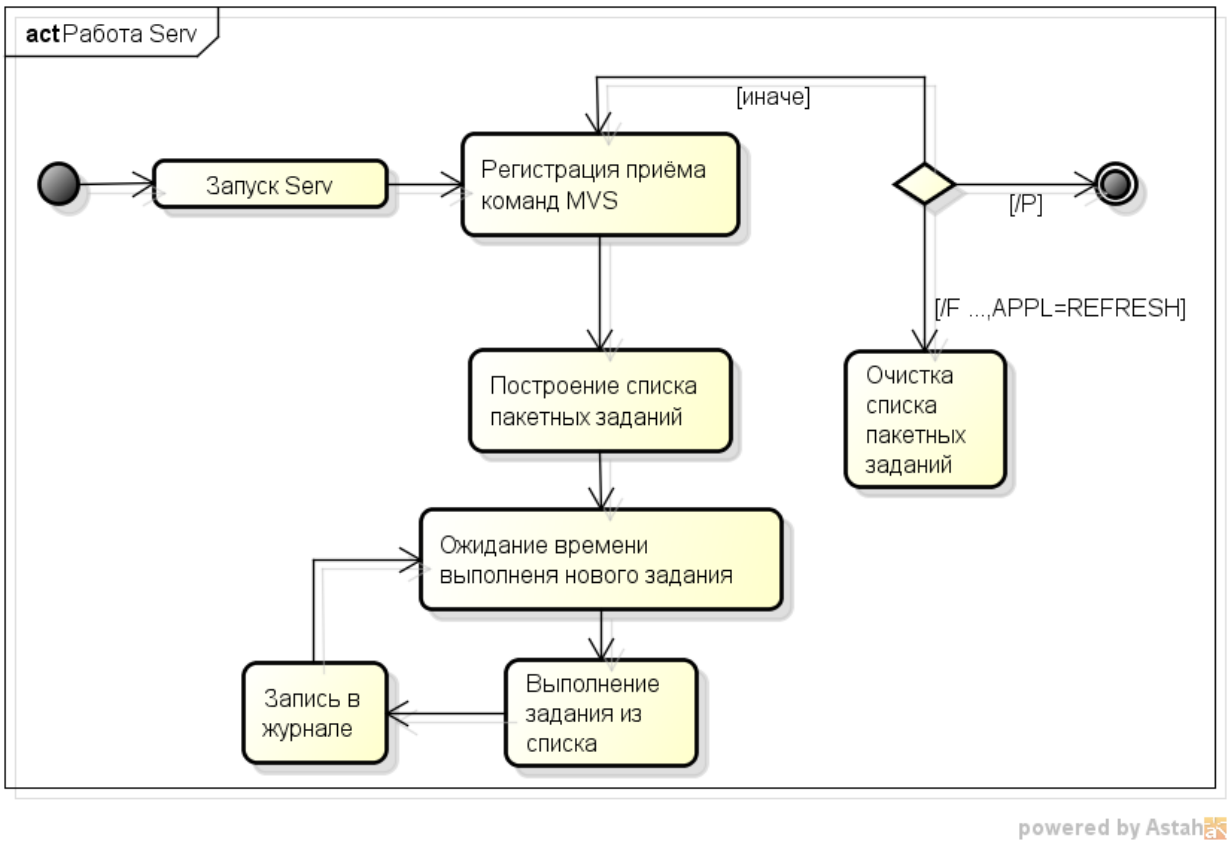

Рис. 1: Serv – Диаграмма видов деятельности.

При правильной настройке и использовании Serv выполняется следующий набор действий (см. Рис. 1):

1. Запуск Serv.

    - Пример пакетного задания, используемого для запуска Serv, можно найти в Приложении, в разделе 6.1.1.

2. SERVMAIN начинает принимать команды MVS и прерывает работу после завершения текущего выполняемого пакетного задания (если такое есть).

- /P – остановка Serv и /F – перечитывание файла настроек и перестроение списка пакетных заданий.

3. SERVMAIN читает файл настроек и строит список пакетных заданий с помощью JOBLIST.

   - Пример файла настроек можно найти в Приложении, в разделе 6.1.2.

4. После этого выполняется следующий цикл (до прерывания командой MVS):

   (a) SERVMAIN ожидает время запуска первого пакетного задания в списке.

   (b) JOBLIST перемещает пакетное задание в списке с указанием следующего времени запуска.

   (c) JOBLIST с помощью SUBRC запускает пакетное задание.

   (d) SUBRC ожидает окончания пакетного задания.

   (e) SERVMAIN делает запись в журнале о результате выполнения пакетного задания (CC).

2.4. Пример использования

Рассмотрим простой пример использования Serv. В качестве пакетного задания для запуска и файла настроек возьмём приведённые в разделе 6.1. Пакетные задания WAIT* обозначают простые JCL сценарии, выполняющие только ожидание определённого количества секунд и завершающиеся с CC = 0.

Выдадим команду `/F R$SERV$M,APPL=REFRESH`. Это заставит Serv обновить список заданий и запустить первое из них ещё раз.

После этого выдадим команду `/P R$SERV$M`. Это заставит Serv прекратить работу, как только текущее исполняемое пакетное задание завершится.

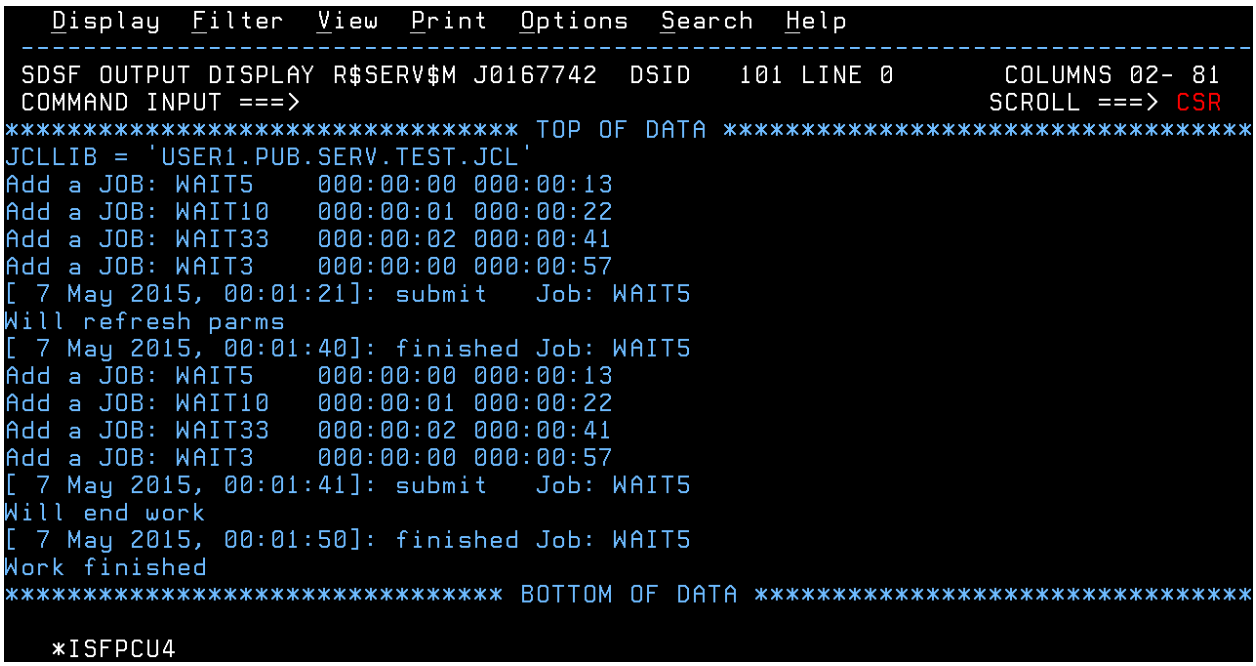

Рис. 2: Serv – Журнал работы.

В результате этих действий получится журнал работы Serv, который можно увидеть на Рис. 2.

Заметим, что Serv откликнулся на обе команды раньше, чем завершилось задание WAIT5, но дождался окончания его работы перед выполнением команд.

3. Решение №2: IBM zOS Connector

Вторым инструментом, который создавался параллельно с Serv, стал "IBM zOS Connector" – дополнение к популярному серверу для непрерывной интеграции Jenkins CI.

В разделе 3.1 описывается структура дополнения. Принцип работы и взаимодействие с IBM z/OS разбираются в разделе 3.2. Остальные разделы посвящены отдельным элементам IBM zOS Connector. Скриншоты внешнего вида, а также некоторые примеры настроек этого модуля можно найти в Приложении, в разделе 6.2.

3.1. Архитектура

IBM zOS Connector – дополнение, написанное на Java (логика работы приложения) и Apache Commons Jelly [2] (внешний вид).

На текущий момент IBM zOS Connector состоит из следующих элементов:

- zFTPConnector – Элемент позволяет подключаться к LPAR[11] по протоколу FTP, запускать пакетные задания и получать журналы их выполнения. Также он позволяет проверять CC из полученного журнала. Подробнее zFTPConnector рассмотрен в разделе 3.3.

- Поддержка SCLM – Данный элемент добавляет SCLM в качестве доступной SCM[12] в Jenkins CI. Это позволяет стандартным образом отслеживать изменения в проекте и запускать тестирование ***только*** при наличии изменений. Подробнее компонент рассмотрен в разделе 3.4.

- zOSJobSubmitter – Элемент, позволяющий Jenkins CI запускать пакетные задания на IBM z/OS, проверять их код завершения и сохранять их журналы выполнения. Рассмотрен в разделе 3.5.

[11] Logical Partition Access Resources – логический раздел мейнфрейма [34].

[12] Software Configuration Manager, Менеджер конфигураций [40].

3.2. Схема работы

В наиболее полном случае пользователь настроит в проекте[13] как запуск пакетных заданий, так и поддержку SCLM. Рассмотрим схему работы Jenkins CI в каждом из этих этапов.

3.2.1. Запуск пакетных заданий

При запуске пакетного задания IBM zOS Connector подключается с помощью zFTPConnector к LPAR и отправляет файл задания на исполнение. Затем, в зависимости от пользовательских настроек, он может дождаться окончания выполнения пакетного задания и сохранить журнал задания в Jenkins CI, а также удалить журнал работы из z/OS, если того пожелал пользователь. Подробнее этот процесс описан в разделе 3.3.1.

3.2.2. Поддержка SCLM

Для проверки проекта из SCLM на наличие изменений выдаётся команда DBUTIL (эта команда SCLM описана в [15]), которая позволяет получить информацию об актуальном состоянии файлов:

- Дату и время их последнего изменения.
- Имя пользователя, который совершил изменения.
- Название группы, откуда был перемещён изменённый файл, если изменение произошло в результате перемещения в группу.

После того, как информация получена, она записывается во временный файл *changelog.xml*, позволяющий позднее извлечь всю необходимую информацию о последних изменениях. С помощью этого файла заполняется страница изменений, а также определяется, нужно ли продолжать работу (изменения в проекте были) или же проект идентичен

[13]Речь идёт о проектах в Jenkins CI.

предыдущей ревизии (в таком случае нет нужды запускать дальнейшие шаги, так как их результат будет идентичен результату прошлого запуска). Этот процесс детально рассмотрен в разделе 3.4.1.

3.3. zFTPConnector

zFTPConnector является оболочкой вокруг класса FTPClient из библиотеки Apache commons-net [1]. Основу функциональности этого элемента составляет возможность общаться с JES посредством протокола FTP, описанная в [10]. Сделать это можно, например, используя следующую команду:

```
[quote] site filetype=jes jesjobname=*
```

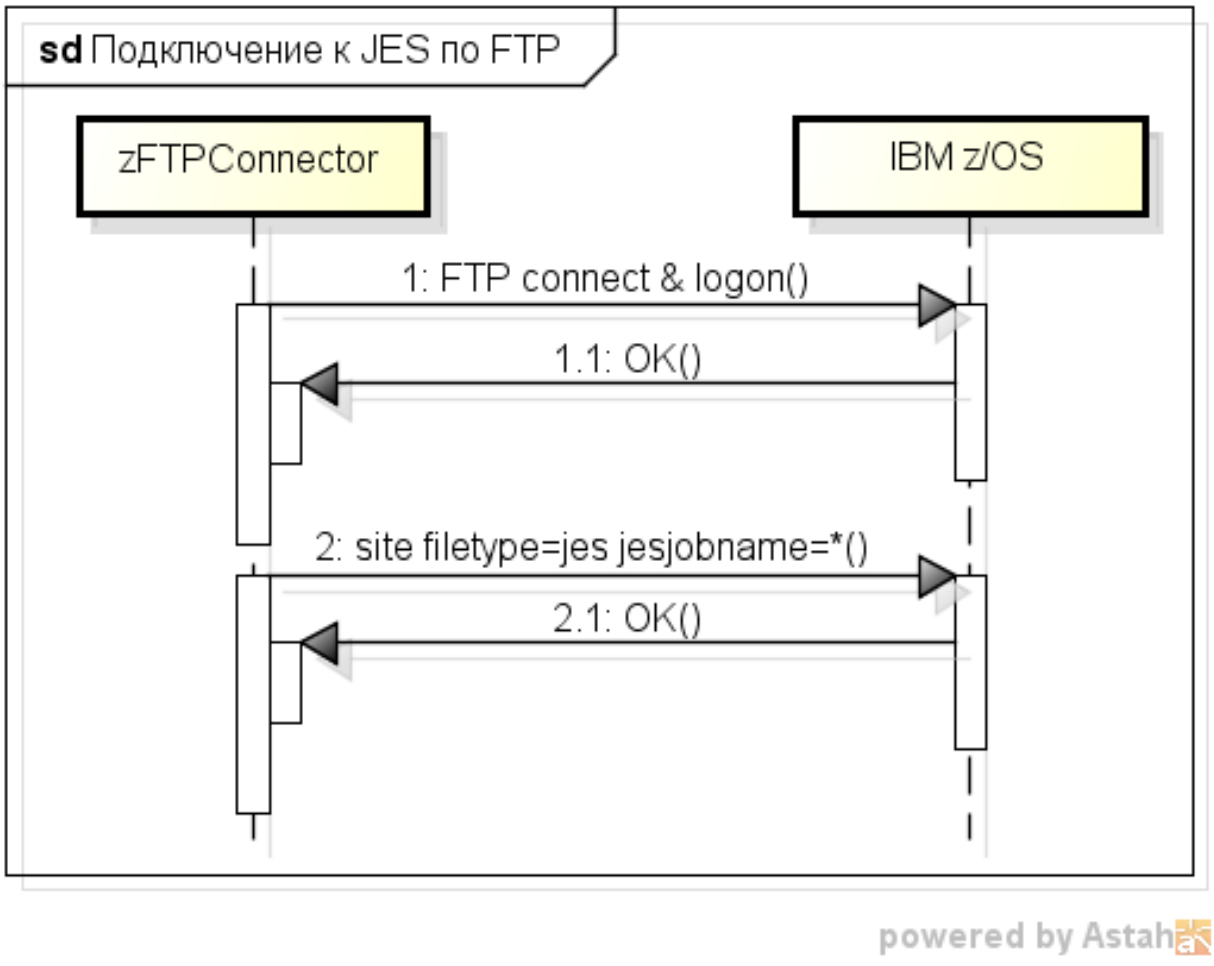

Рис. 3: Подключение zFTPConnector через FTP к JES.

С помощью zFTPConnector можно осуществлять:

- Подключение к IBM z/OS по FTP (см. Рис. 3).
- Отправку пакетных заданий в JES.
- Проверку статуса пакетного задания и получение его журнала работы.
- Удаление журнала работы пакетного задания из z/OS.

Для его использования важны следующие методы:

- Конструктор:
```
public zFTPConnector
    (String server,
    int port,
    String userID,
    String password)
```
  С помощью параметров конструктора задаются настройки для подключения к z/OS:
  - server – Имя или IP-адрес LPAR'a.
  - port – Доступный порт FTP.
  - userID – Имя пользователя, используемое для соединения.
  - password – пароль, используемый для соединения.
- Отправка пакетного задания на исполнение (см. раздел 3.3.1):
```
public boolean submit
    (InputStream inputStream,
    boolean wait,
    int waitTime,
    OutputStream outputStream,
    boolean deleteLogFromSpool)
```
  Этот метод позволяет отправить пакетное задание на исполнение, а также получить журнал исполнения с z/OS и удалить его оттуда:

- inputStream – java.io.InputStream, содержит пакетное задание для запуска.
- wait – Нужно ли дожидаться окончания работы задания.
- waitTime – Максимальное время ожидания работы задания в минутах (0 эквивалентен ∞).
- outputStream – java.io.OutputStream, в который будет записан журнал работы задания, если необходимо дождаться его завершения.
- deleteLogFromSpool – Указывает, нужно ли удалять журнал задания из z/OS (работает только если необходимо дождаться завершения задания).

- Получение дополнительной информации о пакетном задании – его JES[14] Job Id и CC:
  `public String getJobID()`
  `public String getJobCC()`

3.3.1. Запуск пакетного задания

Рассмотрим процесс исполнения пакетного задания подробней (см. Рис. 4). Предположим, что пользователь захотел дождаться завершения пакетного задания, указав waitTime = 0, а также удалить журнал работы из z/OS:

1. zFTPConnector подключается к z/OS по протоколу FTP, используя переданные в конструкторе параметры (см. Рис. 3).
2. zFTPConnector инициирует исполнение пакетного задания с помощью FTP PUT (таким образом содержимое файла передаётся на вход JES).
3. z/OS возвращает JES Job Id для этого задания, которое запоминается zFTPConnector.

[14] Job Entry Subsystem [18, 19].

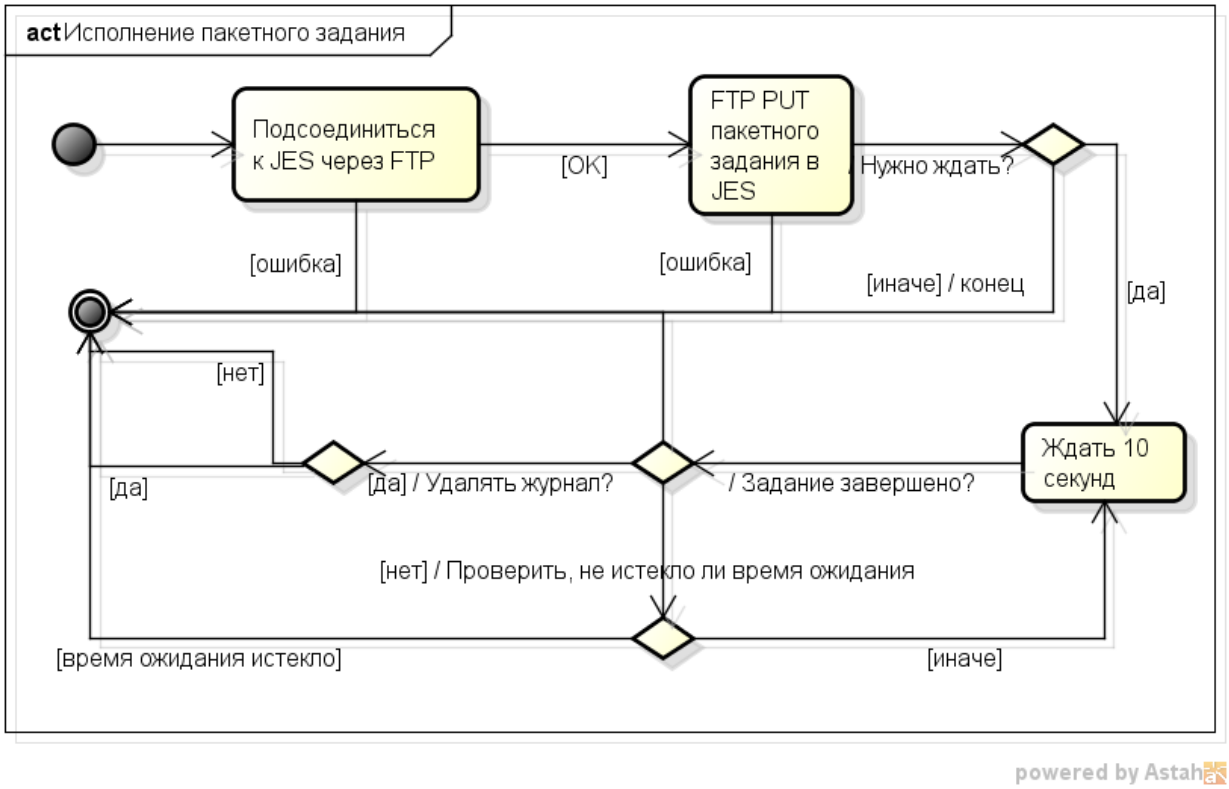

Рис. 4: Запуск пакетного задания с помощью zFTPConnector.

4. С интервалом в 10 секунд начинается проверка завершения пакетного задания по JES Job Id (используется команда FTP GET). Как только оно завершится, во внутренний буфер попадает журнал его работы.

5. zFTPConnector сканирует журнал выполнения пакетного задания и извлекает из него CC.

6. zFTPConnector удаляет журнал из z/OS с помощью FTP DELETE по JES Job Id.

3.4. Интеграция SCLM в качестве SCM в Jenkins CI

Для того, чтобы интегрировать SCLM в Jenkins CI, необходимо реализовать все функции, перечисленные в [26].

Основной трудностью, с которой приходится столкнуться, является то, что Jenkins CI рассчитан на современные системы с различными дополнительными функциями (например, сравнение двух коммитов).

Это связано с тем, что новые системы контроля версий хранят информацию в так называемых "наборах изменений"[15], в то время как SCLM хранит полные версии конкретного файла. Поэтому такой функционал пришлось реализовывать самостоятельно.

В этом разделе не будет описания методов, переопределённых для корректной работы Jenkins CI с SCLM, так как их подробное описание можно получить самостоятельно, сгенерировав JavaDoc из исходного кода [47].

3.4.1. Схема работы с SCLM

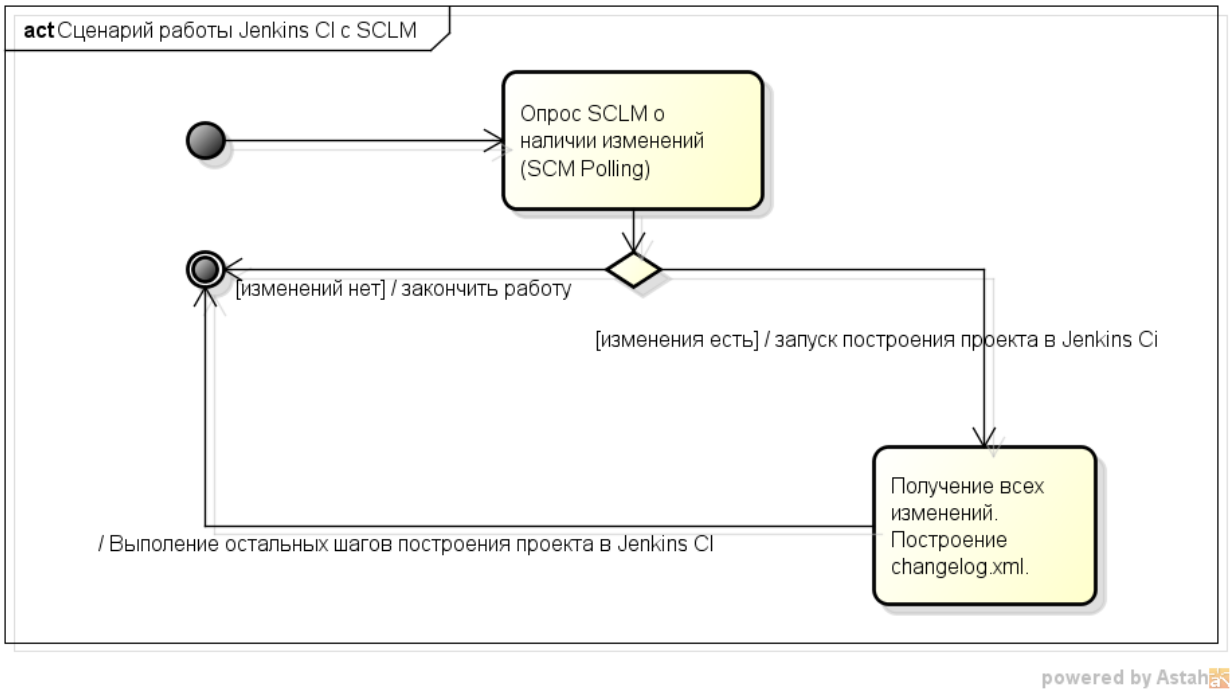

Рис. 5: Схема работы zOSConnector с SCLM.

В полном сценарии работы (с периодическим опросом[16] SCLM о наличии изменений), Jenkins CI выполняет следующие шаги (см. Рис. 5):

1. Опрос SCLM – Jenkins CI опрашивает SCLM с помощью метода SCLMSCM.compareRemoteRevisionWith и получает информацию о том, вносились ли изменения в интересующие пользовате-

[15] Change set.
[16] SCM Polling.

ля файлы. Если нет, построение проекта не инициируется. Иначе, производится переход на следующие шаги.

2. Получение текущего состояния файлов (checkout) – С помощью функции SCLMSCM.checkout Jenkins CI получает информацию о текущем состоянии файлов в SCLM, которую записывает во временный файл *changelog.xml*, используемый в дальнейшем для отображения изменений.

3. Расчёт и отображение изменений в SCLM – С помощью классов SCLMChangeLogSet, SCLMFileState, SCLMChangeLogParser и LogSet производится чтение файла *changelog.xml* и создание отчёта об изменениях (см. Рис. 6).

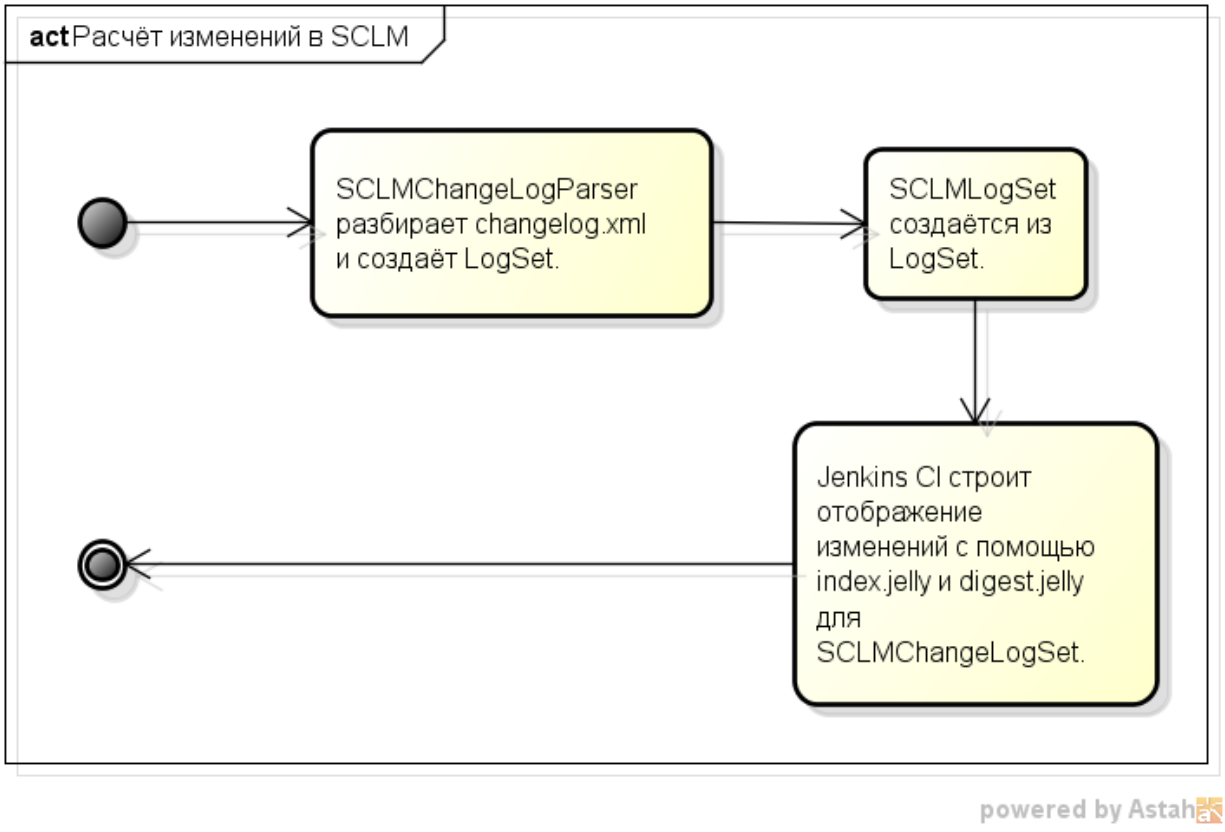

Рис. 6: Расчёт и отображение изменений в SCLM с помощью Jenkins CI.

После этого Jenkins CI запускает шаги построения проекта, настроенные пользователем. Например, это может быть шаг "Submit z/OS Job", подробнее рассмотренный в разделе 3.5.

3.5. zOSJobSubmitter

С помощью этого модуля IBM zOS Connector предоставляет пользователям возможность выполнить шаг ”Submit z/OS Job”. Этот модуль является простой оболочкой над классом zFTPConnector, который, кроме функций, перечисленных в разделе 3.3, обладает возможностью:

- Сохранять журнал работы пакетного задания в Jenkins CI.
  - Данная функция работает только если пользователь указал Jenkins CI дождаться окончания работы пакетного задания.
- Контролировать процесс построения проекта Jenkins CI в зависимости от кода завершения пакетного задания.

Процесс работы этого модуля представлен на Рис. 7.

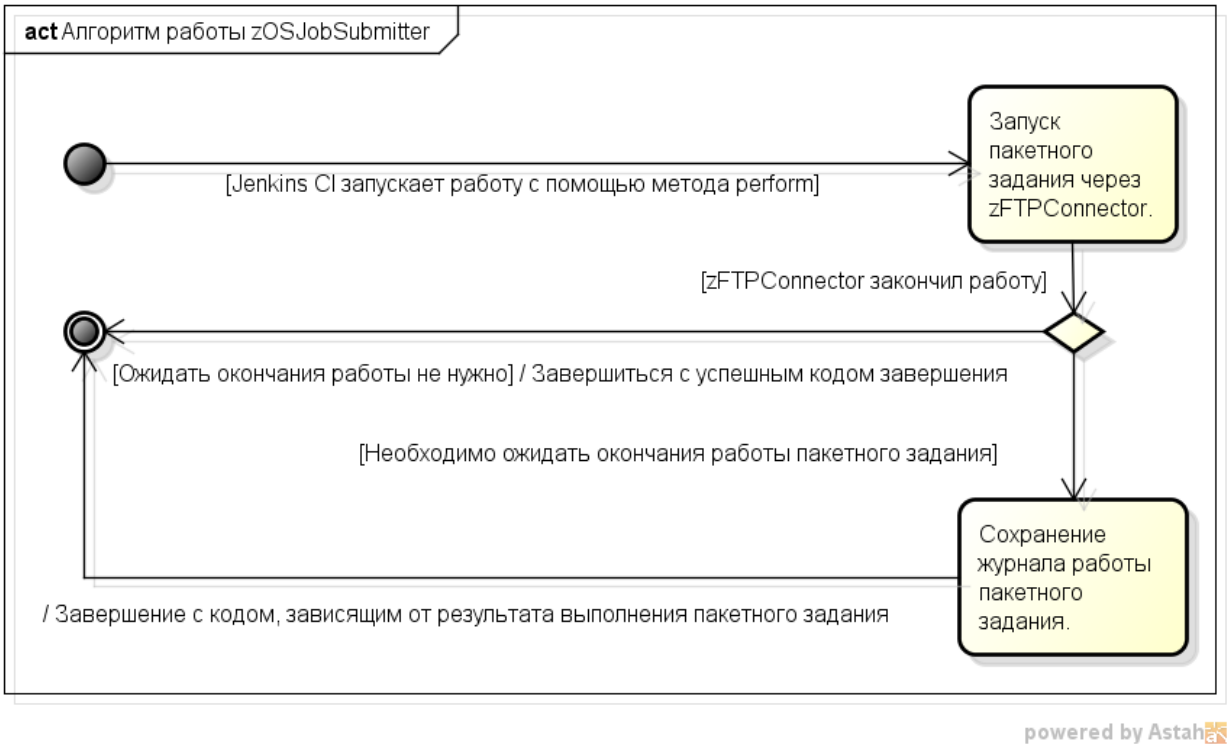

Рис. 7: Алгоритм работы zOSJobSubmitter.

4. Внедрение

Разработанное решение (а именно – IBM zOS Connector (см. раздел 3)) было внедрено[17] в корпорации EMC [5], в команде поддержки продукта Mainframe Enablers – TimeFinder/Mirror.

4.1. Процесс до внедрения непрерывной интеграции и SCLM

В этой части будут рассмотрены этапы процесса разработки ПО, использовавшегося в команде поддержки TimeFinder/Mirror до внедрения непрерывной интеграции (см. Рис. 8).

4.1.1. Особенности процесса до внедрения непрерывной интеграции

До начала использования непрерывной интеграции проверку кода на согласованность проводила команда QA[18]. Разработчики проводили только отладку собственных изменений (unit-тестирование).

Важными особенности процесса до внедрения:

- Разработка построена на двухнедельных итерациях.
- Во время итерации изменения проходили 2 этапа тестирования:
  - Ручное unit-тестирование.
  - Обязательное тестирование командой QA.
- Продвижение кода производилось 1 раз в итерацию.

4.1.2. Архитектура SCLM, используемая в процессе

Архитектура проекта SCLM, используемого в работе команды TimeFinder/Mirror, может быть представлена как последовательность

[17]Все названия внутренних необнародованных решений и модулей изменены, так как подпадают под Соглашение о неразглашении (NDA – Non-Disclosure Agreement), подписанное автором.

[18]Quality Assurance (Проверка Качества).

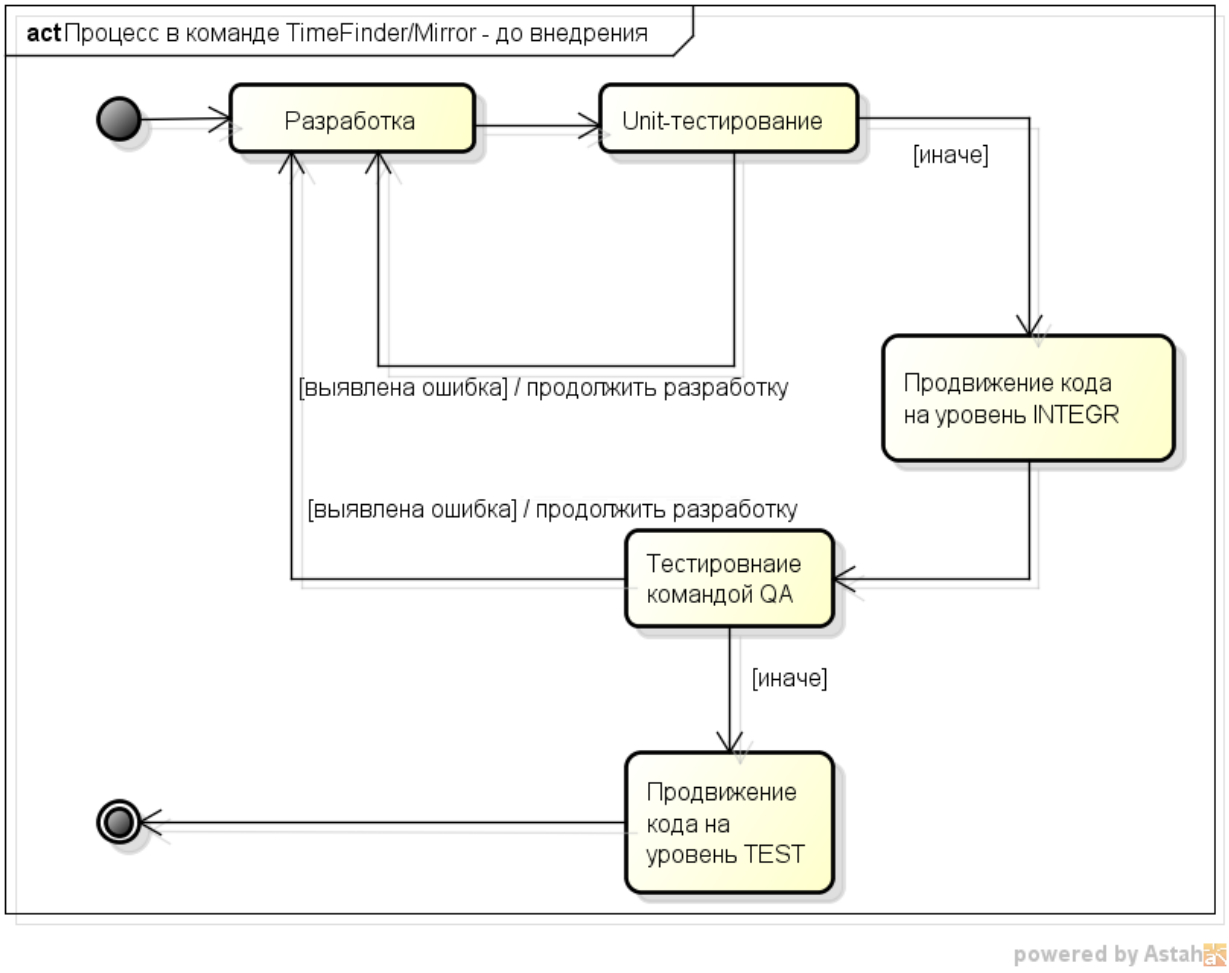

Рис. 8: Процесс разработки в команде TimeFinder/Mirror до внедрения непрерывной интеграции.

PrivDEV → DEVELOP → INTEGR → TEST → ..., в которой:

- PrivDEV – Личная группа разработчика.
- DEVELOP – Общая группа для незафиксированных изменений. Использование этого уровня обусловлено настроенной в проекте SCLM параллельной разработкой.
- INTEGR – Группа для тестирования согласованности кода (интеграционного тестирования [39]).
- TEST – Группа для дальнейшего тестирования кода (например, из неё проводится регрессионное тестирование [44]).

4.1.3. Разработка изменений

При использовании SCLM разработчику необходимо пройти несколько ISPF-панелей, чтобы начать изменение исходного кода. Во-первых, разработчик должен указать, в каком именно проекте находится необходимый исходный код. За это отвечают поля ”Project” и ”Alternate” в главном меню SCLM (см. Рис. 9). Также нужно указать, в какой группе находится исходный код.

Дело в том, что в SCLM принята древовидная структура проектов, с ”продвижением”[19] закрепляемых изменений из более низких групп в более высокие.

Далее разработчик выбирает, к какому типу относится файл исходного кода, подвергающийся изменениям. Делается это при помощи перехода Utilities→Library (можно сразу набрать ”3.1” в строке Option). (см. Рис. 10).

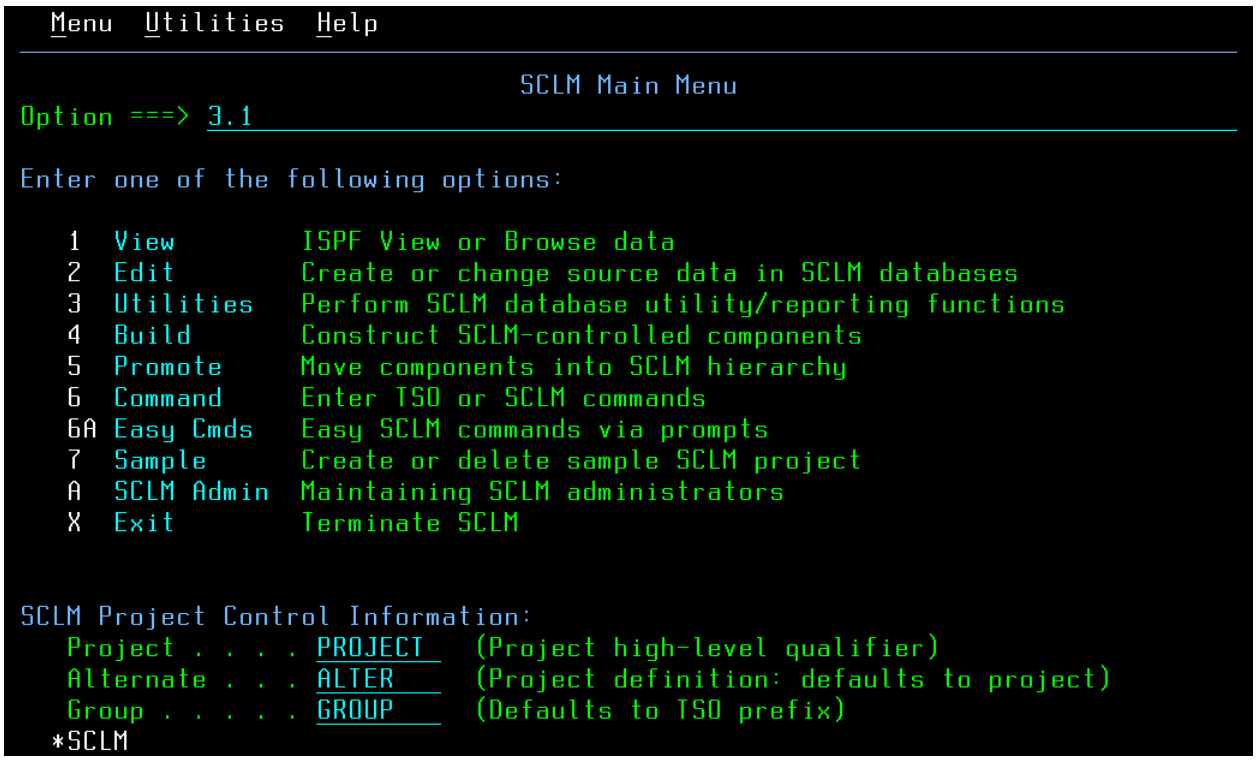

Рис. 9: Главное меню SCLM.

После этого разработчик может просмотреть все доступные для изменения файлы выбранного типа в этой группе. Для этого ему необ-

[19] Promotion.

ходимо оставить поле Member пустым. После этого он попадает на панель "Member List". На Рис. 11 показан пример такой панели. Разработчику доступны 2 файла – MEM1 и MEM2, которые находятся в группах TEST и RELEASE соответственно.

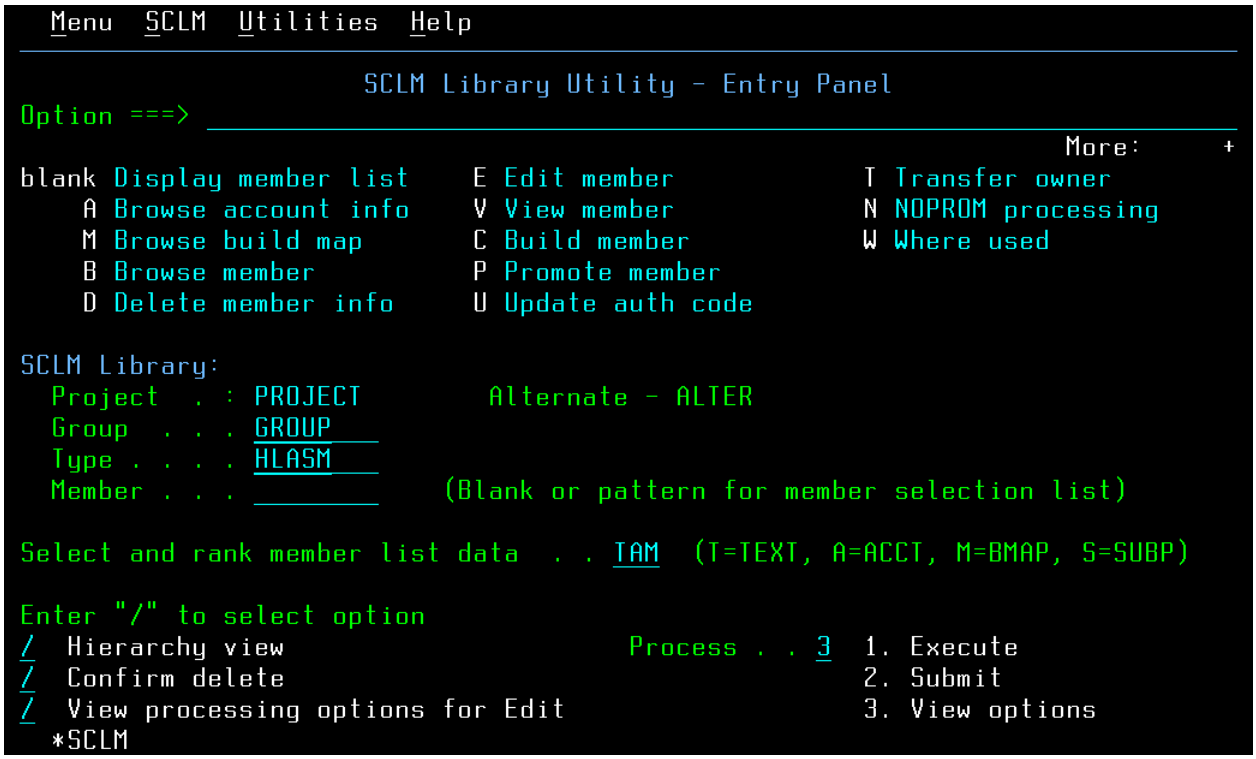

Рис. 10: SCLM – Library Utility – главная панель.

После внесения изменений разработчик отдаёт код на тестирование, например, с помощью перемещения кода на определённый уровень. Процесс перемещения будет подробнее рассмотрен в разделе 4.1.5.

Главной проблемой при завершении разработки является необходимость оповещать тестировщика о новой версии и невозможность протестировать функционал самостоятельно: ресурсы мейнфреймов дороги, поэтому разработчикам из них обычно даётся лишь необходимый минимум для разработки и unit-тестирования.

4.1.4. Тестирование изменений

В компании EMC отдел Mainframe QA[20] проводит как ручное, так и автоматизированное тестирование (используется внутренняя разработ-

[20]Mainframe Quality Assurance.

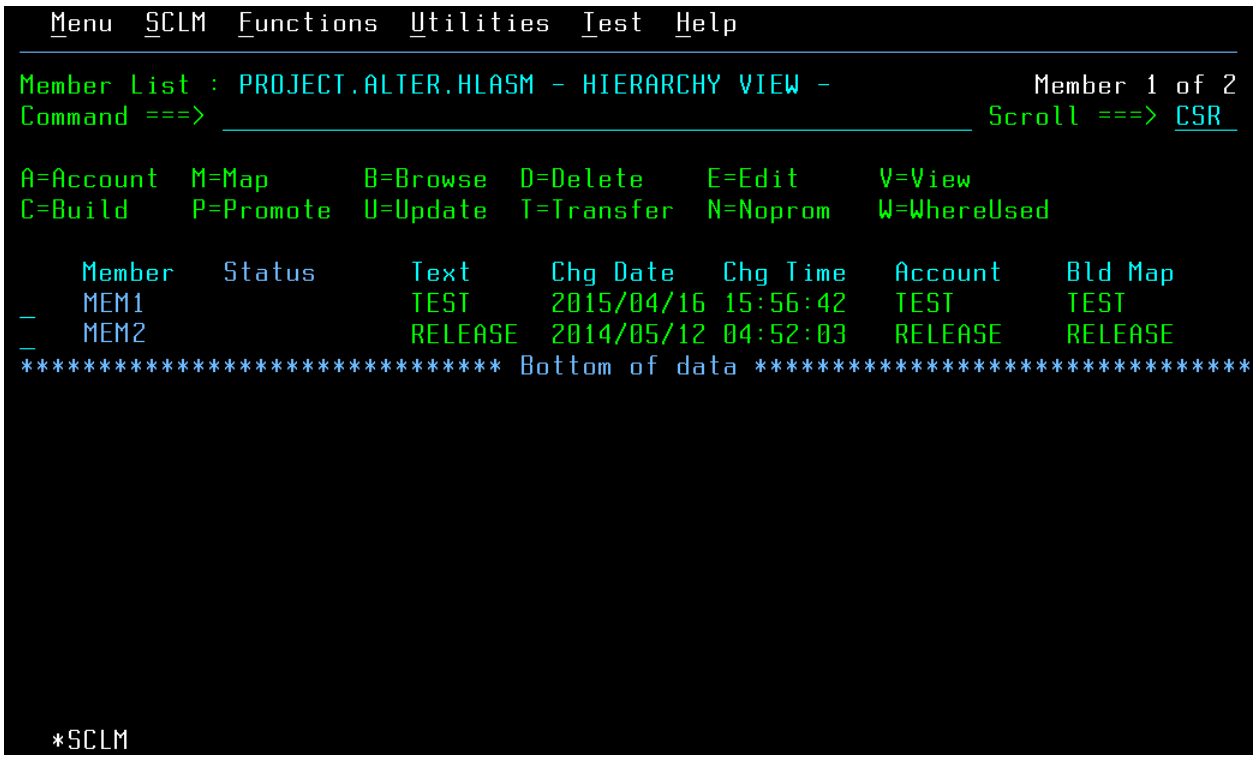

Рис. 11: SCLM – Список файлов.

ка – программа QAUTO[21], которая работает в z/OS).

После оповещения о доступности новой версии, тестировщик запускает различные сценарии, направленные на подтверждение устранения проблемы и отсутствия новых. Создаётся необходимое окружение, включающее в себя системные ресурсы и исполняемые файлы, а также производится настройка системы. После этого начинается само тестирование, во время которого производится запуск сценариев, написанных на JCL, и проверка кодов завершения этих сценариев.

Кроме ручного запуска сценариев, тестировщики используют программу QAUTO. Эта программа позволяет полуавтоматически запускать пакетные задания, а также проверять коды завершения запущенных заданий. Запуск этой программы выглядит как специальный JCL-файл со списком запускаемых пакетных заданий и рядом настроек (какой код завершения теста считать успешным, например).

На Рис. 12 показаны два теста, один из которых (TEST#OK) завершился с CC = 0, а второй (TEST#ERR) – с ошибкой (CC = 16).

[21]Скриншоты этой программы не будут приведены в данной работе из-за Соглашения о Неразглашении.

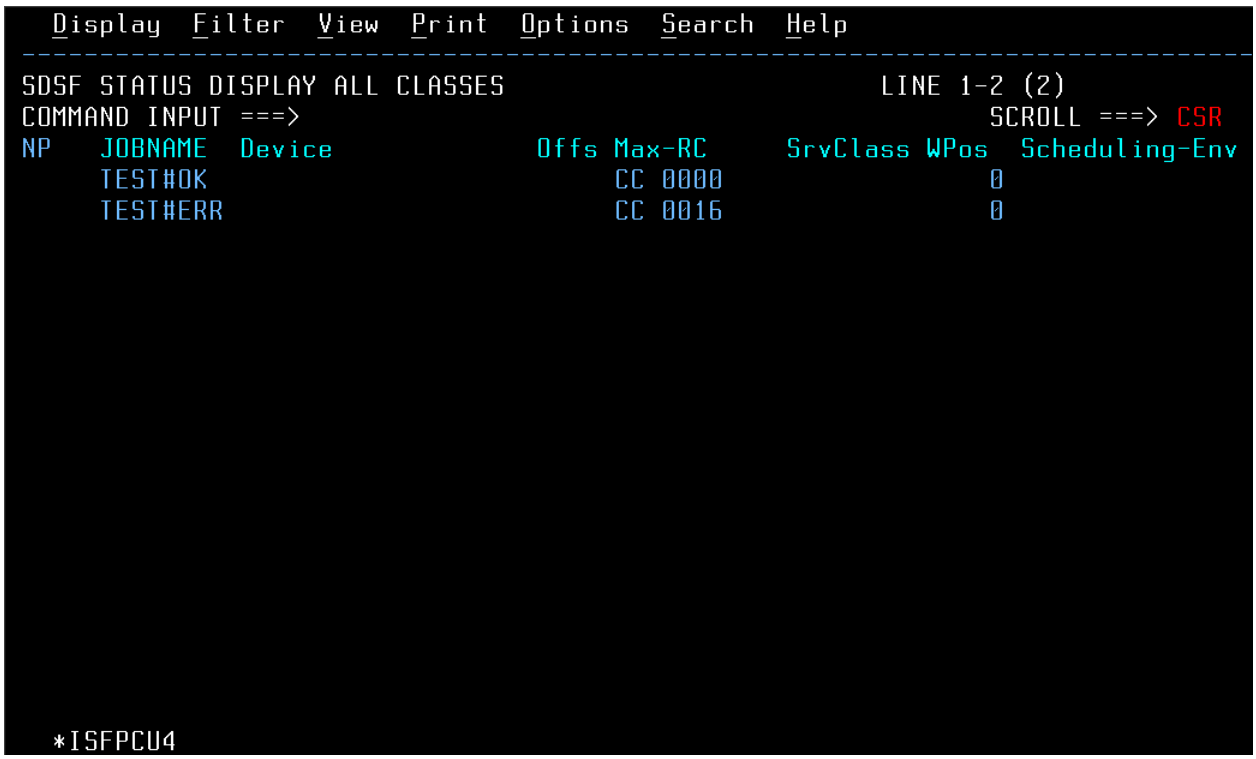

Рис. 12: Результаты тестирования на IBM z/OS – успешный (TEST#OK) и выявивший ошибки (TEST#ERR).

Зачастую, из-за нехватки ресурсов для тестирования, разработчикам приходится переделывать изменение несколько раз, тратя как своё время, так и время тестировщиков. Эти расходы удаётся сократить, проверяя основную функциональность с помощью тестирования ежедневно фиксируемой версии кода[22].

Даже если изменения оказываются удачными с первой попытки, такое тестирование всё равно занимает довольно продолжительное время, как на само тестирование, так и на коммуникации между разработчиками и тестировщиками.

4.1.5. Фиксация изменений (продвижение кода)

Для фиксации изменений с помощью SCLM программист может переместить код из более низкой группы (это может быть группа, где производится сама разработка, или же группа, откуда код попадает на

[22]Имеется ввиду практика тестирования Daily Build'а [32], которая часто применяется при использовании непрерывной интеграции.

тестирование) в более высокую.

Рассмотрим перемещение кода на примере перемещения файла MEM1 (см. раздел 4.1.3) из группы TEST в группу RELEASE. Сделать это можно тремя способами:

1. Написав [P] рядом с именем перемещаемого файла в списке файлов (см. Рис. 11).

2. Через панель Promote (см. Рис 13).

3. Выполнив команду Promote через системную консоль или через специальное пакетное задание.

После успешного перемещения файл оказывается в следующей группе, стоящей выше по иерархии. В нашем случае, файл MEM1 будет перемещён в группу RELEASE, хотя в реальных проектах обычно настраивается большее количество групп (например, иерархия групп, используемая в команде TimeFinder/Mirror, которая рассмотрена в начале раздела 4.1).

```
  Menu  SCLM  Utilities  Jobcard  Workstation Promote  Help
 ──────────────────────────────────────────────────────────────────────────────
                          SCLM Promote - Entry Panel
Command ===> ____________________________________________________________

Promote input:
  Project . . . : PROJECT        Alternate - ALTER
  From group  . . TEST____
  Type  . . . . . HLASM___              Enter "/" to select option
  Member  . . . . MEM1____              _  Workstation Promote

  Mode  . . 1  1. Conditional           Scope . . . 3  1. Normal
               2. Unconditional                        2. Subunit
               3. Report                               3. Extended

Output control:
               Ex Sub                   Process . . 2  1. Execute
  Messages . . 1  1    1. Terminal                     2. Submit
  Report . . . 1  1    2. Printer
                       3. Data set      Printer . . H
                       4. None          Volume  . . ______


  *PROMOTE
```

Рис. 13: SCLM – Панель Promote.

После любого из этих действий SCLM начинает процесс перемещения[23], во время которого выполняются следующие этапы [6]:

1. Проверка:
   - Проверяется правильность всей учётной информации[24].
   - Проверяется, все ли необходимые файлы, которые можно "построить" (скомпилировать, объединить в исполняемый файл и т.п.), были успешно построены.
2. Копирование – Производится копирование всех связанных файлов на уровень следующей более высокой группы.
3. Чистка – Все связанные файлы удаляются из текущей группы.
4. Создание отчёта.

4.2. Изменения, внесённые в процесс с Jenkins CI

В этом разделе приведено описание изменений процесса разработки ПО в команде Timefinder/Mirror, вызванные использованием непрерывной интеграции (см. Рис. 14), а также проведена оценка этих изменений.

4.2.1. Дополнительная группа в SCLM – JENKINT

Для того, чтобы наладить непрерывную интеграцию в проекте, потребовалось создать особую SCLM-группу (JENKINT), предназначенную для тестирования изменений с помощью Jenkins CI. В эту группу код попадает после того как пройдёт все unit-тесты.

Код из этой группы тестируется с помощью отдельного проекта в Jenkins CI, в котором каждую итерацию настраиваются тесты, связанные с планируемыми на эту итерацию изменениями, а также проходит

[23] Promote.
[24] Accounting information.

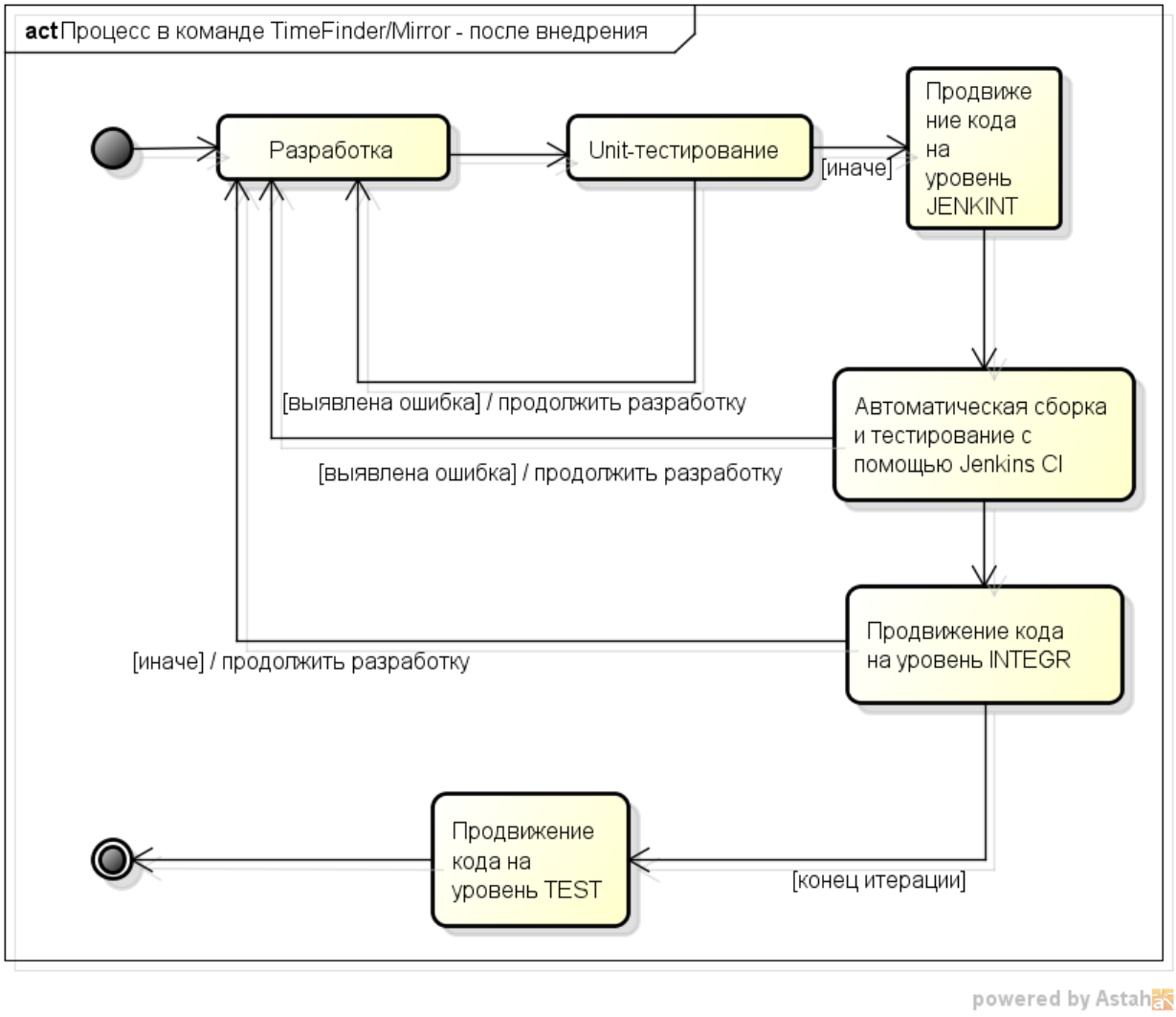

Рис. 14: Процесс разработки в команде TimeFinder/Mirror после внедрения непрерывной интеграции.

обязательные тесты, проверяющие целостность и согласованность изменений, а затем продвигается на уровень TEST. Например, если планируется вносить изменения в логику работы продукта с целостностью[25], то подбираются сценарии, проверяющие целостность данных при работе программы.

Когда все изменения, входящие в итерацию, успешно проходят свои проверки, код из группы JENKINT переходит в группу TEST для дальнейшего тестирования отделом QA.

[25]Consistency. Например, Consistent SPLIT в TimeFinder/Mirror [4].

4.2.2. TDD

Ещё одним изменением в процессе стал переход команды на TDD[26]:

1. Создание контрольных тестов – Сначала разрабатываются тестовые сценарии, которым должен удовлетворять код после изменений (например, отсутствие ошибки во время определённой операции).

2. Настройка проекта в Jenkins CI – Разработчики заводят специальный проект в Jenkins CI для unit-тестирования, в котором они настраивают отслеживание личных групп разработки и запуск созданных ранее сценариев.

3. Изменения – Только после двух предыдущих этапов начинают вноситься изменения.

4.2.3. Оптимизация ресурсов

Из-за создания группы JENKINT тестировщикам теперь не приходится проводить предварительную проверку согласованности изменений (как это было раньше) – этим занимается Jenkins CI. Вместо этого тестировщики разрабатывают новые тестовые сценарии и оптимизируют старые для запуска их с помощью нового инструмента.

Это позволило исключить этап обязательного тестирования изменений командой QA из цикла итерации, освободив их время для более глубокого тестирования из группы TEST. Также это помогло увеличить количество продвижений кода за итерацию и улучшить качество изменений, выпускаемых командой разработчиков за одну итерацию на тестирование отделу QA.

Кроме того, Jenkins CI работает не в z/OS, позволяя освободить ресурсы под другие задачи (как уже упоминалось ранее, рабочее время мейнфреймов очень дорого), что также является большим плюсом.

[26]Test Driven Development – Разработка через тестирование [43].

4.2.4. Время тестирования

Для оценки результатов внедрения также был проведён анализ времени, необходимого для тестирования изменений.

Тип тестирования	Количество тестов в наборе				
	5	10	20	35	60
Ручное тестирование	50	110	200	340	520
QAUTO	30	60	120	200	340
Jenkins CI	25	55	115	195	330
”Чистое” время работы тестов	23	51	105	180	320

Таблица 1: Сравнение времени тестирования (в минутах).

В Таблице 1 приведено время, затраченное на тестирование тем или иным методом (включая все дополнительные расходы на настройку окружения, запуск тестирования и проверку результатов), а также ”чистое” время тестирования – время работы самих тестов без дополнительных затрат. Для большей наглядности приведено сравнение времени тестирования в виде гистограмм на Рис. 15.

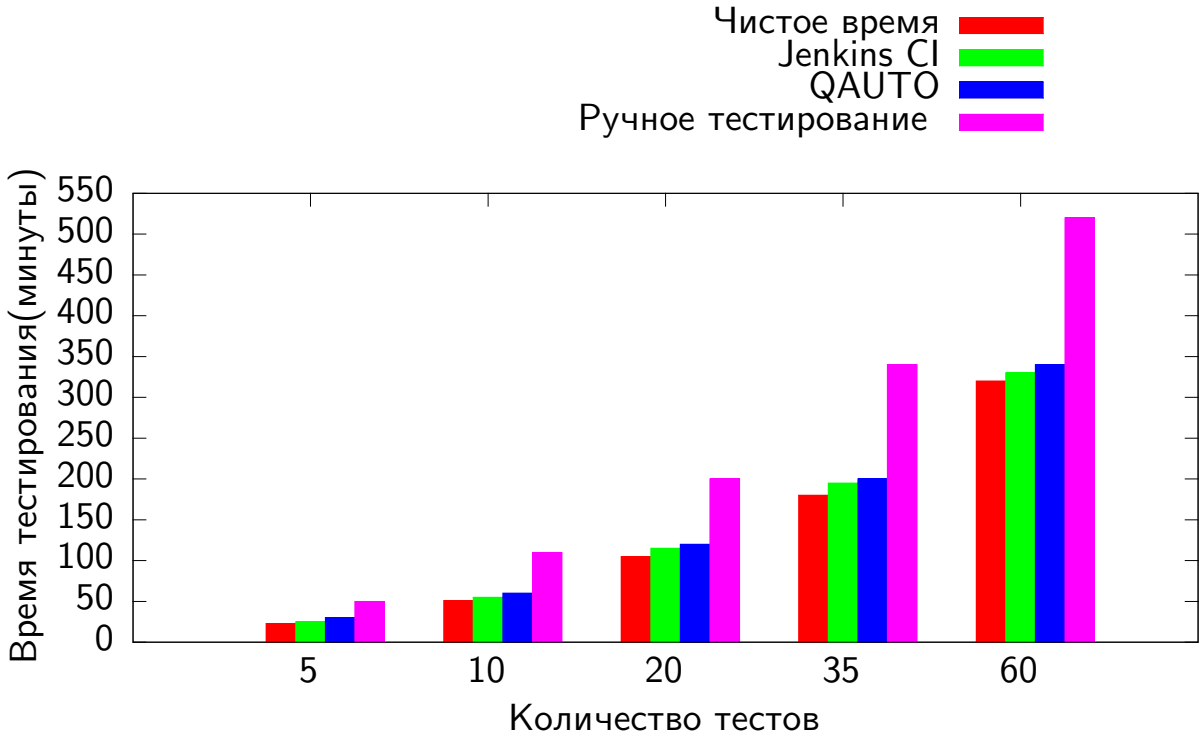

Рис. 15: Сравнение времени тестирования.

Все значения указывают только время тестирования. Для Jenkins CI также производился замер времени, необходимый для построения SCLM-проекта, однако, по сравнению со временем тестирования, это время оказалось незначительным (SCLM-проект TimeFinder/Mirror проходил полную сборку примерно за 2-3 минуты).

Как видно из Таблицы 1, использование Jenkins CI позволило получить небольшой прирост в скорости тестирования и отделу тестирования, улучшив результат внутренней разработки компании EMC – программы QAUTO.

Заключение

В результате проведённой работы были созданы два инструмента автоматизации, с помощью которых можно настроить непрерывную интеграцию при разработке ПО для мейнфреймов с использованием SCLM:

- IBM zOS Connector:
  - Расширение для Jenkins CI, позволяющее использовать SCLM в качестве системы контроля версий, а также запускать пакетные задания на IBM z/OS со сбором журнала выполнения и сохранением его в Jenkins CI.
  - Расширение описано в Jenkins CI Wiki [24].
  - Исходный код расширения доступен на GitHub [47].
- Serv:
  - Планировщик для регулярного запуска пакетных заданий, написанный на C и Rexx.
  - Исходный код расширения доступен на GitHub [46].

IBM zOS Connector был внедрён в реальный процесс разработки ПО для мейнфреймов в корпорации EMC (см. раздел 4). Было проведено перестроение процесса разработки, направленное на использование непрерывной интеграции.

Анализ перестроенного процесса и сравнение его с принятым до этого показал, что использование непрерывной интеграции и, в частности, IBM zOS Connector, позволило добиться уменьшения времени на тестирование изменений разработчиками в 1,5-2 раза[27].

[27] Среднее время unit-тестирования улучшилось в 1,8 раз (см. раздел 4.2).

Список литературы

[1] Apache. Apache Commons Net – Overview // Apache Commons. — 2015. — URL: https://commons.apache.org/proper/commons-net (дата обращения: 30.05.2015).

[2] Apache. Jelly – Executable XML // Apache Commons. — 2015. — URL: http://commons.apache.org/proper/commons-jell.

[3] BCM. MainView Automation // BMC. — 2015. — URL: http://www.bmc.com/it-solutions/mainview-mainframe-automation.html (дата обращения: 03.05.2015).

[4] EMC. EMC Mainframe Enablers TimeFinder/Mirror for z/OS – Version 7.6 – Product Guide / EMC. — 2013.

[5] EMC. EMC – Leading Cloud Computing, Big Data, and Trusted IT Solutions // EMC. — 2015. — URL: http://www.emc.com/index.htm (дата обращения: 30.05.2015).

[6] Ford Rene. — SCLM Education Manual. — IBM Global Services, 2004.

[7] GitLab. GitLab Continuous Integration // GitLab. — 2015. — URL: https://about.gitlab.com/gitlab-ci (дата обращения: 03.05.2015).

[8] Habrahabr. IBM представила новый мейнфрейм z13 // Habrahabr. — 2015. — URL: http://habrahabr.ru/company/ibm/blog/248833 (дата обращения: 02.05.2015).

[9] IBM. z/OS TSO/E – REXX User's Guide / IBM Corporation. No. SA22-7791-01. — 2001.

[10] IBM. z/OS Communications Server – IP User's Guide and Commands / IBM Corporation. No. SC31-8780-09. — Tenth edition. — 2003.

[11] IBM. z/OS MVS – JCL Reference / IBM Corporation. No. SA22-7597-16. — 2011.

[12] IBM. z/OS MVS – JCL User's Guide / IBM Corporation. No. SA22-7598-07. — 2011.

[13] IBM. z/OS TSO/E – REXX Reference / IBM Corporation. No. SA22-7790-10. — 2011.

[14] IBM. z/OS MVS – JCL Reference / IBM Corporation. No. SA22-7627-28. — 2012.

[15] IBM. ISPF for z/OS – Software Configuration and Library Manager Guide and Reference / IBM Corporation. No. SC19-3625-00. — First edition. — 2013.

[16] IBM. ISPF for z/OS – User's Guide Volume I / IBM Corporation. No. SC19-3627-00. — First edition. — 2013.

[17] IBM. ISPF for z/OS – User's Guide Volume II / IBM Corporation. No. SC19-3628-00. — First edition. — 2013.

[18] IBM. z/OS JES2 – Introduction / IBM Corporation. No. SA32-0994-00. — 2013.

[19] IBM. z/OS JES3 – Introduction / IBM Corporation. No. SA32-1004-00. — 2013.

[20] IBM. z/OS UNIX System Services – User's Guide / IBM Corporation. No. SA23-2279-00. — 2013.

[21] IBM. IBM Tivoli NetView for z/OS // IBM. — 2015. — URL: `http://www-03.ibm.com/software/products/ru/tivoli-netview-zos` (дата обращения: 03.05.2015).

[22] IBM. Rational Developer for System z // IBM. — 2015. — URL: `http://www-03.ibm.com/software/products/en/developerforsystemz` (дата обращения: 03.05.2015).

[23] IBM. SCLM Developer Toolkit // IBM. — 2015. — URL: http://www-03.ibm.com/software/products/ru/devtoolkit (дата обращения: 03.05.2015).

[24] Jenkins CI. Jenkins Wiki // Jenkins CI Wiki. — 2015. — URL: https://wiki.jenkins-ci.org (дата обращения: 30.05.2015).

[25] Jenkins CI. Welcome to Jenkins CI! // Jenkins CI. — 2015. — URL: https://jenkins-ci.org (дата обращения: 03.05.2015).

[26] Jenkins CI. Writing an SCM plugin // Jenkins CI Wiki. — 2015. — URL: https://wiki.jenkins-ci.org/display/JENKINS/Writing+an+SCM+plugin (дата обращения: 30.05.2015).

[27] SDS. VitalSigns for Network Automation and Control | VNAC | Overview // SDS. — 2015. — URL: http://www.sdsusa.com/vnac (дата обращения: 03.05.2015).

[28] SEA. ODDS – Console Automation // Software Engineering Of America. — 2014. — URL: http://www.seasoft.com/console-automation.asp (дата обращения: 03.05.2015).

[29] StickyMinds. Make Your Mainframe Systems and Technology More Agile: An Interview with Jay McFarling and Danielle Roecker // StickyMinds. — 2015. — URL: http://goo.gl/dKe2b5 (дата обращения: 03.05.2015).

[30] ThoughtWorks Inc. Go Continuous Delivery // ThoughtWorks Inc. — 2015. — URL: http://www.go.cd (дата обращения: 03.05.2015).

[31] Wikipedia. Comparison of continuous integration software // Википедия, свободная энциклопедия. — URL: http://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software (дата обращения: 06.05.2015).

[32] Wikipedia. Daily build // Википедия, свободная энциклопедия. — URL: http://en.wikipedia.org/wiki/Daily_build (дата обращения: 06.05.2015).

[33] Wikipedia. IBM System/360 // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/IBM_System/360 (дата обращения: 02.05.2015).

[34] Wikipedia. LPAR // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/LPAR (дата обращения: 30.05.2015).

[35] Wikipedia. Legacy code // Википедия, свободная энциклопедия. — URL: http://en.wikipedia.org/wiki/Legacy_code (дата обращения: 02.05.2015).

[36] Wikipedia. Scrum // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Scrum (дата обращения: 02.05.2015).

[37] Wikipedia. cron // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Cron (дата обращения: 30.05.2015).

[38] Wikipedia. Гибкая методология разработки // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Гибкая_методология_разработки (дата обращения: 30.04.2015).

[39] Wikipedia. Интеграционное тестирование // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Интеграционное_тестирование (дата обращения: 30.05.2015).

[40] Wikipedia. Конфигурационное управление // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Конфигурационное_управление (дата обращения: 30.04.2015).

[41] Wikipedia. Мейнфрейм // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Мейнфрейм (дата обращения: 02.05.2015).

[42] Wikipedia. Непрерывная интеграция // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Непрерывная_интеграция (дата обращения: 30.04.2015).

[43] Wikipedia. Разработка через тестирование // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Разработка_через_тестирование (дата обращения: 03.05.2015).

[44] Wikipedia. Регрессионное тестирование // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Регрессионное_тестирование (дата обращения: 30.05.2015).

[45] Wikipedia. Система управления версиями // Википедия, свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/Система_управления_версиями (дата обращения: 30.04.2015).

[46] Щербаков Александр. Serv // GitHub. — 2015. — URL: https://github.com/candiduslynx/serv (дата обращения: 30.05.2015).

[47] Щербаков Александр. Zos-connector-plugin // GitHub. — 2015. — URL: https://github.com/jenkinsci/zos-connector-plugin (дата обращения: 30.05.2015).

Приложение

В Приложении будут приведены различные конфигурационные файлы и сценарии запусков, упомянутые в данной работе. В Приложении не будет приведён исходный код инструментов, описанных в разделах 2 и 3, так как они доступны в сети интернет по следующим ссылкам:

- Serv – https://github.com/candiduslynx/serv
- IBM zOS Connector – https://github.com/jenkinsci/zos-connector-plugin

6.1. Serv

6.1.1. Пакетное задание для запуска Serv

```
//R$SERV$M JOB (EMC),'RUN SERVMAIN - C',CLASS=A,MSGCLASS=X,
//             NOTIFY=&SYSUID,REGION=0M,MSGLEVEL=(1,1)
//* AUTHORIZE LIBRARIES
// SETPROG APF,ADD,DSN=USRHLQ.LINKLIB,SMS
//* START SERVMAIN
//SERVSTRT EXEC PGM=IKJEFT01,PARM='SERVMAIN USRHLQ.TEST.JCL'
//STEPLIB  DD   DISP=SHR,DSN=USRHLQ.LINKLIB
//SYSPRINT DD   SYSOUT=*
//SERVLOG  DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
//SYSIN    DD   DISP=SHR,DSN=USRHLQ.PARMLIB(RUN1)
```

6.1.2. Файл настроек

```
# JOBNAME   DELAY        INTERVAL
# JOBNAME   DDD:HH:MM    DDD:HH:MM
  WAIT5     000:00:00    000:00:13
  WAIT10    000:00:01    000:00:22
  WAIT33    000:00:02    000:00:41
  WAIT3     000:00:00    000:00:57
```

6.2. IBM zOS Connector

6.2.1. Пакетное задание для работы с SCLM

```
//SCLMEX   EXEC PGM=IKJEFT01,
//            REGION=4096K,TIME=1439,DYNAMNBR=200
//STEPLIB  DD DSN=ISP.SISPLPA,DISP=SHR
//         DD DSN=ISP.SISPLOAD,DISP=SHR
//ISPMLIB  DD DSN=ISP.SISPMENU,DISP=SHR
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//         DD DSN=ISP.SISPSLIB,DISP=SHR
//ISPPLIB  DD DSN=ISP.SISPPENU,DISP=SHR
//ISPTLIB  DD UNIT=@TEMP0,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//            DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//            DSN=
//         DD DSN=ISP.SISPTENU,DISP=SHR
//ISPTABL  DD UNIT=@TEMP0,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//            DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//            DSN=
//ISPPROF  DD UNIT=@TEMP0,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//            DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//            DSN=
//ISPLOG   DD SYSOUT=*,
//            DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//ISPCTL1  DD DISP=NEW,UNIT=@TEMP0,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSTERM  DD SYSOUT=*
//SYSPROC  DD DSN=ISP.SISPCLIB,DISP=SHR
//FLMMSGS  DD SYSOUT=(*)
//PASCERR  DD SYSOUT=(*)
//ZFLMDD   DD  *
   ZFLMNLST=FLMNLENU    ZFLMTRMT=ISR3278    ZDATEF=YY/MM/DD
/*
//SYSPRINT DD SYSOUT=(*)
//SYSTSPRT DD SYSOUT=(*)
//SYSTSIN  DD *
  ISPSTART CMD(FLMCMD FILE,DBUWORK)
/*
//MSGS     DD SYSOUT=*
//REPT     DD SYSOUT=*
//TAIL     DD SYSOUT=*
//DBUWORK  DD *
DBUTIL,PROJECT,ALTER,GROUP,,,,,,
+*,*,*,*,*,*,*,*,YES,*,*,,,,NORMAL,N,N,,MSGS,REPT,TAIL,
+@@FLMCLV.@@FLMTYP(@@FLMMBR) <@@FLMCD4 @@FLMCTM> @@FLMCUS @@FLMMVR
/*
```

6.2.2. Настройка SCLM в качестве SCM

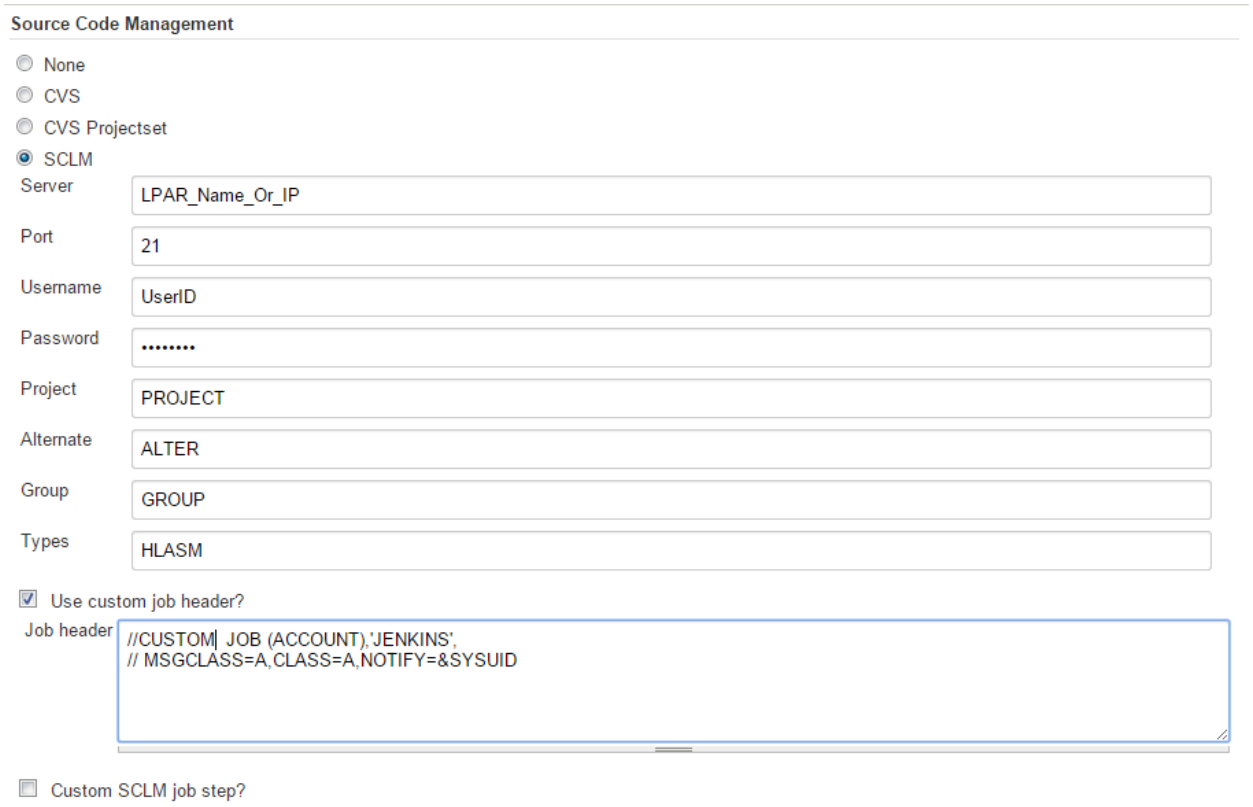

Рис. 16: Пример настройки SCLM в качестве SCM в Jenkins CI.

6.2.3. Настройка пакетного задания

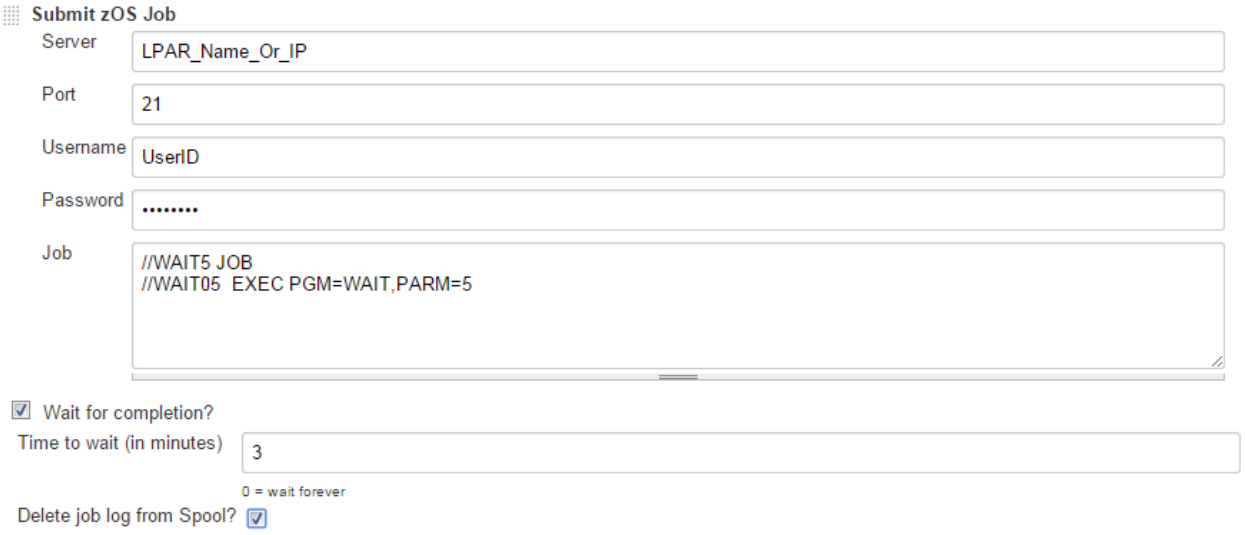

Рис. 17: пример настройки пакетного задания в Jenkins CI.