Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ЭВМ и С

**ОТЧЁТ**

**ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ**

База практики ИООО «ЭПАМ Системз»

Сроки прохождения практики 02.02.2015 - 27.03.2015

Руководитель практики от производства ведущий инженер-программист Оргиш А. Н.

Руководитель практики от университета старший преподаватель Четвёркина Г. А.

Выполнил:

студент 5-го курса ФЭИС

группы Э-44

Самсонов Д. С.

Брест 2015

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc415760257)

[1 ХАРАКТЕРИСТИКА И СТРУКТУРА ПРЕДПРИЯТИЯ 4](#_Toc415760258)

[1.1 Характеристика предприятия 4](#_Toc415760259)

[1.2 Административная структура 6](#_Toc415760260)

[1.3 Должностные инструкции работников подразделения 8](#_Toc415760261)

[1.4 Обзор задач, решаемых подразделением 12](#_Toc415760262)

[1.5 Обзор используемых информационных технологий, средств ИТ 14](#_Toc415760263)

[2. Системы контроля версий (СКВ) на предприятии. 15](#_Toc415760264)

[2.1 Общее описание СКВ, использующихся на предприятии. Обзор децентрализованных и централизованных систем контроля версий. 15](#_Toc415760265)

[2.2 Типовые операции и порядок работы с системой контроля версий. 18](#_Toc415760266)

[2.3 Системы контроля версий, использующиеся на предприятии. 20](#_Toc415760267)

[2.4 Разработка концепции лабораторного практикума. 23](#_Toc415760268)

[2.5 Рекомендации по улучшению работы с системами контроля версий на предприятии. 24](#_Toc415760269)

[ЗАКЛЮЧЕНИЕ 28](#_Toc415760270)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc415760271)

# **ВВЕДЕНИЕ**

Целью практики является приобретение практического опыта, навыков самостоятельной работы необходимых в дальнейшей профессиональной деятельности, а также закрепление теоретических знаний, полученных на предыдущих курсах. Кроме этого, в задачи преддипломной практики входит углубленное изучение предметной области по теме дипломной работы, сбор материалов, их анализ и подготовку к написанию работы.

Соответственно основные задачи практики:

1. изучение организационной структуры предприятия и подразделения;
2. изучение должностных инструкций работников подразделения;
3. обзор задач, решаемых подразделением;
4. обзор используемых информационных технологий, средств ИТ.

Индивидуальное задание:

1. обзор систем автоматического распознавания форм;
2. разработка структурной схемы модуля проектирования форм;
3. разработка общего алгоритма сегментации изображений текстовых бланков.

# **1 ХАРАКТЕРИСТИКА И СТРУКТУРА ПРЕДПРИЯТИЯ**

# **ХАРАКТЕРИСТИКА ПРЕДПРИЯТИЯ**

EPAM Systems — американская ИТ-компания основанная в 1993 году, производитель заказного программного обеспечения, специалист по консалтингу, резидент Белорусского парка высоких технологий. Штаб-квартира компании расположена в Ньютауне (США, штат Пенсильвания).

EPAM Systems неизменно остается признанным лидером в таких областях, как:

* разработка, тестирование, сопровождение и поддержка заказного программного обеспечения и бизнес-приложений;
* интеграция приложений на базе продуктов SAP, Oracle, IBM, Microsoft;
* создание выделенных центров разработки (центров компетенции), центров тестирования и контроля качества программного обеспечения;
* ИТ-консалтинг с учетом отраслевой специфики бизнеса.

EPAM Systems представлена офисами в различных странах мира, таких как: Россия, Беларусь, Украина, Казахстан, Венгрия, Польша, Великобритания, Германия, Швейцария, Швеция, США

В число клиентов EPAM Systems входят такие крупные компании, как: Oracle, Microsoft, Coca-Cola, Reuters, Barclays Capital, British Telecom, Etas, Parallels, Colgate-Palmolive, Wolters Kluwer, Expedia, Viacom, Мозырский НПЗ, БМЗ, «Газпром нефть», «РосНефть», «Росэнергоатом» и др.

В 2006 году компания вошла в список «100 мировых аутсорсеров», #79 в разделе «Восходящие звезды аутсорсинга на глобальной арене» (Global Outsourcing Rising Stars).

По данным рейтингового агентства «Эксперт РА», по итогам 2009 года EPAM Systems занимает 19-е место в общем списке крупнейших ИТ-компаний России.

По данным журнала cNews, компания занимает первое место среди крупнейших ИТ-аутсорсеров и 29-е среди крупнейших ИТ-компаний России в 2008 году.

В 2014 году EPAM названа лидером в отчете «The Forrester Wave™: Software Product Development Services, Q1 2014». По мнению Forrester, «EPAM опередила всех других поставщиков по продвижению инноваций и содействию в создании новых инновационных продуктов».

В 2013 году компания вошла в списки [Forbes](https://ru.wikipedia.org/wiki/Forbes" \o "Forbes) «25 самых быстрорастущих технологических компаний Америки» и «20 самых быстрорастущих технологических звезд Америки»

По данным [«Коммерсанта»](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D1%80%D1%81%D0%B0%D0%BD%D1%82%D1%8A), по итогам 2013 года EPAM Systems занимает 10-е место в общем списке крупнейших ИТ-компаний России, 1-е место в списке разработчиков программного обеспечения, а также 7-е место в рейтинге крупнейших консультационных компаний России.

По данным рейтингового агентства «Эксперт РА» по итогам 2013 года компания возглавляет список ведущих разработчиков программного обеспечения, занимает 8-е место в рейтинге «Российский консалтинг» и 10-е место в рейтинге крупнейших российских компаний в области [ИКТ](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B8).

Общая численность персонала EPAM Systems — свыше 10 тыс. человек.

Выручка компании за 2013 год составила $555,1 млн долл.

# **АДМИНИСТРАТИВНАЯ СТРУКТУРА**

Административная структура компании EPAM Systems представлена на рис.1.1.

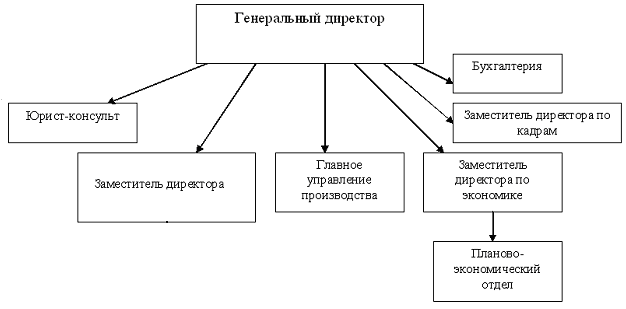


Рисунок 1.1 – Административная структура компании EPAM Systems

Во главе предприятия находится директор. Он организует всю работу предприятия и несет полную ответственность за результат производственно-хозяйственной деятельности EPAM Systems. Директор представляет предприятие во всех учреждениях и организациях, заключает договора, издает приказы по предприятию, открывает в банках счета предприятия и выполняет целый ряд других функций.

В непосредственном подчинении директора предприятия находятся три заместителя: по маркетингу, по экономике и по кадрам, а также главный бухгалтер и юрист-консульт.

Заместитель директора по кадрам возглавляет работу по обеспечению предприятия кадрами рабочих, служащих и инженерно-технических работников требуемых профессий в соответствии с уровнем и профилем полученной подготовки, заботится о создании положительного мнения общественности о предприятии. Он также подготавливает документы для начисления пенсий, осуществляет контроль за правильностью оформления документов по начислению зарплаты.

Заместитель директора по маркетингу и перспективному планированиюявляется первым заместителем директора предприятия, в обязанности которого входит целый ряд наиболее важных и ответственных функций. Первый заместитель директора предприятия организует и руководит деятельностью службы, занятой проведением комплекса маркетинговых мероприятий, направленных на повышение конкурентного уровня качества выпускаемой продукции; рекламирование продукции и услуг предприятия. Он участвует в принятии решений по претензиям потребителей продукции и услуг; руководит мероприятиями, направленными на сбыт выпускаемой продукции.

Заместитель директора по экономике руководит и координирует работу планово-экономического отдела. Планово-экономический отдел подготавливает необходимые материалы для рассмотрения проектов планов, итогов деятельности предприятия, организует работу по нормированию и ценообразованию на предприятии, проводит экономический анализ производственно-хозяйственной деятельности.

Юрист-консульт подчиняется непосредственно директору предприятия и осуществляет проверку соответствия действующему законодательству приказов, распоряжений, соглашений и других актов правового характера, участие в заключении коллективных договоров и т.д.

Общая характеристика предприятия EPAM Systems показала, что на современном этапе развития данная организация является одним из лидеров по разработкам программного обеспечения. Показатели результативной деятельности предприятия являются удовлетворительными и свидетельствуют о большем потенциале организации.

# **ДОЛЖНОСТНЫЕ ИНСТРУКЦИИ РАБОТНИКОВ ПОДРАЗДЕЛЕНИЯ**

**I. Общие положения**

1. Инженер-программист относится к категории специалистов.

2. На должность:

- инженера-программиста назначается лицо, имеющее высшее профессиональное (техническое или инженерно-экономическое) образование без предъявления требований к стажу работы или среднее профессиональное (техническое или инженерно-экономическое) образование и стаж работы в должности техника I категории не менее 3 лет либо других должностях, замещаемых специалистами со средним профессиональным образованием, не менее 5 лет;

- инженера-программиста III категории - лицо, имеющее высшее профессиональное (техническое или инженерно-экономическое) образование и опыт работы по специальности, приобретенный в период обучения, или стаж работы на инженерно-технических должностях без квалификационной категории;

- инженера-программиста II категории - лицо, имеющее высшее профессиональное (техническое или инженерно-экономическое) образование и стаж работы в должности инженера-программиста III категории или других инженерно-технических должностях, замещаемых специалистами с высшим профессиональным образованием, не менее 3 лет;

- инженера-программиста I категории - лицо, имеющее высшее профессиональное (техническое или инженерно-экономическое) образование и стаж работы в должности инженера-программиста II категории не менее 3 лет.

3. Инженер-программист должен знать:

3.1. Руководящие и нормативные материалы, регламентирующие методы разработки алгоритмов и программ и использования вычислительной техники при обработке информации.

3.2. Основные принципы структурного программирования.

3.3. Виды программного обеспечения.

3.4. Технико-эксплуатационные характеристики, конструктивны особенности, назначение и режимы работы ЭВМ, правила ее технической эксплуатации.

3.5. Технологию автоматической обработки информации и кодирования информации.

3.6. Формализованные языки программирования

3.7. Действующие стандарты, системы счислений, шифров и кодов.

3.8. Порядок оформления технической документации.

3.9. Передовой отечественный и зарубежный опыт программирования и использования вычислительной техники.

3.10. Основы экономики, организации производства, труда и управления.

3.11. Основы трудового законодательства.

3.12. Правила внутреннего трудового распорядка.

3.13. Правила и нормы охраны труда.

4. На время отсутствия инженера-программиста (отпуск, болезнь, пр.) его обязанности исполняет лицо, назначенное в установленном порядке. Данное лицо приобретает соответствующие права и несет ответственность за качественное и своевременное исполнение возложенных на него обязанностей.

**II. Должностные обязанности**

Инженер-программист:

1. На основе анализа математических моделей и алгоритмов решения экономических и других задач разрабатывает программы, обеспечивающие возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники, проводит их тестирование и отладку.

2. Разрабатывает технологию решения задачи по всем этапам обработки информации.

3. Осуществляет выбор языка программирования для описания алгоритмов и структур данных.

4. Определяет информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля.

5. Выполняет работу по подготовке программ к отладке и проводит отладку.

6. Определяет объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению.

7. Осуществляет запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач.

8. Проводит корректировку разработанной программы на основе анализа выходных данных.

9. Разрабатывает инструкции по работе с программами, оформляет необходимую техническую документацию.

10. Определяет возможность использования готовых программных продуктов.

11. Осуществляет сопровождение внедрения программ и программных средств.

12. Разрабатывает и внедряет системы автоматической проверки правильности программ, типовые и стандартные программные средства, составляет технологию обработки информации.

13. Выполняет работу по унификации и типизации вычислительных процессов.

14. Принимает участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектировании программ, позволяющих расширить область применения вычислительной техники.

**III. Права**

Инженер-программист имеет право:

1. Знакомиться с проектами решений руководства предприятия, касающихся его деятельности.

2. Вносить на рассмотрение руководства предложения по совершенствованию работы, связанной с предусмотренными настоящей инструкцией обязанностями.

3. В пределах своей компетенции сообщать своему непосредственному руководителю о всех выявленных в процессе осуществления должностных обязанностей недостатках в деятельности предприятия (его структурных подразделениях) и вносить предложения по их устранению.

4. Запрашивать лично или по поручению своего непосредственного руководителя от специалистов подразделений информацию и документы, необходимые для выполнения его должностных обязанностей.

5. Привлекать специалистов всех (отдельных) структурных подразделений к решению задач, возложенных на него (если это предусмотрено положениями о структурных подразделениях, если нет - то с разрешения их руководителей).

6. Требовать от своего непосредственного руководителя, руководства предприятия оказания содействия в исполнении им своих должностных обязанностей и прав.

**IV. Ответственность**

Инженер-программист несет ответственность:

1. За ненадлежащее исполнение или неисполнение своих должностных обязанностей, предусмотренных настоящей должностной инструкцией, - в пределах, определенных действующим трудовым законодательством Республики Беларусь.

2. За правонарушения, совершенные в процессе осуществления своей деятельности, - в пределах, определенных действующим административным, уголовным и гражданским законодательством Республики Беларусь.

3. За причинение материального ущерба - в пределах, определенных действующим трудовым и гражданским законодательством Республики Беларусь.

# 

# **ОБЗОР ЗАДАЧ, РЕШАЕМЫХ ПОДРАЗДЕЛЕНИЕМ**

Отдел разработки программного обеспечения выполняет одну из самых важных и ответственных задач в компании - производство программных продуктов, которые в дальнейшем организация будет поставлять, внедрять, сопровождать и обслуживать.

Работа отдела разработки программного обеспечения строится по типичным схемам разработки продуктов:

* определение целей разработки;
* написание и утверждение Технического Задания;
* кодирование и отладка системы;
* тестирование продукта;
* подготовка описаний и руководств для конечных пользователей.

Хотя внешне схема может показаться довольно простой, на самом деле она имеет много подводных камней. Здесь так же, как и в любой иной области производства, есть свои трудности. Для каждого проекта нужно найти и подготовить квалифицированных специалистов. Ведь каким бы грамотным программист не был, он должен отлично разбираться не только в платформе, но и знать предметную область проекта.

В отделе разработки программного обеспечения создаются небольшие персональные приложения и программы, а также информационные системы объемного масштаба. В этом направлении предлагается:

* разработка информационных систем;
* создание программного обеспечения под заказ;
* сопровождение продуктов.

Разработка программных продуктов - это многоэтапный проект, который кроме непосредственно разработки включает в себя обследование, аналитическую и управленческую деятельность, проектирование, внедрение проекта и его сопровождение.

В своей работе любой отдел разработки программного обеспечения стремится к выполнению своей работы, основываясь на следующих принципах:

* Анализ предметной области;
* Составление модели будущего программного продукта;
* Использование новых технологий разработки программного обеспечения и СУБД;
* Привлечение высококвалифицированной команды программистов и менеджеров;
* Использование лицензионного программного обеспечения сред разработок;
* Глубокое тестирование готового программного обеспечения;
* Сопровождение и поддержка на стадии внедрения и эксплуатации разработанного программного продукта.

# **ОБЗОР ИСПОЛЬЗУЕМЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, СРЕДСТВ ИТ**

Средства информационных технологий являются неотъемлемой и наиболее существенной их составляющей. Они призваны выполнять ту же роль, что и средства производства в процессе трудовой деятельности.

Средства информационных технологий - это совокупность средств деятельности человека, которые создаются и используются с целью осуществления процессов производства и удовлетворения непроизводственных потребностей общества. К средствам информационных технологий относят разнообразную компьютерную технику, которая облегчает и повышает уровень эффективности трудовых усилий человека, расширяет его возможности в процессе трудовой деятельности и освобождает человека частично или полностью от работы.

В состав средств информационных технологий входят приспособления и инструменты, машины, механизмы, автоматические устройства. Также, их можно классифицировать на средства организационной, коммуникационной и вычислительной техники.

В состав организационной техники входят средства, которые облегчают и обеспечивают офисную и инженерно-техническую работу, копировальное и проектное оборудование (компьютеры, принтеры, проекторы, локальная и глобальная сети). К коммуникационной техники входят телефоны, радиосвязь, факс и другие предметы, с помощью которых осуществляется передача информации. В состав компьютерной техники входят автоматизированные средства обработки данных и информации.

Средства информационных технологий позволяют сделать процесс обработки информации и данных наиболее легче, удобнее и быстрее.

# **2. СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ (СКВ) НА ПРЕДПРИЯТИИ.**

# **2.1 ОБЩЕЕ ОПИСАНИЕ СКВ, ИСПОЛЬЗУЮЩИХСЯ НА ПРЕДПРИЯТИИ. ОБЗОР ДЕЦЕНТРАЛИЗОВАННЫХ И ЦЕНТРАЛИЗОВАННЫХ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ.**

С приходом более совершенных IDE и языков программирования, вопрос взаимодействия с коллегами является ключевым для многих разработчиков. В основном чтобы решить эту проблему, начали появляться централизованные системы контроля версий (ЦСВ или CVS – Centralized Version Control systems). Примерами таких систем являются:

* CVS;
* Subversion;
* Perforce.

В них используется следующий метод сохранения информации: имеется один сервер, который хранит все версии файлов, а также некоторое количество клиентов, на которые поставляется информация с сервера. Диаграмма таких взаимоотношений представлена на схеме 1:



Схема 1 – Диаграмма взаимоотношений в VCS

Такой подход имеет много преимуществ, особенно относительно RCS. Основное из них в том, что разработчики могут работать с одним и тем же файлом одновременно. Кроме этого, есть ещё и другие плюсы, например, каждый разработчик в любой момент времени может знать, что делают все остальные; кроме этого, администраторы систем, имея точно такую же возможность, могут эффективнее управлять своими кадрами и всем продуктом в целом.

Но, несмотря на решение достаточно большого спектра проблем, у таких систем есть свои недостатки. Первый и наиболее существенный – единая точка отказа. Если сервер по каким-то причинам выходит из строя, то все работают только со своими локальными копиями, не имея возможности смотреть изменения других разработчиков.

Чтобы решить эту проблему, и, в свою очередь забыть о проблемах RCC, существуют децентрализованные системы контроля версий (ДСКВ или DVCS – Distributed Version Control System). В качестве примеров можно привести следующие ДСКВ:

* Git;
* Mercurial;
* Darcs;
* Bazaar.

Как следует из названия, в таких системах клиенты не просто забирают последнюю (или выбранную, но всегда одну) версию проекта: на самом деле они полностью копируют его содержание и историю. Таким образом, при таком подходе у системы появляются два больших преимущества перед VCS:

* В любой момент времени разработчик может посмотреть состояние проекта в какой-то момент времени даже без наличия доступа к центральному серверу;
* Каждая копия проекта это, по сути дела, полная резервная копия проекта. Таким образом, если сервер выйдет из строя, то любой разработчик сможет восстановить его состояние на любой другой машине.

Диаграмма взаимоотношений между клиентом и сервером представлена на схеме 2:



Схема 2 – Децентрализованная система контроля версий

На предприятии Epam Systems используется достаточно много различный версий VCS. Это объясняется спецификой заказов (а именно необходимостью поддерживать различные устаревшие проекты). Тем ни менее, основную часть VCS составляют git-системы. На них пишутся большинство новых приложений и проектов. Во время прохождения практики мною были изучены возможности именно git-систем в рамках сразу двух проектов. Таким образом, можно сделать вывод, что именно они являются преобладающими.

Тем ни менее, также большое количество проектов написаны с использованием svn-систем. В основном это достаточно долгосрочные проекты для крупных международных организаций, то есть те, старт которых был дал примерно в прошлом десятилетии. О способе решения проблем, связанных с svn системами написано в пункте 2.5.

На основании полученных данных, в моей дипломной работе я планирую использовать git как децентрализованную систему и svn (subversion) как централизованную.

Основная причина в использовании git заключается в его популярности. Git является одним из основных инструментов, которые требуют заказчики, а также в том, что именно для работы с git написан самая популярная работа для работы с файлами – GitHub.

Основная причина в использовании subversion в качестве одного из инструментов в дипломной работе заключается в необходимости поддержки уже существующего ПО. На данный момент в брестском офисе EPAM Systems не менее 10% программистов используют именно subversion в качестве рабочего инструмента.

Таким образом, именно эти две системы составляют основу систем контроля версий в EPAM Systems, и, как следствие, именно они были выбраны в качестве рабочих для дипломного проекта.

# **2.2 ТИПОВЫЕ ОПЕРАЦИИ И ПОРЯДОК РАБОТЫ С СИСТЕМОЙ КОНТРОЛЯ ВЕРСИЙ.**

Первое, что делает любой разработчик, начиная работать с проектом – создание рабочей копии проекта. В Subversion для этих целей служит команда checkout, для Git – clone. После выполнения этой команды на рабочей машине разработчика появляется директория, в которой находятся непосредственно сам проект. После этого можно непосредственно приступать к разработке.

Стоит отметить, что, в зависимости от реализации системы контроля версий, может установиться не одна папка, а две: одна с рабочей копией, а вторая – полный её дубль, позволяющий в любой момент времени локально определить, какие файлы были изменены.

Далее разработчик будет модифицировать рабочую копию директории, чтобы в дальнейшем зафиксировать изменения (commit). После этого изменённое состояние отправляется на сервер (push). Кроме этого, можно узнать текущее состояние сервера (pull).

Тем ни менее, такой идеальный сценарий случается редко, так как клиенты зачастую вносят правки в одну и ту же копию документа одновременно (при этом два и более клиента могут быть одним и тем же пользователем, просто работающим с разных машин). В таком случае применяется система разрешения конфликтов (resolve conflicts). Существуют следующие способы решения конфликтов:

* Вручную. При таком сценарии каждый файл пересматривается руками; из него удаляются вспомогательные строки, например, “<<<<<<< HEAD” и “>>>>>>> master”, указывающие пользователю, каким образом строки конфликтуют между собой, а также применяются изменения, которые должны быть зафиксированы в текущей версии проекта;
* При помощи специальных программ. На данный момент их насчитывается несколько десятков, самые популярные из них это KDiff3, DiffMerge, P4Merge и другие, которые могут работать на разных ОС, с разными СКВ, с разными проектами. Кроме этого, большинство таких систем бесплатны или условно бесплатны, последняя даёт возможность пользования в целях учёбы или в некоммерческих организациях;
* При помощи встроенных возможностей СКВ. Некоторые СКВ имеют возможность автоматического разрешения конфликтов, в более же сложных ситуациях же они отдают выбор способа решения пользователю (например, команда git mergetool).

Чаще всего такие ситуации возникают при слиянии двух веток в одну, так как традиционно процесс разработки в команде строится таким образом, чтобы каждый разработчик работал в одной ветке проекта, а потом, после выполнения текущего задания и удачного прохождения тестов (или после ручной проверки), объединял все изменения в главную ветку (чаще всего – master).

Ветка (branch) – это на самом деле ссылка на какую-либо версию проекта. Каждый раз, когда происходит фиксация изменений, ссылка автоматически меняет свой адрес на следующую версию. Именно при помощи механизма веток в VCS стало возможным распараллеливание работы разработчиков.

# **2.3 СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ, ИСПОЛЬЗУЮЩИЕСЯ НА ПРЕДПРИЯТИИ.**

На предприятии EPAM Systems используется GitLab ­– FOSS решение для хостинга git-репозиториев внутри закрытой инфраструктуры. Функциональность во многом аналогична GitHub (github.com), в частности доступны базовые возможности администрирования и разделения полномочий между пользователями, issue трекер, wiki, code review и мердж реквесты (аналог пулл-реквестов на GitHub).

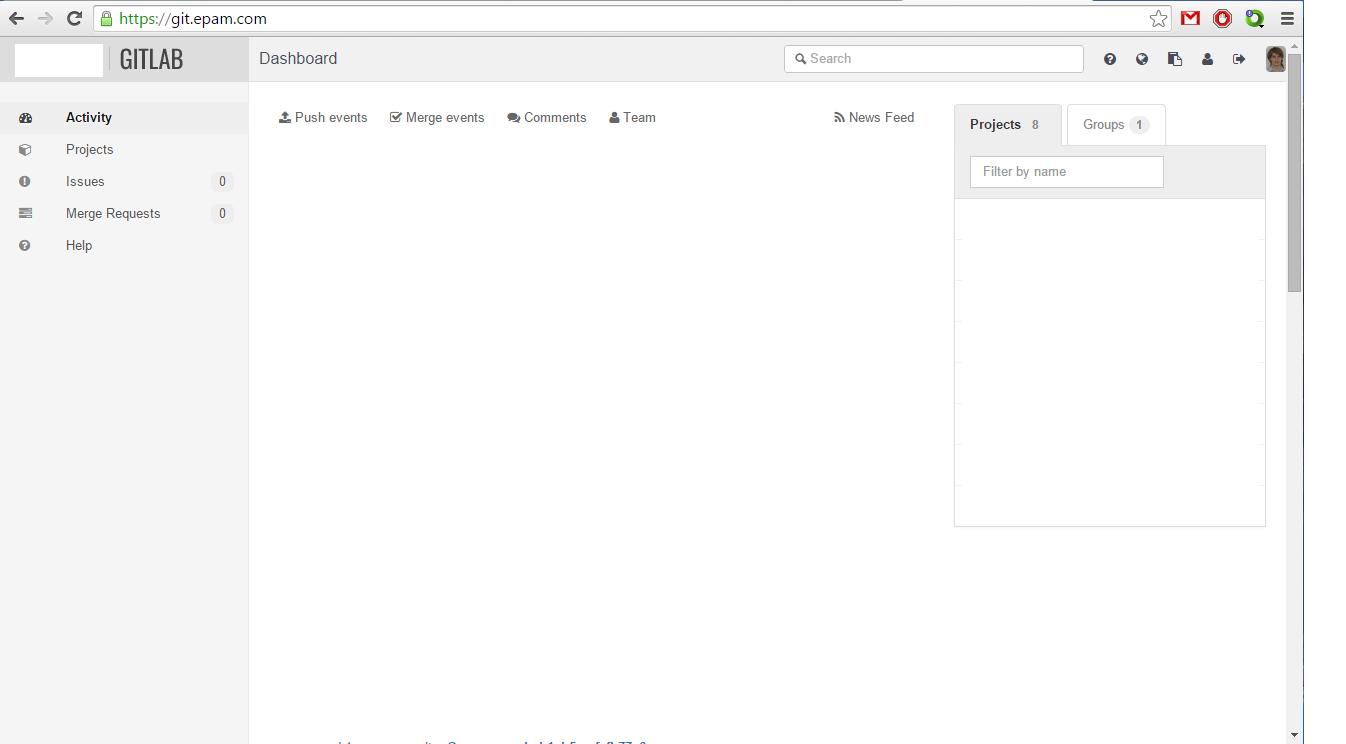


Рисунок 2.1 – приблизительный интерфейс системы gitlab в корпоративной сети epam.com.

GitLab часто применяется для организации командной разработки в корпоративных проектах, так как позволяет без привязки к облачному вендору развернуть инструментарий, похожий на Github Enterprise и Gitorious. Web-интерфейс GitLab во многом напоминает GitHub, но, тем не менее, не копирует его, а лишь повторяет наиболее удачные принципы. GitLab поддерживает создание отдельных проектов, отслеживание ошибок, обработку запросов на добавление кода (рабочий процесс основан на обработке merge-запросов), навигацию по веткам и тегам, контроль за изменениями, рецензирование кода, многоуровневое управление доступом, Wiki, обмен небольшими кусками кода, наглядный анализ различий между версиями кода, средства визуализации ветвления репозитория и многое другое.

Код проекта написан на языке Ruby с использованием фреймворка Ruby on Rails и распространяется под свободной лицензией MIT. Для хранения данных поддерживается связка Redis с MySQL или PostgreSQL. Параллельно развивается свободный сервер непрерывной интеграции GitLab CI, который можно использовать совместно с GitLab для автоматизации тестирования и постоянного контроля за качеством кодовой базы, в том числе обеспечивая проверку пересборкой для каждого вносимого изменения.

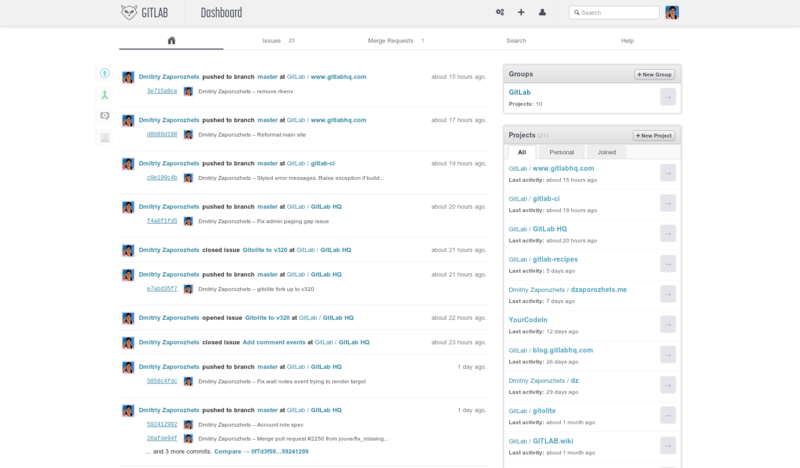


Рисунок 2.2 – интерфейс системы GitLab.

Поддержка создания или удаления git-веток и тегов доступна из web-интерфейса GitLab, что избавляет разработчиков от необходимости обращения к git вручную, так как теперь все операции доступны через интерфейс пользователя. Например, для создания ветки можно просто поправить код в web-редакторе GitLab. Запросы на слияние (Merge Request) теперь можно отправлять между веткой (branch) и оригинальным проектом, что довольно удобно в повседневной разработке ПО.

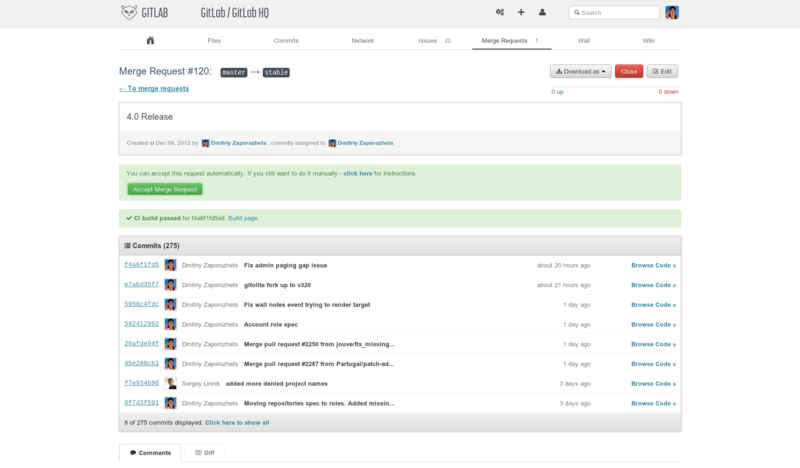


Рисунок 2.3 – merge request в системе gitlab.

Кроме этого, GitLab позволяет достаточно просто работать с файлами: поддерживается возможность просмотра изменений между двумя коммитами (или между двумя файлами в разных коммитах); также достаточно неплохо реализован интерфейс команды blame, позволяющей узнать автора той или иной строчки кода, что достаточно удобно для менеджеров и ведущих инженеров проекта.

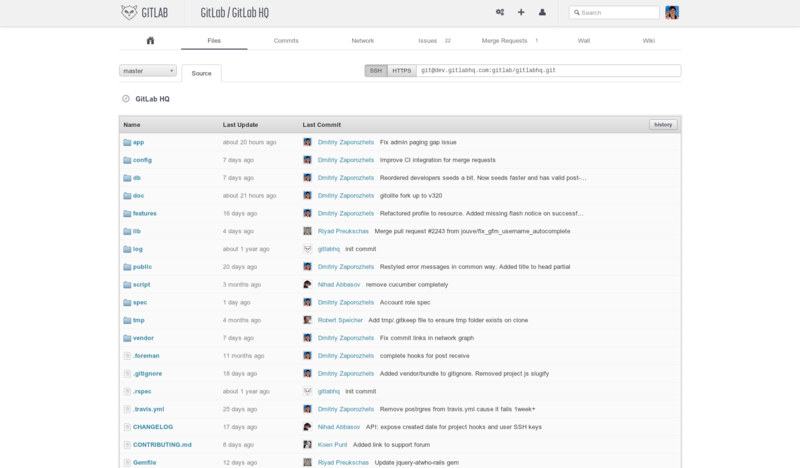


Рисунок 2.4 – файлы и директории в системе gitlab.

Кроме этого, есть возможность работать с заданиями (issues), что позволяет не использовать сложные системы наподобие jira:

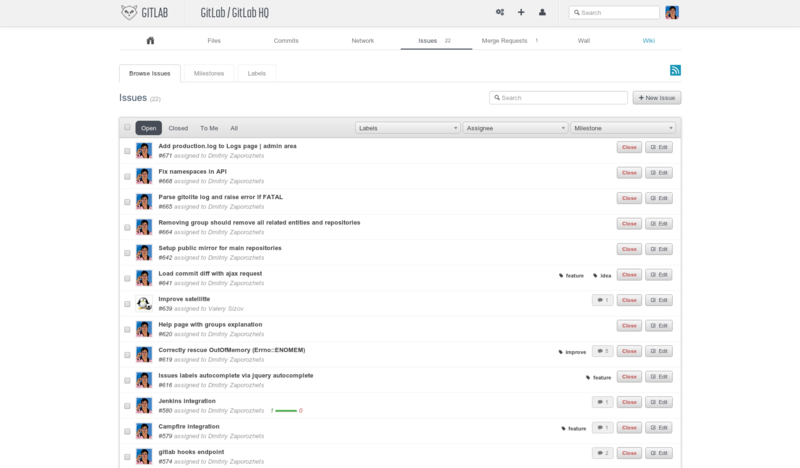


Рисунок 2.5 – работа с issues в системе GitLab.

# **2.4 РАЗРАБОТКА КОНЦЕПЦИИ ЛАБОРАТОРНОГО ПРАКТИКУМА.**

В ходе прохождения преддипломной практики были выявлены наиболее часто используемые функции git и subversion в процессе разработки ПО. На их основе будет составлен лабораторный практикум. Функции git:

1. git add

Подготавливаем коммит, добавляя в него файлы.

1. git status

Список всех измененных и добавленных файлов.

1. git commit

Выполнение коммита.

1. git pull

Заливаем последнюю версию ветки dev на удаленный сервер.

1. git stash

Прятанье поглощает грязное состояние рабочего каталога, то есть изменённые отслеживаемые файлы и изменения в индексе, и сохраняет их в стек незавершённых изменений.

1. git checkout

Чтобы создать ветку и сразу же перейти на неё, можно выполнить это команду с ключом –b.

1. git log

Просмотр логов.

1. git merge

Выполняет слияние текущей и указанной ветки. Изменения добавляются в текущую ветку.

1. git cherry-pick

Помогает применить один-единственный коммит из одной ветки к дереву другой.

Функции subversion:

1. svn checkout http://repository.url/svn/name

Извлекаем файлы проекта из репозитория, сокращение: svn co.

1. svn update

Получаем обновления из репозитория, сокращение: svn up.

1. svn add ./file\_name

Добавляем файл в репозиторий (не важно текстовый или бинарный).

1. svn revert -R ./

Откатывает все локальные изменения файлов.

1. svn blame ./file\_name

Показывает авторов изменений файла построчно, синоним: svn annotate.

1. svn cleanup

Снимает блокировки с файлов.

1. svn diff ./file\_name

Показывает локальные изменения в файле построчно;

1. svn copy http://repository.url/svn/name/trunk/ http://repository.url/svn/name/branches/new\_branch\_name/

Создаем ветку с названием new\_branch\_name из главной линии разработки.

1. svn merge -r rev\_num1:rev\_num2 http://repository.url/svn/name/trunk/

Синхронизирует вашу ветку с главной линией разработки с учетом ревизий: rev\_num1 — номер ревизии, когда ваша ветка была «открыта», или это м.б. номер предыдущего объединения (слияния), rev\_num2 — версия главной линии разработки, с которой производим объединение. Необходимо отметить, что все изменения будут применены для директории, в которой выполнялась эта команда.

Для данных функций (а также некоторые других) необходимо будет написать соответствующую пояснительную документацию, а также придумать задания для реализации. Кроме этого, необходимо будет использовать инструмент для проверки уже существующих решений студентов.

# **2.5 РЕКОМЕНДАЦИИ ПО УЛУЧШЕНИЮ РАБОТЫ С СИСТЕМАМИ КОНТРОЛЯ ВЕРСИЙ НА ПРЕДПРИЯТИИ.**

На самом деле, почти все проекты на svn можно перевести на git-системы. Это позволит достичь лучшей производительности и надёжности всей системы в целом. Ниже представлена краткая инструкция, полученная в результате проведённой работы в ходе прохождения преддипломной практики.

1. Установка сервера для центрального репозитория

1.1. Установка gitolite

$ emerge gitolite

1.2. Первичная настройка

$ ssh-keygen -t rsa

$ cp ~/.ssh/id\_rsa.pub /tmp/admin.pub

$ su git

$ cd

$ mkdir -p bin

$ gitolite/install -ln

$ gitolite setup -pk /tmp/admin.pub

1.3. Создание репозитория для проекта

$ cd

$ git clone git@server:gitolite-admin.git

repo project

RW+ = javer

$ git add .

$ git commit -am "Repository for project added"

$ git push origin master

2. Импорт проекта из SVN

$ git svn clone

2.1. Определение стартовой ревизии

-r19815:HEAD

2.2. Соответствие SVN-пользователей с Git-пользователями

javer = javer <javer@domain.tld>

developer1 = developer1 <developer1@domain.tld>

...

--authors-file=/path/to/authors

2.3. Исключение ненужных файлов

--ignore-paths="\.(avi|mov)$"

2.4. Дополнительные опции

--username javer

git-svn-id: svn://svn.domain.tld/repo/project/trunk@19815 e13dc095-444b-fa4e-8f24-06838a8318a5

2.5. Клонирование проекта из SVN-репозитория

$ cd

$ mkdir project && cd project

$ git svn clone -r19815:HEAD --authors-file=/path/to/authors --ignore-paths="\.(avi|mov)$" --username javer --no-metadata svn://svn.domain.tld/repo/project/trunk .

2.6. Исключение более ненужных файлов и каталогов

$ git filter-branch --tree-filter 'rm -rf unneeded\_directory' -f HEAD

2.7. Удаление пустых коммитов

$ git filter-branch --commit-filter 'git\_commit\_non\_empty\_tree "$@"' HEAD

2.8. Пустые каталоги и svn:ignore

$ git svn create-ignore

$ git add .

$ git commit -am "Added .gitignore"

2.9. Удаление упоминания об SVN

$ git branch -rd git-svn

$ git config --remove-section svn-remote.svn

$ git config --remove-section svn

$ rm -rf .git/svn

2.10. svn:externals

$ git submodule add git://github.com/propelorm/sfPropelORMPlugin.git plugins/sfPropelORMPlugin

$ git submodule add git://github.com/n1k0/npAssetsOptimizerPlugin.git plugins/npAssetsOptimizerPlugin

$ git submodule add git@server:customPlugin.git plugins/customPlugin

$ git submodule update --init --recursive

$ git commit -am "Added submodules: sfPropelORMPlugin, npAssetsOptimizerPlugin, customPlugin"

2.11. Отправка локальной копии проекта на сервер

$ git gc

$ git remote add origin git@server:project.git

$ git push origin master

2.12. Обновление submodules в будущем

$ git pull

$ git submodule update

$ git submodule foreach git pull

$ git commit -am "Updated submodules"

3. Настройка прав доступа к репозиторию

3.1. Пользовательские ключи для Linux/Unix

$ ssh-keygen -t rsa

3.2. Настройка прав доступа

Полученные на предыдущем шаге публичные ключи пользователей помещаем в репозиторий администратора в каталог ~/gitolite-admin/keydir/ в файлы с названиями USERNAME.pub, где USERNAME — имя пользователя.  
  
 Поскольку gitolite имеет достаточно широкие возможности по настройке, используем их для настройки прав доступа к репозиторию нашего проекта. Для этого редактируем файл ~/gitolite-admin/conf/gitolite.conf и приводим его к виду:

@owners = javer  
@project\_developers = user1 user2 user3  
@deploy = root@production  
  
repo project  
- master$ = @project\_developers  
- refs/tags = @project\_developers  
RW+ = @project\_developers @owners  
R = @deploy

Этим мы даем полный доступ для группы пользователей owners. Для группы project\_developers — также полный доступ с возможностью создания своих веток, за исключением записи в ветку master и создания тегов. Для группы deploy, которая используется для deploy на production, разрешаем доступ только для чтения. В конце не забываем сохранить все изменения:

$ git add .

$ git commit -am "New users for project: user1, user2, user3..."

$ git push origin master

4. Установка и настройка на машинах разработчиков Linux/Unix

$ git config --global user.name "javer"

$ git config --global user.email "user@domain.tld"

# **ЗАКЛЮЧЕНИЕ**

Был проведен анализ организационно-управленческой структуры подразделения, служебной документации. Изучены используемые средства информационных технологий, изучены должностные инструкции работников подразделения, рассмотрены задачи, решаемые подразделением и информационные технологии, используемые на предприятии изучены нормативные документы и стандарты (ГОСТ 19.701, UML и пр.), описаны задачи, проблемы и процессы автоматизируемой предметной области, сформулированы требования к системе, выбрана структура системы автоматизации. Было проведено описание предметной области, а также описание технологий, которые будут использоваться при выполнении дипломной работы.

Условия практики соответствовали требованиям. Мне была оказана всяческая помощь при прохождении преддипломной практики.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Официальный сайт компании <EPAM>. Режим доступа: <http://epam.by>. Дата доступа 20.03.2014
2. Система управления версиями. Режим доступа: <https://ru.wikipedia.org/wiki/Система_управления_версиями>. Дата доступа 19.03.2015
3. Переезд проекта с SVN на Git. Режим доступа: <http://habrahabr.ru/post/144626/>. Дата доступа: 20.03.1015