НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГУМАНИТАРНЫЙ
УНИВЕРСИТЕТ ПРОФСОЮЗОВ»

Кафедра Информатики и математики

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ
Программная инженерия

Основная профессиональная образовательная программа
высшего образования программы бакалавриата
по направлению подготовки

09.03.03 «Прикладная информатика»

Профиль подготовки «Прикладная информатика в экономике»

Квалификация:
Бакалавр

Согласовано:
Руководитель ОПОП по направлению 09.03.03 «Прикладная информатика» Профиль «Прикладная информатика в экономике»
_____/Путькина Л.В.

Рассмотрена и утверждена на заседании кафедры

«01» июня 2020 г., протокол № 10

Зав. кафедрой _____/Путькина Л.В.
Рекомендована решением
Методического совета

«15» июня 2020 г., протокол № 10

Секретарь МС _____Волкова А.М.

Авторы-разработчики:

_____/Мокрый В.Ю.

Санкт-Петербург

СТРУКТУРА

1. Цель и задачи освоения дисциплины
2. Место дисциплины в структуре ОПОП
3. Требования к результатам освоения дисциплины
4. Тематический план изучения дисциплины
5. .Содержание разделов и тем дисциплины
6. План подгрупповых (лабораторных) занятий
7. Образовательные технологии
8. План самостоятельной работы студентов
9. Контроль знаний по дисциплине
10. Учебно-методическое и информационное обеспечение дисциплины
11. Материально-техническое обеспечение дисциплины

Учебно-методическое обеспечение самостоятельной работы студентов

1. Методические рекомендации по организации самостоятельной работы студентов
2. Методические рекомендации по подготовке к практическим (подгрупповым) занятиям
3. Методические рекомендации по написанию контрольных работ
4. Методические рекомендации по написанию курсовой работы

Оценочные и методические материалы

1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы
2. Описание показателей и критериев оценивания компетенций, шкал оценивания
3. Типовые контрольные задания и методические материалы, процедуры оценивания знаний, умений и навыков

Глоссарий

Методические рекомендации для преподавателя по дисциплине

1. **Цель и задачи освоения дисциплины:**

Целью изучения дисциплины является расширение мировоззрения и формирование у студентов знаний, представлений и навыков о промышленной разработке программ.

Основные **задачи** дисциплины - изучение способов сертификации персонала и информационных продуктов, получение систематических знаний о стандартах, используемых в данной области, приобретение специальных знаний и умений, необходимых для ведения проектов в области информационных технологий и оценки качественных показателей разрабатываемых программных продуктов.

2. **Место дисциплины в структуре ОПОП**

Междисциплинарные связи с обеспечиваемыми (последующими) дисциплинами

№ п/п	Наименование обеспечиваемых (последующих) дисциплин	№ № разделов данной дисциплины, необходимых для изучения обеспечиваемых дисциплин			
		1	2	3	4
1.	Объектно-ориентированное программирование	+	+	+	+
2.	Проектирование информационных систем	+	+		
3.	Проектный практикум	+	+	+	+
4.	Высокоуровневые методы информатики и программирования	+	+	+	+

3. Требования к результатам освоения дисциплины

Процесс изучения дисциплины направлен на формирование следующих компетенций с установленными к ним индикаторами:

Компетенции и индикаторы их достижения

Категория компетенций	Код и наименование компетенции	Код и наименование индикатора достижения компетенции
Исследовательская деятельность	ОПК-7. Способен разрабатывать алгоритмы и программы, пригодные для практического применения	ОПК-7.1. Знает принципы сбора, отбора и обобщения информации.
		ОПК-7.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами.
		ОПК-7.3. Владеет навыками программирования, отладки и тестирования программного обеспечения.
Исследовательская	ОПК-9. Способен принимать участие в реализации	ОПК-9.1. Знает принципы организации проектирования и содержание этапов

деятельность	профессиональных коммуникаций с заинтересованными участниками проектной деятельности и в рамках проектных групп	процесса разработки программных комплексов дисциплины, задачи и методы исследования и обеспечения качества и надежности программных компонентов, основы разработки программных комплексов.
		ОПК-9.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами.
		ОПК-9.3. Владеет навыками работы с инструментальными средствами моделирования предметной области, прикладных и информационных процессов, использования технологических стандартов информационных систем и программных продуктов.

4. Тематический план изучения дисциплины

См. приложение

5. Содержание разделов и тем дисциплины

РАЗДЕЛ 1. Технологии разработки программного обеспечения

Тема 1. Жизненный цикл программного обеспечения

История разработки программного обеспечения как промышленной технологии. Проблема внедрения информационных технологий и программного обеспечения. Понятие о программном продукте. Рынок программных средств. Конкурентоспособность программного обеспечения. Характеристики программного обеспечения. Компоненты программного средства. Мифы программного обеспечения. Занятость в сфере разработки программного обеспечения. Экономика, этика и правовые нормы. Жизненный цикл программного обеспечения. Модели жизненного цикла. Каскадная модель жизненного цикла ПО и ее принципиальные особенности. Модель с перекрытиями. Итерационная и контролируемая итерационная модели. Спиральная модель. Сравнение моделей. Стадии и рабочие процессы жизненного цикла. Организационные мероприятия.

Прикладные программы с высокой степенью автоматизации управления. Адаптируемость пакетов программ. Проектирования программ сложной структуры. Типовые приемы конструирования пакетов программ сложной структуры.

Программное обеспечение поддержки жизненного цикла. Классификация CASE. Понятия методологии, метода, нотации и средства CASE. Репозиторий и артефакты процесса разработки. Прототипы и их классификация. Возвратное проектирование. Реинжиниринг. Шаблоны проектирования, цели, правила применения. Рефакторинг, цели, правила применения. Каталог рефакторингов, классификация, примеры применения. Обзор современных методологий и CASE.

Тема 2. Анализ и проектирование программного обеспечения

Общая характеристика этапов анализа, проектирования и реализации

программного обеспечения. Анализ и проектирование программного обеспечения на основе структурного подхода. Сущность структурного подхода к разработке программного обеспечения. Моделирование потоков данных (процессов). Диаграммы потоков данных. Контекстная диаграмма и ее детализация. Моделирование данных. Варианты спецификации процессов. Диаграммы Костантайна.

Анализ и проектирование программного обеспечения на основе объектно-ориентированного подхода. Сущность объектно-ориентированного подхода. Методология OMT. Методика Шлеера-Меллора. Информационное моделирование, модели состояний и процессов.

Унифицированный язык моделирования UML. Элементы моделей, основные отношения. Механизмы расширения UML. Варианты использования. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов. Моделирование поведения системы. Диаграммы взаимодействия (диаграммы последовательности и кооперативные диаграммы). Диаграммы состояний. Диаграммы действий. Моделирование реализации системы. Диаграммы компонентов. Диаграммы размещения. Статический и динамический виды. Генерация кода программ и описаний баз данных.

РАЗДЕЛ 2. Метрология и стандартизация

Тема 3. Метрология программного обеспечения

Меры, метрики и индикаторы. Метрики в и проектах: метрики процессов и улучшение процессов программного обеспечения, метрики проектов. Метрики для измерения программ. Программное обеспечение для измерения и контроля метрик.

Метрическая теория программ. Объемные метрики. Меры Холстеда. Длина программы, объемы программы, уровень программы, интеллектуальное содержание программы, работа программиста, уровень языка программирования, ожидаемое число ошибок в программе.

Топологические меры сложности. Сложность управляющего графа и цикломатическое число Мак-Кейба.

Меры сложности, основанные на концепции информационных потоков между компонентами системы. Архитектурные меры.

Введение в объектно-ориентированные метрики. Сложность классов, сложность взаимодействия классов, сложность иерархии наследования.

Свойства и характеристики качества программного обеспечения. Показатели качества. Конструктивные и функциональные критерии. Характеристики качества и их влияние на различных этапах жизненного цикла программного продукта. Методики оценки показателей качества программного обеспечения в нашей стране и за рубежом.

Тема 4. Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения

Стандартизация и сертификация в разработке программного обеспечения. Стандартизация информационных технологий; действующие стандарты и проблемы программных интерфейсов. Международные (ISO), национальные (ГОСТ, ANSI), и отраслевые стандарты (ЕСПД, CMM) и организации (OMG, SEI). Центры сертификации, сертификация персонала. Международные и отечественные стандарты, регламентирующие жизненный цикл программного обеспечения. Стандарт ISO/IEC 12207 и его практическое применение. Стандарты серии ISO 9000 и сложности их применения при стандартизации процесса разработки программного обеспечения. ГОСТы “Качество программных средств” и “Оценка качества программных средств”.

Способы формального представления знаний, основы устройства и использование экспертных систем в разработке адаптируемого программного обеспечения. Основные направления интеллектуализации ПО.

Модель зрелости процесса разработки программного обеспечения CMM. Уровни

зрелости – начальный повторяемый, определенный, повторяемый, оптимизирующий. Ключевые области обследования. Современное состояние и перспективы сертификации по CMM.

Стандарты по управлению жизненным циклом. Отраслевые стандарты для процесса разработки программного обеспечения. RUP - унифицированный процесс разработки объектно-ориентированного программного обеспечения с использованием UML. Модель жизненного цикла. Основные рабочие процессы, стадии. Заинтересованные лица и роли исполнителей проекта. Артефакты. Планирование итераций.

Обзор рабочих процессов RUP. Бизнес – моделирование. Анализ требований. Прецеденты использования и их детализация. Анализ и проектирование. Разработка и сопровождение. Поддерживающие рабочие процессы. Рекомендации по внедрению.

Рабочие процессы для малых проектов. Экстремальное программирование (XP). Особенности организации работ. Взаимодействие с заказчиком. Роль шаблонов проектирования, рефакторингов и модульного тестирования.

Стандартизация и творчество в программировании. Тенденции развития индустрии программного обеспечения. Основные направления интеллектуализации программного обеспечения.

6. План подгрупповых (лабораторных) занятий

№ п/п	Наименование темы дисциплины	Наименование и содержание подгрупповых (лабораторных) занятий, литература для подготовки к занятиям	Формируемые компетенции	Формы контроля усвоения знаний
1.	**Тема 1.** Жизненный цикл программного обеспечения **Тема 2.** Анализ и проектирование программного обеспечения	Визуальное моделирование на этапах анализа и проектирования программного продукта. 1). Разработка системы обработки заказов. При получении заказа заполняется форма заказа, которая поступает в магазин и в бухгалтерию. Система должна обеспечивать возможность добавления новых заказов, изменения старых, выполнения заказов, проверки и возобновления инвентарных описей. 2). Выполнение индивидуального задания по анализу и проектированию программного обеспечения на UML Литература: 1-3.	ОПК-7, ОПК-9	Устный опрос Проверка файлов общих и индивидуальных заданий, устный опрос, оперативное исправление ошибок
2.	**Тема 3.** Метрология программного обеспечения	Метрический анализ программ 1). Разработка вычислительного	ОПК-7, ОПК-9	Устный опрос Проверка файлов общих и индивидуальных

	Тема 4. Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения	алгоритма для заданного варианта программы на языке Паскаль. 2). Расчет метрик Холстеда вычислительного алгоритма для заданного варианта программы на языке Паскаль. 3). Выполнение индивидуального задания по вычислению дополнительных метрик. 4) Построение блок-схемы алгоритма программы и управляющего графа.5). Вычисление цикломатического числа и метрик сложности потока управления программы. Литература: 1-3.		заданий, устный опрос, оперативное исправление ошибок

7. Образовательные технологии

При проведении учебных занятий по дисциплине для успешного освоения применяются различные образовательные технологии, которые обеспечивают развитие навыков командной работы, межличностной коммуникации, принятия решений, лидерских качеств.

Методы / Формы	**Лекции (Л)**	**Семинарские занятия (С)**
Диалого-дискуссионное обсуждение проблем	+	+
Работа в команде		+
Поисковый метод	+	+
Проектный метод		+

8. План самостоятельной работы студентов

№ п/п	Содержание самостоятельной работы студентов	Формируемые компетенции	Форма отчетности студента
1	Изучение литературы по теме	ОПК-7, ОПК-9	Представления обоснованной и развернутой структуры работы
2	Подготовка к практическому занятию	ОПК-7, ОПК-9	Представление выполненных работ
3	Выполнение заданий лабораторных работ	ОПК-7, ОПК-9	Отчеты о выполнении лабораторных работ
4	Выполнение дополнительных заданий для самостоятельной работы	ОПК-7, ОПК-9	Отчеты о выполнении самостоятельных работ, тест-опрос

9. Контроль знаний по дисциплине

По дисциплине предусмотрены текущий контроль и промежуточная аттестация.

Текущий контроль успеваемости студента – одна из составляющих оценки качества усвоения образовательных программ. Текущий контроль проводится в течение семестра в результате проверки выполнения циклов лабораторных работ.

Промежуточная аттестация проводится по окончании изучения дисциплины в виде зачета. Вопросы к промежуточной аттестации сформулированы в **Оценочных и методических материалах**.

10. Учебно-методическое и информационное обеспечение дисциплины:

а) Основная литература

1. Лаврищева Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/436514
2. Лаврищева Е. М. Программная инженерия. Парадигмы, технологии и CASE-средства : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/444952
3. Черткова Е. А. Программная инженерия. Визуальное моделирование программных систем : учебник для академического бакалавриата / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/437536

б) Дополнительная литература:

1. Зыков С. В. Программирование : учебник и практикум для вузов / С. В. Зыков. — Москва : Издательство Юрайт, 2021. — Режим доступа: https://urait.ru/bcode/469579
2. Кувшинов Д. Р. Основы программирования : учебное пособие для вузов / Д. Р. Кувшинов. — Москва : Издательство Юрайт, 2021. — Режим доступа: https://urait.ru/bcode/473570
3. Путькина Л. В. Интеллектуальные информационные системы : учебное пособие / Л. В. Путькина, Т. Г. Пискунова ; СПб Гуманит. ун-т профсоюзов. — СПб. : Изд-во СПбГУП, 2008. — Режим доступа**:** http://library.gup.ru/jirbis2/index.php?option=com_irbis&view=irbis&Itemid=108&task=set_static_req&sys_code=32/39/П 90-825442&bns_string=IBIS

в) Периодические издания

1. Журнал «Вестник Томского государственного педагогического университета» [Электронный ресурс]. Режим доступа: https://vestnik.tspu.edu.ru/
2. Журнал «Проблемы передачи информации» [Электронный ресурс]. Режим доступа: http://sciencejournals.ru/journal/ppinf/

г) Лицензионное программное обеспечение

Перечень лицензионного и свободно распространяемого программного обеспечения:

1. Семейство программ Microsoft Office Standart Russian (Включает набор продуктов: Word, Excel, PowerPoint, Publisher, Outlook);
2. Mirapolis Virtual Room;
3. Антиплагиат;
4. КонсультантПлюс
5. Project Expert 7
6. Prime Expert
7. FineModel Expert
8. Обеспечено доступом к сети «Интернет» и электронной информационно-образовательной среде СПбГУП

д) Современные профессиональные базы данных и информационные справочные системы

1. Официальный сайт СПбГУП: http://www.gup.ru/
2. Электронно-библиотечная система СПбГУП,
3. Системы поддержки самостоятельной работы СПбГУП: http://edu.gup.ru/
4. Справочная правовая система «Консультант плюс» http://www.consultant.ru
5. Российское образование http://www.edu.ru/
6. Единое окно доступа к образовательным ресурсам http://window.edu.ru/
7. Электронно-библиотечная система http://e.lanbook.com/

11. Материально-техническое обеспечение дисциплины

Аудиторный фонд с демонстрационным оборудованием и техническими средствами обучения, учебно-наглядные пособия и методические ресурсы кафедры, фонды Научной библиотеки.

Изучение дисциплины инвалидами и обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья обучающихся.

УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

1. Методические рекомендации по организации самостоятельной работы студентов

Целями самостоятельной работы являются расширение и углубление знаний по определенным темам курса, а также развитие навыков анализа теоретических и практических проблем на основе изучения монографий, учебников, учебных пособий и т. п. Самостоятельная работа содействует развитию творческого отношения к учебной деятельности, повышает ее эффективность, а также активизирует усилия студента по освоению изучаемого предмета.

Подгрупповые занятия — важная форма учебного процесса. Они способствуют закреплению и углублению знаний, полученных студентами на лекциях и в результате самостоятельной работы над научной и учебной литературой и нормативными источниками. Они призваны развивать самостоятельность мышления, умение делать выводы, связывать теоретические положения с практикой. На занятиях вырабатываются необходимые каждому специалисту навыки и умения, логика, культура профессиональной речи. Кроме того, семинары — это средство контроля преподавателей за самостоятельной работой студентов, они непосредственно влияют на уровень подготовки к итоговым формам отчетности — зачетам и экзаменам. В выступлении на занятии должны содержаться следующие элементы:

- четкое формулирование соответствующего теоретического положения в виде развернутого определения;
- приведение и раскрытие основных черт, признаков, значения и роли изучаемого явления или доказательства определенного теоретического положения;
- подкрепление теоретических положений конкретными фактами.

Для качественного и эффективного изучения дисциплины необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Многие новейшие научные положения появляются, прежде всего, в статях, опубликованных в журналах.

Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Для качественного и эффективного изучения дисциплины необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Многие новейшие научные положения появляются, прежде всего, в статях, опубликованных в журналах.

Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Работа с научной литературой, в конечном счете, должна привести к выработке у студента умения самостоятельно размышлять о предмете и объекте изучения, которое должно проявляться:

- в ясном и отчетливом понимании основных понятий и суждений, содержащихся в публикации, разработке доказательств, подтверждающих истинность тех или иных положений;
- в понимании студентами обоснованности и целесообразности, приводимых в книге и статье примеров, поясняющих доказательства и выводы автора. При этом будет уместно, если студент самостоятельно приведет дополнительные примеры к этим выводам;
- в отделении основных положений от дополнительных, второстепенных сведений;
- в способности студента критически разобраться в содержании публикации, определить свое отношение, дать общую оценку, характеристику.

Самостоятельная работа студентов предполагает тщательное освоение студентами учебной и научной литературы по изучаемой дисциплине. Изучение научной литературы – это серьезная работа, которую следует проводить по этапам:

- общее ознакомление с источником в целом по его оглавлению;
- беглый просмотр всего содержания;
- последовательное чтение материала;
- выборочное чтение какой-либо части текста;
- выписка материала, представляющего интерес;
- критическая оценка выписанного материала, его сравнение с другими точками зрения.

При самостоятельном изучении *основной рекомендуемой литературой* студентам необходимо обратить внимание на выделение основных понятий, их определение, узловых положений, представленных в изучаемом тексте.

В качестве информационно-справочного материала можно использовать глоссарий, в котором содержится перечень и определение ключевых понятий документооборота.

Нужно обратить внимание на образно-схематическое представление излагаемого материала в виде рисунков, схем, графиков и диаграмм, присутствующих в изучаемом тексте. Они способствуют более быстрому восприятию и запоминанию учебного материала.

Для контроля усвоения содержания темы в конце соответствующих глав и параграфов учебников и учебных пособий обычно дается перечень контрольных вопросов, на которые студент должен уметь дать четкие и конкретные ответы.

Самостоятельная работа с дополнительной литературой предполагает умение студента выделять в ней необходимый аспект изучаемой темы (сто в данном материале относится непосредственно к изучаемой теме и основным вопросам). к дополнительной литературе как правило относят широкий спектр текстов (учебных, научных, научно-популярных, художественных, публицистических и др.), в которых изучаемых вопрос рассматривается частично либо с нетрадиционной точки зрения.

Знакомство с дополнительной литературой, несомненно, обогащает знания и расширяет научный кругозор студентов.

Дополнительную литературу нужно обрабатывать после основной литературы и исследовать ее надо комплексно, рассматривая различные точки зрения на исследуемый вопрос.

При самостоятельном освоении информационных технологий и работе с применением программных продуктов следует использовать методическую литературу, содержащую детальное описание последовательности и технологии выполнения базовых операций.

Для закрепления теоретических знаний и практических навыков рекомендуется отрабатывать технологии на контрольном примере.

Общими правилами самостоятельного изучения элементов курса являются:

- изучение теоретических основ соответствующих разделов программы, изложенных в лекционном материале и литературных источниках;
- обучение на контрольных примерах (с заранее известным результатом – правильным ответом);

- эффективное использование системы помощи (HELP) применяемого программного обеспечения;
- заданная последовательность в изучении и освоении курса («от простого к сложному» — от отдельных элементарных операций к комплексным);
- использование индивидуальных заданий, уменьшающих вероятность списывания недобросовестными студентами;
- подготовка к ответам на контрольные вопросы.

Основные виды самостоятельной работы студентов:

1. ***Предварительная подготовка к занятиям, в том числе и к тем, на которых будет изучаться новый материал***. Такая подготовка предполагает изучение учебной программы, установление связи с ранее полученными знаниями, выделение наиболее значимых и актуальных проблем, на изучение которых следует обратить особое внимание, ознакомление с материалом учебника. Также рекомендуется работа со словарями по новым экономическим терминам.

2. ***Прослушивание и восприятие лекций***, что предполагает осмысление учебной информации, сообщаемой преподавателем, ее обобщение и краткую запись, своевременную доработку конспектов лекций. Основная задача студентов на лекционных занятиях – выделить главное в изучаемом материале, а также уяснить связь данной темы с логикой дисциплины в целом и предыдущих лекций. На лекционных занятиях приветствуются уточняющие вопросы со стороны студентов.

3. ***Подбор, изучение, анализ и конспектирование рекомендуемой литературы***. Данная форма самостоятельной работы позволяет углубить знания в рамках определенной темы учебной программы. Основные рекомендации для эффективного освоения дополнительного материала: 1)до чтения: осмысление заголовка, анализ оглавления и аннотации; 2)по ходу чтения: старайтесь понять все слова и предложения в тексте, выделить важные и ключевые из них; задавайте вопросы самому себе по содержанию прочитанного, стройте свои предположительные ответы и сверяйте их с текстом; по ходу чтения попытайтесь осознать, что Вам не понятно, в чем возникают сомнения; делайте выписки, выделяйте главные мысли, составляйте схемы, графики, таблицы; 3)после прочтения: сформулируйте главную мысль прочитанного; прочитайте повторно наиболее сложные для Вас части текста; задайте вопросы себе по всему тексту; составьте конспект прочитанного; через время проверьте себя. Предварительное изучение рекомендуемой литературы позволяет отобрать необходимую учебную информацию и выяснить, по каким вопросам следует подобрать дополнительные литературные источники.

4. ***Подготовка к семинарским, практическим занятиям и зачету***. Подготовку к семинару и практическому занятию лучше начинать сразу же после лекции по данной теме. Прежде всего, следует доработать текст лекции по соответствующей теме, внимательно изучить план семинара (практического занятия), содержания основных учебных вопросов, выносимых для обсуждения, а также список рекомендованной литературы и дополнительные задания, которые могут быть даны преподавателем. Далее необходимо подобрать просмотреть литературу, которая рекомендована для подготовки к занятию. Это даст возможность выбрать те источники, где имеются ответы на поставленные учебные вопросы. Затем более внимательно ознакомиться с содержанием книги или статьи, отметить те части текста, в которых вопросы семинара раскрываются наиболее глубоко и подробно. В ходе углубленного чтения выделяются главные мысли, составляются краткие конспекты, тезисы своих будущих выступлений. Конспекты, которые отражают содержание вопросов семинарского и практического занятия, лучше всего вести в той же тетради, в которой конспектируются лекции по дисциплине. Концентрация всей информации по проблеме в одном месте позволяет студентам активно участвовать в обсуждении вопросов семинара. В дальнейшем такие записи значительно облегчат подготовку к зачету. Подготовка к семинарским, практическим занятиям, зачету не сводится только к поиску ответов на поставленные вопросы. Любая теоретическая проблема должна быть осмыслена с точки зрения ее связи с реальной жизнью и

возможностью реализации на практике. По каждому вопросу студент должен быть готов высказать и свою точку зрения.

Особую роль самостоятельная работа играет при освоении материала данной дисциплины. Научить программированию нельзя. Программированию можно только научиться. Преподаватель в этом процессе самообучения играет роль консультанта, более опытного коллеги и заказчика одновременно.

Дисциплина обеспечивает развитие базовых знаний и навыков в области информатики и программирования.

Самостоятельная работа студента по этой дисциплине способствует расширению мировоззрения и формированию у студентов самостоятельного мышления в области информационных подходов к анализу процессов и явлений профессиональной сферы и информационных технологий; закреплению систематических знаний об информационных процессах, системах, средствах и технологиях.

Самостоятельная работа является основой подготовки студента к выполнению контрольных индивидуальных заданий тестов и работ. *Целью* методических рекомендаций является помощь студентам и в организации самостоятельной работы.

Задачи методических рекомендаций:

– активизация самостоятельной работы студентов по освоению учебной дисциплины;
– содействие развитию у студентов алгоритмического и объектного мышления, творческого отношения к изучению дисциплины;
– приобретение студентами умения самостоятельно осваивать принципиально новые для них знания.

Контроль и самоконтроль знаний, полученных в ходе самостоятельного изучения дисциплины, осуществляется ответами на вопросы, предлагаемые по каждой теме; решением текущих задач и заданий; выполнением контрольных и домашних работ.

Методические материалы содержат перечень контрольных вопросов по дисциплине. В процессе самостоятельной работы студентам необходимо ответить на предложенные вопросы по каждой теме. При подготовке к ответу на данные контрольные вопросы рекомендуется опираться на литературные источники и интернет-ресурсы.

Тема “Жизненный цикл программного обеспечения”

Жизненный цикл программного обеспечения. Модели жизненного цикла. Каскадная модель жизненного цикла ПО и ее принципиальные особенности. Модель с перекрытиями. Итерационная и контролируемая итерационная модели. Спиральная модель. Сравнение моделей. Стадии и рабочие процессы жизненного цикла. Организационные мероприятия.

Программное обеспечение поддержки жизненного цикла. Классификация CASE. Понятия методологии, метода, нотации и средства CASE. Репозиторий и артефакты процесса разработки. Прототипы и их классификация. Возвратное проектирование. Реинжиниринг. Шаблоны проектирования, цели, правила применения. Рефакторинг, цели, правила применения. Каталог рефакторингов, классификация, примеры применения. Обзор современных методологий и CASE.

При изучении этой темы следует уделить особое внимание изучению моделей жизненного цикла и задач программной поддержки. Важно понимание основных функций CASE-систем и методов реинжиниринга и рефакторинга. Самостоятельное изучение данных вопросов желательно начинать со знакомства с Интернет сайтами наиболее известных фирм-производителей данной категории программного обеспечения.

Рекомендуется самостоятельно проработать наиболее важные шаблоны проектирования и попрактиковаться в применении простейших рефакторингов для программ, разработанных студентами в курсе “Информатика и программирование”. Для решения этих задач следует использовать существующие каталоги рефакторингов (М. Фаулера) и шаблонов проектирования (Э. Гаммы и GRASP).

Перечень контрольных вопросов.

1. Жизненный цикл программного обеспечения. Основные стадии и рабочие продукты.
2. Каскадная модель жизненного цикла ПО и ее недостатки. Каскадная модель с

перекрытиями.

3. Сравнение итерационной и контролируемой итерационной моделей.
4. Спиральная модель. Роль прототипирования в модели. Прототипы и их классификация.
5. Стадии и рабочие процессы жизненного цикла. Организационные мероприятия.
6. Программное обеспечение поддержки жизненного цикла.
7. Классификация CASE. Понятия методологии, метода, нотации и средства CASE.
8. Репозиторий и артефакты процесса разработки.
9. Возвратное проектирование.
10. Реинжиниринг. Стадии обратного и прямого инжиниринга. Редокументирование.
11. Обзор каталогов шаблонов проектирования и рефакторингов.

Тема "Анализ и проектирование программного обеспечения"

Общая характеристика этапов анализа, проектирования и реализации программного обеспечения. Анализ и проектирование программного обеспечения на основе структурного подхода. Сущность структурного подхода к разработке программного обеспечения. Моделирование потоков данных (процессов). Диаграммы потоков данных. Контекстная диаграмма и ее детализация. Моделирование данных. Варианты спецификации процессов. Диаграммы Костантайна.

Анализ и проектирование программного обеспечения на основе объектно-ориентированного подхода. Сущность объектно-ориентированного подхода. Методология OMT. Методика Шлеера-Меллора. Информационное моделирование, модели состояний и процессов.

Унифицированный язык моделирования UML. Элементы моделей, основные отношения. Механизмы расширения UML. Варианты использования. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов. Моделирование поведения системы. Диаграммы взаимодействия (диаграммы последовательности и кооперативные диаграммы). Диаграммы состояний. Диаграммы действий. Моделирование реализации системы. Диаграммы компонентов. Диаграммы размещения. Статический и динамический виды. Генерация кода программ и описаний баз данных.

Данная тема является ключевой темой курса и обеспечивает знания и навыки в области анализа и проектирования программного обеспечения.

Методики структурного подхода изучаются в рамках основного курса только на ознакомительном уровне, поэтому полное освоение материала основано на самостоятельной работе. Важнейшими диаграммами являются диаграммы потоков данных действий, активно используемые при реинжиниринге информационных систем, и диаграммы "сущность-связь", изучение которых будет продолжено в дисциплине "Проектирование информационных систем".

Объектный подход и унифицированный язык моделирования UML являются ключевыми темами первой лабораторной работы и подробно рассматриваются в лекционном курсе. Особое внимание следует уделить видам связей между элементами диаграмм классов. На самостоятельное изучение оставлены такие механизмы расширения, как стереотипы, ограничения и теговые значения. Рекомендуется при изучении ограничений самостоятельно ознакомиться с объектным языком ограничений OCL и ввести в создаваемые лабораторных работах модели соответствующие элементы. С целью более глубокого изучения объектного подхода полезно подробнее познакомиться с альтернативными методиками объектно-ориентированного анализа и проектирования – OMT и методикой Шлеера-Меллора.

Перечень контрольных вопросов.

1. Анализ и проектирование программного обеспечения на основе структурного подхода.
2. Диаграммы потоков данных.
3. Моделирование данных.
4. Диаграммы Костантайна.

5. Методология OMT.
6. Методика Шлеера-Меллора.
7. Унифицированный язык моделирования UML. Элементы моделей, основные отношения. Механизмы расширения UML.
8. Статические диаграммы. Классов, объектов, пакетов.
9. Динамические диаграммы. Диаграммы действий. Диаграммы взаимодействия. Диаграммы состояний.
10. Диаграммы для физического представления системы. Диаграммы компонентов и диаграммы размещения.

Тема “Метрология программного обеспечения"

Меры, метрики и индикаторы. Метрики в проектах: метрики процессов и улучшение процессов программного обеспечения, метрики проектов. Метрики для измерения программ. Программное обеспечение для измерения и контроля метрик.

Метрическая теория программ. Объемные метрики. Меры Холстеда. Длина программы, объемы программы, уровень программы, интеллектуальное содержание программы, работа программиста, уровень языка программирования, ожидаемое число ошибок в программе.

Топологические меры сложности. Сложность управляющего графа и цикломатическое число Мак-Кейба. Интервальная и топологическая меры сложности Хансена. Узловая мера.

Меры сложности, основанные на концепции информационных потоков между компонентами системы. Меры Хенри, Кафуры, Овиедо и Тая. Комплексные меры.

Архитектурные меры. Модульная мера Мак-Клура. Мера стабильности плана модуля. Цикломатическая сложность архитектуры системы

Введение в объектно-ориентированные метрики. Сложность классов, сложность взаимодействия классов, сложность иерархии наследования.

Свойства и характеристики качества программного обеспечения. Показатели качества. Конструктивные и функциональные критерии. Характеристики качества и их влияние на различных этапах жизненного цикла программного продукта. Методики оценки показателей качества программного обеспечения в нашей стране и за рубежом.

Современная программная индустрия накопила большую коллекцию моделей и метрик, оценивающих отдельные производственные и эксплуатационные свойства программного обеспечения. Тем не менее, анализ технологического опыта лидеров производства программного обеспечения показывает, насколько дорого обходится несовершенство прогноза разрешимости и трудозатрат, сложности программ, негибкость контроля и управления их разработкой и многое другое, указывающее на отсутствие сквозной методической поддержки и приводящее в конечном итоге к его несоответствию требованиям пользователя или требуемому стандарту. Эти обстоятельства требуют тщательного отбора методик, моделей, методов оценки качества ПО, учета ограничений их пригодности для различных жизненных циклов и в пределах жизненного цикла, установления порядка их совместного использования, применения избыточного разномодельного исследования одних и тех же показателей для повышения достоверности текущих оценок, накопления и интеграции разнородной метрической информации для принятия своевременных производственных решений и заключительной сертификации продукции.

Этот факт существенно усложняет организацию самостоятельной работы студентов по данной теме. В целях сокращения изучаемого материала для практической проработки на лабораторных занятиях выбраны классические объемные метрики Холстеда и ряд топологических метрик. Подробное изучение остальных подходов должно производить самостоятельно. От студентов требуется понимание способа определения соответствующей метрики, качественное сравнение метрик в рамках одной группы и способность обоснованно сформировать набор метрик для анализа конкретного вида программного обеспечения. Самостоятельно можно также опробовать программы для измерения метрик.

Перечень контрольных вопросов.

1. Метрики в проектах: метрики процессов и улучшение процессов программного обеспечения, метрики проектов. Роль менеджера в измерении и анализе метрик.
2. Программное обеспечение для измерения и контроля метрик.
3. Меры Холстеда - анализ применимости.
4. Сравнение топологических мер сложности.
5. Меры сложности, основанные на концепции информационных потоков между компонентами системы.
6. Особенности архитектурных мер для анализа многомодульных программ.
7. Обзор объектно-ориентированных метрик.
8. Показатели качества. Характеристики качества и их влияние на различных этапах жизненного цикла программного продукта.

Тема "Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения"

Стандартизация и сертификация в разработке программного обеспечения. Стандартизация информационных технологий; действующие стандарты и проблемы программных интерфейсов. Международные (ISO), национальные (ГОСТ, ANSI), и отраслевые стандарты (ЕСПД, CMM) и организации (OMG, SEI). Центры сертификации, сертификация персонала. Международные и отечественные стандарты, регламентирующие жизненный цикл программного обеспечения. Стандарт ISO/IEC 12207 и его практическое применение. Стандарты серии ISO 9000 и сложности их применения при стандартизации процесса разработки программного обеспечения. ГОСТы "Качество программных средств" и "Оценка качества программных средств".

Модель зрелости процесса разработки программного обеспечения CMM. Уровни зрелости – начальный повторяемый, определенный, повторяемый, оптимизирующий. Ключевые области обследования. Современное состояние и перспективы сертификации по CMM.

Стандарты по управлению жизненным циклом. Отраслевые стандарты для процесса разработки программного обеспечения. RUP - унифицированный процесс разработки объектно-ориентированного программного обеспечения с использованием UML. Модель жизненного цикла. Основные рабочие процессы, стадии. Заинтересованные лица и роли исполнителей проекта. Артефакты. Планирование итераций.

Обзор рабочих процессов RUP. Бизнес – моделирование. Понятие бизнес-процесса. Формализация бизнес-процесса. Нотации для описания бизнес-процессов. Анализ требований. Прецеденты использования и их детализация. Анализ и проектирование. Разработка и сопровождение. Поддерживающие рабочие процессы. Рекомендации по внедрению.

Рабочие процессы для малых проектов. Экстремальное программирование (XP). Особенности организации работ. Взаимодействие с заказчиком. Роль шаблонов проектирования, рефакторингов и модульного тестирования.

Данная тема является настолько обширной, что ее полноценное изучение возможно только при эффективной и продолжительной самостоятельной работе. Знакомство студентов с государственными стандартами в области информационных технологий начинается с курсовой работы по дисциплине "Информатика и программирование". Оформление пояснительной записки и отчетов по лабораторным работам также должно было производиться в соответствии с ГОСТами. В рамках данной дисциплины в первую очередь требуется знакомство студентов с ГОСТами "Качество программных средств" и "Оценка качества программных средств".

Отраслевые стандарты, как правило, поддерживаются коммерческими продуктами. Особенно полезным в данной области является знакомство с унифицированным процессом разработки программного обеспечения RUP (коммерческий продукт фирмы Rational), который фактически является электронным гипертекстовым учебником. Студенты должны ознакомиться не только с основными рабочими процессами, работами, ролями и артефактами, но и с прилагаемыми шаблонами создаваемых документов. Для

изучения RUP рекомендуется использовать оценочную версию данного продукта. Дополнительно можно ознакомиться с предыдущими версиями продукта или с подробными комментариями специалистов. Существенной трудностью при изучении данного продукта является отсутствие русифицированной версии. Альтернативный подход к организации процесса разработки – экстремальное программирование (XP) используется в основном для малых проектов. Основное внимание при изучении должно быть уделено пониманию методов XP. Все необходимые ссылки приведены в демонстрационных материалах к лекциям, размещенных в ЭУМК дисциплины.

Перечень контрольных вопросов.

1. Международные, национальные, отраслевые стандарты и организации.
2. Роль и место стандартов в процессе разработки программного обеспечения.
3. Стандарты серии ISO 9000.
4. ГОСТ “Качество программных средств”.
5. ГОСТ “Оценка качества программных средств”.
6. Модель зрелости процесса разработки программного обеспечения CMM. Уровни зрелости. Ключевые области обследования.
7. Унифицированный процесс разработки объектно-ориентированного программного обеспечения с использованием UML (RUP).
8. Рабочие процессы, работы, роли, артефакты и шаблоны RUP.
9. Рабочие процессы для малых проектов. Экстремальное программирование (XP).

2. Методические рекомендации по подготовке к практическим (подгрупповым) занятиям

Для выполнения заданий рекомендуется использовать программный продукт MS Visio. Возможно также применение более мощных программных средств визуального моделирования, с поддержкой нотации UML версии не ниже 1.3. Их применение позволяет в отличие от MS Visio выполнять процессы реинжиниринга и кодогенерации для наиболее распространенных языков программирования.

В работе требуется разработать систему обработки заказов. При получении заказа заполняется форма заказа, которая поступает в магазин и в бухгалтерию. Система должна обеспечивать возможность добавления новых заказов, изменения старых, выполнения заказов, проверки и возобновления инвентарных описей. При получении заказа система должна также послать сообщение бухгалтерской системе, которая выписывает счет. Если требуемого товара нет на складе, заказ должен быть отклонен.

Самостоятельно требуется дополнить разработанную в лабораторной работе модель элементами в соответствии с индивидуальным заданием.

Варианты индивидуальных заданий для самостоятельной работы:

1. Документирование модели. Дополните элементы диаграммы прецедентов тэговыми значениями, документацией и ограничениями.
2. Документирование модели. Дополните элементы последней диаграммы последовательности тэговыми значениями, документацией и ограничениями с использованием языка OCL.
3. Документирование модели. Дополните элементы диаграммы классов тэговыми значениями, документацией и ограничениями с использованием языка OCL.
4. Проектирование новых прецедентов. Выполните анализ и проектирование прецедента "Изменить существующий заказ".
5. Проектирование новых прецедентов. Выполните анализ и проектирование прецедента "Оформить заказ".
6. Проектирование новых прецедентов. Выполните анализ и проектирование прецедента "Отклонить заказ".
7. Проектирование новых прецедентов. Выполните анализ и проектирование прецедента "Обновить инвентарную опись".
8. Детализация альтернативных потоков событий. Детализируйте прецедент "Ввести новый заказ" для случая, когда товара нет.

9. Детализация альтернативных потоков событий. Детализируйте прецедент "Ввести новый заказ" для случая, когда форма заказа товара заполнена неверно.
10. Проектирование динамики системы. Постройте диаграмму состояний для класса "Менеджер транзакций".
11. Проектирование динамики системы. Постройте диаграмму действий (Activity Diagram) для алгоритма сортировки заказов по дате.
12. Проектирование динамики системы. Постройте диаграмму действий (Activity Diagram) для алгоритма сортировки заказов по дате.
13. Проектирование динамики системы. Постройте диаграмму действий (Activity Diagram) для алгоритма определения максимального номера заказа.
14. Проектирование физической структуры реализации. Нарисуйте диаграмму компонентов (Component Diagram) для реализации разработанных классов.
15. Проектирование физической структуры реализации. Нарисуйте диаграмму размещения (Deployment Diagram) для реализации разработанной системы в сети с выделенными сервером приложений и сервером базы данных.

Работа “Метрический анализ программ”

Для самостоятельной подготовки к выполнению первой части работы необходимо для заданного варианта программы на языке Паскаль разработать вычислительный алгоритм и добиться, чтобы программа была работоспособна и давала корректные результаты. В дальнейших расчетах важен только вычислительный алгоритм, реализуемый программой. Поэтому для получения более корректных оценок характеристик программ следует учитывать только вычислительные операторы и исключить операторы описания переменных, а также, операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений. Файл отчета формируется в одной книге программы MS Excel. В книгу вставляются текст программы (в виде надписи), необходимые пояснения и рисунки.

В соответствии с индивидуальным заданием составить формулы для вычисления и рассчитать дополнительные метрики.

Варианты индивидуальных заданий для самостоятельной работы:

1. Уровень качества программирования, интеллектуальное содержание, трудоемкость понимания готовой программы, время программирования.
2. Оценка уровня качества программирования, интеллектуальное содержание, трудоемкость кодирования, уровень используемого языка программирования, ожидаемое число ошибок в программе.
3. Интеллектуальное содержание, время программирования, трудоемкость кодирования, ожидаемое число ошибок в программе.
4. Интеллектуальное содержание, уровень используемого языка программирования, ожидаемое число ошибок в программе.
5. Оценка уровня качества программирования, интеллектуальное содержание, трудоемкость понимания готовой программы, время программирования, ожидаемое число ошибок в программе.
6. Уровень качества программирования, интеллектуальное содержание, время программирования, ожидаемое число ошибок в программе.
7. Интеллектуальное содержание, трудоемкость кодирования, трудоемкость понимания готовой программы, ожидаемое число ошибок в программе.
8. Уровень качества программирования, интеллектуальное содержание, время программирования, уровень используемого языка программирования, ожидаемое число ошибок в программе.
9. Оценка уровня качества программирования, интеллектуальное содержание, трудоемкость кодирования, трудоемкость понимания готовой программы, ожидаемое число ошибок в программе.
10. Оценка уровня качества программирования, интеллектуальное содержание, трудоемкость кодирования, время программирования, уровень используемого языка программирования, ожидаемое число ошибок в программе.

Самостоятельная подготовка к выполнению второй части работы состоит в построении с помощью программы MS Visio диаграммы действий (activity diagram), представляющей вычислительный алгоритм программы на языке программирования Паскаль. Выбранная программа должна иметь достаточно сложную логическую структуру – иметь вложенные циклы и ряд условных операторов.

3. Методические рекомендации по написанию контрольных работ

Важнейшей формой учебной отчетности студента является **контрольная работа**.

Выполнение контрольной работы является промежуточной формой отчетности по изучаемой дисциплине и преследует цель лишь оценить способность студента к самостоятельному поиску источников, формированию содержания и его письменного изложения по указанной проблеме. Это важная составляющая изучения дисциплины, а также эффективная форма контроля знаний. При заочном обучении она выступает как обязательная, основная форма самостоятельной работы. В работе (в соответствии с учебным планом) студент обязан самостоятельно глубоко разобраться в изучаемых проблемах, усвоить суть темы, уяснить ее содержание и только затем письменно представить свою отчетную работу.

Выполнение контрольной работы является одним из условий допуска студента к сдаче экзамена. Работа должна соответствовать установленным требованиям, то есть в ней должны быть раскрыты все проблемы, определенные темой. Для этого студент обязан самостоятельно проанализировать первоисточники и дать исчерпывающие ответы на вопросы темы. Контрольная работа — серьезное учебное задание, и чтобы написать ее как следует, необходимо использовать те первоисточники и учебные пособия, которые позволяют полнее разобраться в проблеме. Студент должен регулярно работать в университетской и городской библиотеке, вдумчиво конспектировать лекции преподавателей.

При написании контрольной работы следует обращать особое внимание на грамотное использование терминологии. При употреблении впервые тех или иных терминов и понятий следует давать их определения либо в самом тексте, либо в сносках.

Приступая к контрольной работе, требуется сначала ознакомиться с имеющейся литературой по теме, изучить первоисточники и составить план. Здесь, в отличие от курсовой работы, план предполагает рассмотрение одной, причем довольно широкой, проблемы, и он может состоять из двух-трех вопросов. Минимальное количество первоисточников, привлекаемых для написания курсовой работы — пять наименований.

Как правило, контрольные работы по дисциплине сугубо индивидуальны, то есть их тематика персонифицирована. Однако в отдельных случаях темы контрольных работ могут быть адресованы и сразу нескольким, и группе в целом. Таким приемом преподаватель выявляет степень усвоения какой-то важной учебной проблемы и определяет необходимость проведения дополнительных занятий по какой-либо теме. В настоящее время широко используется методика компьютерного тестирования знаний студентов по дисциплинам, в результате чего появляется возможность быстро проверять знания по наиболее важными темами и объективно оценивать их. Эта форма также может выступать как вид контрольной работы.

В качестве контрольной работы широко применяется самостоятельное изучение монографического исследования по конкретной, крайне важной проблеме, требующей глубокого рассмотрения. Этот вид работы предполагает не простое знакомство с определенным монографическим исследованием, а детальное его изучение. Для этого студенту важно знать некоторые правила работы с первоисточником, которым для него будет являться монография. Следует выяснить фамилию автора, его имя и отчество, ученую степень и звание, а также что побудило его взяться за изучение данной проблемы; обратить внимание на основные вопросы монографии и их разрешение автором, уметь раскрывать их в ходе собеседования с преподавателем.

Студенту следует письменно (предельно кратко) очертить те вопросы (полностью или частично), которые поставлены автором в монографическом исследовании; при изложении их следует указывать страницы источника.

Задания для написания контрольных работ (для заочной формы обучения)

Целями выполнения контрольных работ по учебной дисциплине являются расширение и углубление знаний по определенным темам курса, а также демонстрация навыков решения теоретических и практических проблем. Контрольная работа содействует развитию творческого отношения к учебной деятельности, повышает ее эффективность, а также активизирует усилия студента по освоению изучаемого предмета.

Дисциплина преподается студентам 2 курса в третьем семестре, она является логическим продолжением дисциплины “Информатика и программирование” и обеспечивает знания и навыки в области разработки и стандартизации программного обеспечения и информационных технологий.

Методические рекомендации содержат перечень вариантов заданий для выполнения контрольной работы по дисциплине. Каждый вариант содержит теоретический вопрос и практическое задание. Ответ на вопрос оформляется в форме мини реферата объемом не более 5 страниц. Результаты выполнения задания представляются соответствующими иллюстрациями, расчетами и формулами. Выбор варианта задания осуществляется преподавателем.

Вариант №1

- *Перечислите модели жизненного цикла программного обеспечения.*
- *Практическое задание:*

Требуется разработать с использованием UML модель сценариев взаимодействия (**диаграмма прецедентов использования и диаграммы последовательности**) с программным обеспечением банкомата. В его состав банкомата входят следующие устройства:

– дисплей;
– панель управления с кнопками;
– приемник кредитных карт;
– хранилище денег и лоток для их выдачи;
– принтер для печати справок.

Банкомат подключен к линии связи для обмена данных с компьютером банка, хранящим сведения о счетах клиентов. После распознавания типа пластиковой карточки, банкомат выдает на дисплей приглашение ввести персональный код. Затем банкомат проверяет правильность введенного кода и предлагает выбрать операцию. Клиент может либо снять наличные со счета, либо узнать остаток на его счету. После выбора клиентом суммы банкомат запрашивает, нужно ли печатать справку по операции. Затем банкомат посылает запрос на снятие выбранной суммы компьютеру банка. В случае получения разрешения на операцию, банкомат проверяет, имеется ли требуемая сумма в его хранилище денег. Если он может выдать деньги, то на дисплей выводится сообщение «Выньте карту». После удаления карточки из приемника, банкомат выдает указанную сумму в лоток выдачи. Затем банкомат печатает справку по произведенной операции, если она была затребована клиентом.

Вариант №2

- *Какие программы для поддержки жизненного цикла программного обеспечения Вы знаете?*
- *Практическое задание:*

Требуется разработать с использованием UML информационную модель (**диаграммы пакетов и классов**) программного обеспечения банкомата. Банкомат - это автомат для выдачи наличных денег по кредитным пластиковым карточкам. В его состав входят следующие устройства:

– дисплей;

– панель управления с кнопками;
– приемник кредитных карт;
– хранилище денег и лоток для их выдачи;
– принтер для печати справок.

Банкомат подключен к линии связи для обмена данных с компьютером банка, хранящим сведения о счетах клиентов. После распознавания типа пластиковой карточки, банкомат выдает на дисплей приглашение ввести персональный код. Затем банкомат проверяет правильность введенного кода и предлагает выбрать операцию. Клиент может либо снять наличные со счета, либо узнать остаток на его счету. После выбора клиентом суммы банкомат запрашивает, нужно ли печатать справку по операции. Затем банкомат посылает запрос на снятие выбранной суммы компьютеру банка. В случае получения разрешения на операцию, банкомат проверяет, имеется ли требуемая сумма в его хранилище денег. Если он может выдать деньги, то на дисплей выводится сообщение «Выньте карту». После удаления карточки из приемника, банкомат выдает указанную сумму в лоток выдачи. Затем банкомат печатает справку по произведенной операции, если она была затребована клиентом.

Вариант №3

- *Что такое репозиторий и артефакты процесса разработки, реинжиниринг и рефакторинг программного обеспечения?*
- *Практическое задание:*

Требуется разработать с использованием UML модель системы поддержки заказа и учета товаров в магазина. Для каждого товара фиксируется место хранения (определенная полка), количество товара и его поставщик. Система поддержки заказа и учета товаров должна обеспечивать добавление информации о новом товаре, изменение или удаление информации об имеющемся товаре, хранение (добавление, изменение и удаление) информации о поставщиках, включающей в себя название фирмы, ее адрес и телефон. При помощи системы составляются заказы поставщикам. Каждый заказ может содержать несколько позиций, в каждой позиции указываются наименование товара и его количество в заказе. Система учета по требованию пользователя формирует следующую справочную информацию:

– список всех товаров;
– список товаров, имеющихся в наличии;
– список товаров, количество которых необходимо пополнить;
– список товаров, поставляемых данным поставщиком.

Вариант №4

- *Приведите основные объемные метрики Холстеда.*
- *Практическое задание:*

Требуется разработать с использованием UML модель системы автоматизирующей деятельность библиотеки. Система поддержки управления библиотекой должна обеспечивать добавление информации о читателях в регистрационный список, редактирование этой информации и удаление ее. В регистрационном списке хранятся фамилия, имя и отчество читателя; номер его читательского билета и дата выдачи билета. В каталоге библиотеки хранится информация о книгах: название, список авторов, библиотечный шифр, год и место издания, название издательства, общее количество экземпляров книги в библиотеке и количество экземпляров, доступных в текущий момент. Система обеспечивает поиск книг в каталоге на основании введенного шифра или названия книги. В системе осуществляется регистрация взятых и возвращенных читателем книг. Про каждую выданную книгу хранится запись о том, кому и когда была выдана книга, и когда она будет возвращена. При возврате книги в записи делается соответствующая пометка, а сама запись не удаляется из системы. Система также должна выдавать следующую информацию:

– какие книги были выданы за данный промежуток времени;
– какие книги были возвращены за данный промежуток времени;
– какие книги находятся у данного читателя;

– имеется ли в наличии некоторая книга.

Вариант №5

- *Назовите основные подходы к анализу топологической сложности программ.*
- *Практическое задание:*

Требуется разработать с использованием UML модель программного обеспечения Интернет-магазина. Интернет-магазин позволяет делать покупки с доставкой на дом. Клиентам магазина имеют доступ к каталогу продаваемых товаров. В каталоге товары распределены по разделам. О каждом товаре доступна полная информация (название, вес, цена, изображение, дата изготовления и срок годности) Предусмотрена система поиска товаров в каталоге. Заполнение каталога информацией происходит автоматически в начале рабочего дня, информация берется из системы автоматизации торговли. При отборе клиентами товаров поддерживается виртуальная «торговая корзина». Любое наименование товара может быть добавлено в «корзину» или изъято с последующим пересчетом общей стоимости покупки. Текущее содержимое «корзины» постоянно показывается клиенту.По окончании выбора товаров производится оформление заказа и регистрация покупателя. Клиент указывает в регистрационной форме свою фамилию, имя и отчество, адрес доставки заказа и телефон, по которому с ним можно связаться для подтверждения сделанного заказа. Заказы передаются для обработки в систему автоматизации торговли. Проверка наличия товаров на складе и их резервирование Интернет-магазином не производятся.

Вариант №6

- *Опишите особенности статических диаграмм UML (классов, объектов, пакетов).*
- *Практическое задание:*

Для заданного варианта программы на языке Паскаль составить таблицу (в MS Excel), содержащую все простые операторы программы и число вхождений каждого из них в программу. Вычислить число простых операторов и общее число всех операторов. Аналогично составить таблицу, содержащую все простые операнды программы и число вхождений каждого из них в программу. Вычислить число простых операндов, и общее число всех операндов. Следует учитывать только вычислительные операторы и исключить операторы описания переменных, а также, операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений.

Составить формулы для вычисления и рассчитать следующие метрические характеристики (по Холстеду):

- длину программы;
- реальный и потенциальный объем программы;
- теоретическую длину программы;
- уровень качества программирования, интеллектуальное содержание, трудоемкость понимания готовой программы, время программирования.

Анализируемая программа на языке Паскаль:

```
program gun;
uses Crt;
var
      v0, alpha, t,
      dt, x, y, x0, y0,
      vx0, vy0, tc          : Extended;
      N, i                  : Integer;
const
      g = 9.81;
begin
 ClrScr;
 WriteLn('Введите начальную скорость');
 ReadLn(v0);
 WriteLn;
 WriteLn('Введите наклон траектории');
 ReadLn(alpha);
```

```
 WriteLn;
 WriteLn('Введите число точек');
 ReadLn(N);
 vx0 := v0 * Cos(alpha);
 vy0 := v0 * Sin(alpha);
 x0 := 0;
 y0 := 0;
 T := 2 * vy0 / g;
 dt := T / (N - 1);
 i := 1;
 tc := 0;
while i <= N do
 begin
      x := x0 + vx0 * tc;
      y := y0 + vy0 * tc - g * Sqr(tc) / 2;
      WriteLn(x,' ',y);
      Inc(i);
      tc := tc + dt;
      if i mod 20 = 0 then
      begin
           WriteLn('Нажмите <Enter>');
           ReadLn;
      end;
 end;
 WriteLn('Нажмите <Enter>');
 ReadLn;
end.
```

Вариант №7

- *В чем отличия динамических диаграмм UML (последовательности, сотрудничества, состояний, действий).*
- *Практическое задание:*

Для заданного варианта программы на языке Паскаль построить управляющий граф, представляющий вычислительный алгоритм программы. Вычислить следующие топологические меры сложности:

- цикломатическое число Мак-Кейба;
- структурную сложность программы по критерию минимального покрытия дуг;
- структурную сложность программы по критерию базовых маршрутов.

Анализируемая программа на языке Паскаль:

```
program diophantine_equation_2;
var
      x, y, z, w, n: LongInt;
begin
      n := MaxLongint - 63;
      n := Trunc(Sqrt(n));
      n:=n-8;
      x := 0;
      WriteLn('Все целые решения уравнения x^3 = y^2 + 63,');
      WriteLn('для 1 <= y <= ', n, ':');
      for  y := 1  to  n  do
      begin
           z := y*y + 63;
           repeat
                Inc(x);
                w := x*x*x;
           until  w >= z;
```

```
            if  w = z  then
            WriteLn('(x, y) = (', x, ', ', y, ')')
            else  Dec(x);
        end;
        Write('Работа закончена, нажмите <Enter>:');
        ReadLn;
end.
```

Вариант №8

- *Опишите основные принципы и нотации структурного подхода.*
- *Практическое задание:*

Для заданного варианта программы на языке Паскаль составить таблицу (в MS Excel), содержащую все простые операторы программы и число вхождений каждого из них в программу. Вычислить число простых операторов и общее число всех операторов. Аналогично составить таблицу, содержащую все простые операнды программы и число вхождений каждого из них в программу. Вычислить число простых операндов, и общее число всех операндов. Следует учитывать только вычислительные операторы и исключить операторы описания переменных, а также, операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений.

Составить формулы для вычисления и рассчитать следующие метрические характеристики (по Холстеду):

- длину программы;
- реальный и потенциальный объем программы;
- теоретическую длину программы;
- интеллектуальное содержание, трудоемкость кодирования, трудоемкость понимания готовой программы, ожидаемое число ошибок в программе.

Анализируемая программа на языке Паскаль:

```
program maximum_search;

const
        max = 10000;

var
        random_array : array[1..max] of Integer;
        max_element, index, index_max : Integer;

begin
 Randomize;
 for  index := 1  to  max  do
        random_array[index] :=  Random(10000);

 max_element := random_array[1];
 for  index := 2  to  max  do
        if max_element < random_array[index] then
        begin
                max_element := random_array[index];
                index_max := index;
        end;
 WriteLn('Значение ', index_max, '-го, максимального    элемента равно ',
random_array[index_max]);
 WriteLn('Для завершения работы нажмите <Enter>');
 ReadLn;
end.
```

Вариант №9

- *Перечислите основные элементы и виды отношений, используемые на диаграммах UML.*

- *Практическое задание:*

Для заданного варианта программы на языке Паскаль построить управляющий граф, представляющий вычислительный алгоритм программы. Вычислить следующие топологические меры сложности:

- цикломатическое число Мак-Кейба;
- структурную сложность программы по критерию минимального покрытия дуг;
- структурную сложность программы по критерию базовых маршрутов.

Анализируемая программа на языке Паскаль:

```
program eratosphen1;

uses crt;

const max = 1000;

var
    b, j, k : word;
    flag : array[1..max] of boolean;

begin
    clrscr;
    flag[1] := false;
    for j := 2 to max do
        flag[j] := true;
    b := trunc(sqrt(max));
    k := 0;
    while k <= b do
    begin
        repeat
            inc(k)
        until flag[k];
        j := 2 * k;
        while j <= max  do
        begin
            flag[j] := false;
            j := j + k;
        end;
    end;
    for j := 1 to max do
        if flag[j] then
            write(j:8);
    writeln;
    write('Нажмите <Enter>');
    readln;
end.
```

Вариант №10

- *Дайте сравнительную характеристику стандартизованным процессам разработки программного обеспечения Rational Unified Proccess (RUP) и экстремального программирования (XP).*
- *Практическое задание:*

Для заданного варианта программы на языке Паскаль составить таблицу (в MS Excel), содержащую все простые операторы программы и число вхождений каждого из них в программу. Вычислить число простых операторов и общее число всех операторов. Аналогично составить таблицу, содержащую все простые операнды программы и число вхождений каждого из них в программу. Вычислить число простых операндов, и общее

число всех операндов. Следует учитывать только вычислительные операторы и исключить операторы описания переменных, а также, операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений.

Составить формулы для вычисления и рассчитать следующие метрические характеристики (по Холстеду):

- длину программы;
- реальный и потенциальный объем программы;
- теоретическую длину программы;
- оценку уровня качества программирования, интеллектуальное содержание, трудоемкость кодирования, уровень используемого языка программирования, ожидаемое число ошибок в программе.

Анализируемая программа на языке Паскаль:

```
program unit_roundoff_error;

var
      s, u: real;
      k: word;

begin
      u := 1.0;
      while  1.0 + u > 1.0  do  u := 0.1 * u;
      s := u;
      for  k := 1  to  3  do
      begin
            while  1.0 + u <= 1.0  do  u := u + s;
            u := u - s;
            s := 0.1*s;
      end;
      writeln('Ошибка округления единицы: ', u:14);
      write('Нажмите <Enter>: ');
      readln;
end.
```

4. Методические рекомендации по написанию курсовой работы

Курсовая работа учебным планом не предусмотрена.

ОЦЕНОЧНЫЕ И МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ

Оценочные и методические материалы включают в себя:
- перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы;
- показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания;
- типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций в процессе освоения образовательной программы;
- методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций.

1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы

№ п\п	Контролируемые темы дисциплины	Код формируемой компетенции	Код и наименование индикатора достижения	Наименование оценочного средства
	Жизненный цикл программного обеспечения	ОПК-7, ОПК-9	ОПК-7.1. Знает принципы сбора, отбора и обобщения информации. ОПК-9.1. Знает принципы организации проектирования и содержание этапов процесса разработки программных комплексов дисциплины, задачи и методы исследования и обеспечения качества и надежности программных компонентов, основы разработки программных комплексов.	Опрос, участие в коллоквиуме, выполнение практических заданий, кейсы
	Анализ и проектирование программного обеспечения	ОПК-7, ОПК-9	ОПК-7.1. Знает принципы сбора, отбора и обобщения информации. ОПК-7.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его	Подготовка сообщений, выполнение практических заданий, тестирование

			качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами. ОПК-7.3. Владеет навыками программирования, отладки и тестирования программного обеспечения. ОПК-9.1. Знает принципы организации проектирования и содержание этапов процесса разработки программных комплексов дисциплины, задачи и методы исследования и обеспечения качества и надежности программных компонентов, основы разработки программных комплексов. ОПК-9.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами. ОПК-9.3. Владеет навыками работы с инструментальными средствами	

			моделирования предметной области, прикладных и информационных процессов, использования технологических стандартов информационных систем и программных продуктов.	
	Метрология программного обеспечения	ОПК-7, ОПК-9	ОПК-7.1. Знает принципы сбора, отбора и обобщения информации. ОПК-7.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами. ОПК-7.3. Владеет навыками программирования, отладки и тестирования программного обеспечения. ОПК-9.1. Знает принципы организации проектирования и содержание этапов процесса разработки программных комплексов дисциплины, задачи и методы исследования и обеспечения качества и надежности программных компонентов,	Опрос, тестирование, коллоквиум, кейсовые задания

			основы разработки программных комплексов. ОПК-9.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами. ОПК-9.3. Владеет навыками работы с инструментальными средствами моделирования предметной области, прикладных и информационных процессов, использования технологических стандартов информационных систем и программных продуктов.	
	Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения	ОПК-7, ОПК-9	ОПК-7.1. Знает принципы сбора, отбора и обобщения информации. ОПК-7.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами.	Опрос, тестирование, коллоквиум, практические задания

			ОПК-7.3. Владеет навыками программирования, отладки и тестирования программного обеспечения. ОПК-9.1. Знает принципы организации проектирования и содержание этапов процесса разработки программных комплексов дисциплины, задачи и методы исследования и обеспечения качества и надежности программных компонентов, основы разработки программных комплексов. ОПК-9.2. Умеет проводить анализ и проектирование современного программного обеспечения, определять его качественные показатели, организовывать процесс разработки и вести документацию в соответствии с современными стандартами. ОПК-9.3. Владеет навыками работы с инструментальными средствами моделирования предметной области, прикладных и информационных процессов, использования технологических стандартов информационных	

			систем и программных продуктов.	
Результат достижения планируемых результатов изучения дисциплины				зачет

2. Описание показателей и критериев оценивания компетенций, шкал оценивания

Критерии оценивания (текущий контроль)

1. Оценка **«отлично»** выставляется студенту, если студент имеет глубокие знания учебного материала по теме практического задания, в логической последовательности излагает материал; смог ответить на все уточняющие и дополнительные вопросы;
2. Оценка **«хорошо»** выставляется, если студент показал знание учебного материала, смог ответить почти полностью на все заданные дополнительные и уточняющие вопросы;
3. Оценка **«удовлетворительно»** выставляется, если студент в целом освоил материал; однако, ответил не на все уточняющие и дополнительные вопросы;
4. Оценка **«неудовлетворительно»** выставляется студенту, если он имеет существенные пробелы в знаниях основного учебного материала по теме практического задания, который полностью не раскрыл содержание вопросов, не смог ответить на уточняющие и дополнительные вопросы.

Критерии оценивания (зачет)

Знания, умения, навыки и компетенции студентов оцениваются следующими оценками: «зачет», «незачет».

- **«зачет»** - студент хорошо и прочно усвоил весь программный материал, исчерпывающе, последовательно, грамотно и логически стройно его излагает, увязывает с практикой, свободно справляется с решением ситуационных задач и тестовыми заданиями, правильно обосновывает принятие решений, умеет самостоятельно обобщать программный материал, не допуская ошибок, знает дополнительную литературу по изучаемой дисциплине.
- «**незачет»** - студент не знает значительной части основного программного материала, в ответах допускает существенные ошибки, не владеет умениями и навыками в выполнении тестовых заданий и решении задач, не способен ответить на дополнительные вопросы.

3. Типовые контрольные задания и методические материалы, процедуры оценивания знаний, умений и навыков

ТЕКУЩИЙ КОНТРОЛЬ

Тестовые материалы

Важными в методическом плане на семинарских занятиях являются проводимые тестовые задания, которые содействуют превращению теоретико-правовых знаний в глубокие убеждения, дают простор для развития творческо-эмоциональной сферы, позволяют сделать выводы об эффективности занятий с учащимися, что в итоге повышает интерес к овладению знаниями.

Решение тестовых заданий является важным методическим приемом для

закрепления и осмысления, полученных бакалаврами знаний по изучаемому предмету.

1. Студент тестируемой учебной группы получает 50 тестовых заданий. Для каждого из вопросов тестового задания предусмотрен только один правильный вариант ответа, который должен выбрать студент. Результаты тестирования оцениваются в зависимости от количества неверно выбранных ответов.
2. Итоги тестирования заносятся в ведомость, составляемую на всю учебную группу. Предоставленные сведения должны содержать данные о количестве опрошенных, о количестве отличных, хороших, удовлетворительных и неудовлетворительных оценок.
3. В заключение работы выводиться средний балл итогового контроля знаний студентов.

ПАСПОРТ ТЕСТОВЫХ ЗАДАНИЙ

1. Общее количество тестовых заданий в базе – 161
2. Ограничение времени выполнения теста (в мин) – 30 Автоматическое перемешивание вопросов в тесте: □ (да)
3. Случайный порядок ответов в тестовом задании: □ (да)
4. Критерии оценки результатов тестирования :_свыше 50% правильных ответов – зачет

Пример тестовых заданий для текущего контроля представлен ниже:

1. Сформулируйте понятие жизненного цикла программного средства
   – Последовательность этапов, частных работ и операций, регламентирующих процесс создания и эксплуатации программных средств от подготовки технического задания до окончания эксплуатации
   – Последовательность этапов, частных работ и операций, регламентирующих процесс создания и эксплуатации программных средств от подготовки технического задания до завершения испытаний ряда версий
2. Дайте определение модели жизненного цикла программного средства
   – Каскадный набор процессов по разработке, эксплуатации и сопровождению программного продукта на протяжении всего его жизненного цикла
   – Структура процессов, охватывающая жизнь системы от установления требований к ней до прекращения испытаний различных версий программного продукта
   – Спиральная структура, состоящая из процессов, работ и задач по разработке и эксплуатации информационной системы
   – Модель создания и использования программного обеспечения, начиная с момента возникновения необходимости в нем и заканчивая полным выходом из употребления
3. Объясните смысл каскадной и спиральной модели жизненного цикла программного средства
   – В каскадной модели каждый этап завершается выпуском полного комплекта документации для того, чтобы работа была продолжена на следующем этапе
   – В спиральной модели на этапах анализа и проектирования создаются прототипы (версии), неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь завершения предыдущего
   – В спиральной модели каждый этап завершается выпуском полного комплекта документации для того, чтобы работа была продолжена на следующем этапе
   – В каскадной модели на этапах анализа и проектирования создаются прототипы (версии), неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь завершения предыдущего
4. В чем заключаются главные положительные свойства каскадной модели?
   – На каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности
   – Выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты
   – Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь завершения предыдущего
   – Показать пользователям работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований
5. Охарактеризуйте недостатки каскадной модели
   – Запаздывание с получением результатов
   – Модели автоматизации могут устареть одновременно с их утверждением
   – Не ясны сроки перехода на следующий этап
6. В чем заключается основная проблема спиральной модели?

– Запаздывание с получением результатов
– Модели автоматизации могут устареть одновременно с их утверждением
– Не ясны сроки перехода на следующий этап

7. Какой современный вариант каскадной модели Вы знаете
– Контролируемая каскадная
– Планируемая спиральная
– Ограниченная водопадная
– Каскадная с перекрытиями

8. Какой современный вариант итерационной модели Вы знаете
– Контролируемая итерационная
– Планируемая спиральная
– Ограниченная водопадная
– С перекрытиями

9. Какая модель жизненного цикла используется в RUP
– Контролируемая каскадная
– Планируемая спиральная
– Ограниченная водопадная
– Каскадная с перекрытиями
– Контролируемая итерационная

10. Кольцевая разработка (round-trip design)
– Это синхронизация рабочих продуктов и генерируемого исходного кода
– Модель жизненого цикла
– Может быть релизована с использованием меток в исходнои коде
– Может быть релизована с использованием репозитория

11. Вариант реализации проектируемой системы или ее аспекта, абстрагирующийся от требований к системе называется
– Модель
– Итерация
– Артефакт
– Прототип
– Образец

12. При построении прототипа могут быть изменены требования к
– Надежности
– Языку реализации
– Качеству
– Назначению
– Решаемой задаче
– Интерфейсу
– Функциональности

13. Рефакторинг – это
– Изменения выполняемые во внутренней структуре ПО без изменения функциональности
– Изменения выполняемые во внутренней структуре ПО с изменением функциональности
– Изменение времени разработки с целью масштабирования

14. Почему следует производить рефакторинг?
– Улучшает процесс проектирования ПО
– Добавляет новые функции
– Позволяет находить ошибки
– Делает ПО легче познаваемым
– Улучшает стиль программирования
– Сокращает время разработки

15. Когда следует проводить рефакторинг?
– Когда добавляется новая функция
– Постоянно
– После тестирования
– Когда исправляются ошибки
– При обзоре кода
16. Множество канонических типов диаграмм UML включает в себя
– Диаграммы классов
– Диаграммы состояний
– Диаграммы последовательности
– Диаграммы потоков данных
17. Имя ассоциации
– образуется из имен ассоциированных классов
– образуется из имен ролей ассоциированных классов
– указывается в виде текста над линией ассоциации
18. Если множественность роли ассоциации задана символом * , то это означает, что
– ни одного экземпляра классификатора на не участвует в связях, порождаемых ассоциацией
– по меньшей мере один экземпляр классификатора участвует в связях, порождаемых ассоциацией
– неопределенное количество экземпляров классификатора участвует в связях, порождаемых ассоциацией
19. Что такое KAELOC
– Тысяч эквивалентных строк кода на Ассемблере
– Тысяч эквивалентных строк машинного кода
– Тысяч строк кода после трансляции
– Тысяч строк кода языке программирования
20. В метриках Холстеда что больше длина программы или реальный размер
– Длина
– Размер
– Они равны
21. В метриках Холстеда интеллектуальное содержание программы инвариантно по отношению к используемым языкам реализации
– Да
– Нет
– Зависит от уровня абстрактности языка программирования
22. Число Страуда
– Для обычных людей находится в интервале от -5 до 10
– Для обычных людей находится в интервале от 5 до 100
– Для обычных людей находится в интервале от 0 до 50
– Для обычных людей находится в интервале от 5 до 10
23. В метриках Холстеда формула ожидаемого числа ошибок в программе
– Получена экспериментально
– Доказана теоретически
– Отсутствует
24. Цикломатическое число Мак-Кейба
– Числу компонентов связности графа в не связном графе
– Равно числу линейно независимых контуров в сильносвязном графе
– Числу предикатов в услових компонентов связности графа
25. Критерий минимального покрытия дуг графа требует
– Минимального множества маршрутов программы, охватывающих все последовательности передач управления и учитывающих исполнение программы по

каждому направлению при ветвлении.

– Соответствует однократной проверке каждого линейно-независимого ациклического маршрута и каждого линейно-независимого цикла, в совокупности образующих базовые маршруты

26. Для структурированных программ

– Цикломатическое число на 1 меньше количества вершин, в которых происходит ветвление

– Цикломатическое число на 1 больше количества вершин, в которых происходит ветвление

– Цикломатическое число равно количеству вершин, в которых происходит ветвление

27. В модели зрелости процесса разработки ПО СММ

– Пять уровней зрелости, но сертификация производится только по трем

– Пять уровней зрелости, но сертификация производится только по четырем

– Четыре уровня зрелости, но сертификация производится только по двум

28. В модели зрелости процесса разработки ПО СММ на втором уровне главная фигура от которой зависят сроки и качество проекта – это

– Менеджер проекта

– Руководитель организации

– Заказчик

– Программист

– Тестеровщик

– Нет правильного ответа

29. В модели зрелости процесса разработки ПО СММ на первом уровне главная фигура от которой зависят сроки и качество проекта – это

– Менеджер проекта

– Руководитель организации

– Заказчик

– Программист

– Тестеровщик

– Нет правильного ответа

30. В модели зрелости процесса разработки ПО СММ на третьем уровне главная фигура от которой зависят сроки и качество проекта – это

– Менеджер проекта

– Руководитель организации

– Заказчик

– Программист

– Тестеровщик

– Нет правильного ответа

31. В модели зрелости процесса разработки ПО СММ успешность использования ключевой области на практике наличии систематического и измеримого положительного результата практически во всей организации оценивается

– 5 по 5 бальной системе

– 10 по 10 бальной системе

– 8 по 10 бальной системе

32. К правилам экстремального программирования XP относятся

– Заказчик всегда рядом.

– Весь код должен соответствовать принятому стандарту.

– Весь код должен быть создан парным программированием.

– Частая интеграция кода.

– Оставлять тестирование на потом.

– Коллективное владение кодом.

– Оставлять оптимизацию на потом.

33. К правилам экстремального программирования XP относятся
- Любой код должен иметь Unit Test
- Все Unit тесты должны проходить перед отдачей
- Если найдена ошибка, то тесты корректируются или но не создаются
- Функциональные тесты периодически выполняются и их результаты публикуются

34. Укажите основные уровни стандартизации программных средств
- Международный
- Региональный
- Национальный
- Внутрифирменный
- Внутрипрограммный
- Внутрипроцедурный

35. Как определяется понятие «стандарт» в области программного обеспечения?
- Консенсус по спецификации, производству и использованию аппаратных и программных средств вычислительной техники
- Соглашение по установке и применению норм и правил взаимодействия между различными программами
- Регламент приемлемого уровня качества выпускаемого программного обеспечения
- Множество разнообразных стандартов, процедур, методов, инструментальных средств и типов операционной среды для разработки и управления программным обеспечением

36. В чем различие между понятиями стандарта «де-факто» и «де-юре»?
- Стандарт «Де-факто» - продукт поставщика, имеющего максимальную долю на рынке
- Стандарт «Де-юре» - продукт поставщика, имеющего минимальную долю на рынке
- Стандарт «Де-факто» - продукт, утвержденный в качестве стандарта International Standard Organization
- Стандарт «Де-юре» - продукт, утвержденный в качестве стандарта American National Standard Institute

37. Укажите известные вам международные организации, разрабатывающие стандарты
- ANSI
- ISO
- SQL
- МЭК
- UML

38. Почему нужны внутрифирменные стандарты?
- Потому, что они имеют узкую сферу полномочий
- Потому, что они абсолютно конкретны
- Потому, что они регламентируют внутренний процесс разработки программных приложений
- Потому, что они базируются на лучших методиках и технологиях, используемых фирмой

39. Укажите основные стандарты, характеризующие жизненный цикл программного средства
- DOD-STD-2167A
- ГОСТ Р ИСО/МЭК 12207
- ГОСТ ЕСПД

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ

Вопросы для подготовки к промежуточной аттестации по дисциплине (зачету)

1. Жизненный цикл программного обеспечения. Основные стадии и рабочие продукты.
2. Каскадная модель жизненного цикла ПО и ее недостатки. Каскадная модель с перекрытиями.
3. Сравнение итерационной и контролируемой итерационной моделей.
4. Спиральная модель. Роль прототипирования в этой модели. Прототипы и их классификация.
5. Стадии и рабочие процессы жизненного цикла. Организационные мероприятия.
6. Программное обеспечение поддержки жизненного цикла.
7. Классификация CASE. Понятия методологии, метода, нотации и средства CASE.
8. Репозиторий и артефакты процесса разработки.
9. Возвратное проектирование.
10. Реинжиниринг. Стадии обратного и прямого инжиниринга. Редокументирование.
11. Обзор каталогов шаблонов проектирования и рефакторингов.
12. Анализ и проектирование программного обеспечения на основе структурного подхода.
13. Диаграммы потоков данных.
14. Моделирование данных.
15. Диаграммы Костантайна.
16. Методология OMT.
17. Методика Шлеера-Меллора.
18. Унифицированный язык моделирования UML. Элементы моделей, основные отношения. Механизмы расширения UML.
19. Статические диаграммы. Классов, объектов, пакетов.
20. Динамические диаграммы. Диаграммы действий. Диаграммы взаимодействия. Диаграммы состояний.
21. Диаграммы для физического представления системы. Диаграммы компонентов и диаграммы размещения.
22. Метрики в проектах: метрики процессов и улучшение процессов программного обеспечения, метрики проектов. Роль менеджера в измерении и анализе метрик.
23. Программное обеспечение для измерения и контроля метрик.
24. Меры Холстеда - анализ применимости.
25. Сравнение топологических мер сложности.
26. Меры сложности, основанные на концепции информационных потоков между компонентами системы.
27. Особенности архитектурных мер для анализа многомодульных программ.
28. Обзор объектно-ориентированных метрик.
29. Показатели качества. Характеристики качества и их влияние на различных этапах жизненного цикла программного продукта.
30. Международные, национальные, отраслевые стандарты и организации.
31. Роль и место стандартов в процессе разработки программного обеспечения.
32. Стандарты серии ISO 9000.
33. ГОСТ “Качество программных средств”.
34. ГОСТ “Оценка качества программных средств”.
35. Модель зрелости процесса разработки программного обеспечения CMM. Уровни зрелости. Ключевые области обследования.
36. Унифицированный процесс разработки объектно-ориентированного программного обеспечения с использованием UML (RUP).
37. Рабочие процессы, работы, роли, артефакты и шаблоны RUP.
38. Рабочие процессы для малых проектов. Экстремальное программирование (XP).

ГЛОССАРИЙ

Агрегирование (Aggregation)
Агрегирование - это вид ассоциации, определяющей отношение "часть - целое". Для указания на элемент играющий роль целого к концу линии присоединяется терминатор в виде полого ромба.
Актор (Actor)
Актор является ролью объекта или объектов вне системы, которая напрямую взаимодействует с системой. Один физический объект может играть несколько ролей и, следовательно, моделируется несколькими акторами. Актор может показываться как прямоугольник класса со стереотипом "actor". Стандартная картинка стереотипа для актора рисунок человечка с названием под ним.
Ассоциация (Association)
Ассоциации - есть статические связи между классами, изображаются в виде связующей линии. Возможны рефлексивные ассоциации от класса к самому себе. Тернарные и более высокого порядка ассоциации показываются как ромбы, соединенные линиями с символами классов.
Конец ассоциации, которым она присоединяется к классу, называется ролью ассоциации (см. подробнее Роль ассоциации).
На линии связи показывается строка названия. Строка названия может содержать необязательный маленький черный сплошной треугольник; указывающий направление, в котором понимается название. Классы в ассоциации упорядочиваются, как указано стрелкой направления названия.
Для реализации этой связи может быть описан специальный ассоциированный класс. Он обозначает ассоциацию, которая имеет подобные классу свойства, такие как атрибуты, операции и, возможно, другие ассоциации. Ассоциированный класс изображается как класс, присоединенный пунктирной линией к пути ассоциации.
Атрибут (Atribute)
Атрибут - свойство объекта или класса. Атрибут семантически эквивалентен построению ассоциаций. Тип атрибута - может быть именем класса или быть составным, как array[String] of Point. В любом случае детали выражений, описывающих тип атрибута, не определяются в UML; они зависят от синтаксиса выражений, поддерживаемых используемым языком или программирования. Атрибут отображается как текстовая строка, которая может разбираться на различные свойства атрибута как элемента модели.
visibility name : type-expression = initial-value {property-string}
видимость имя : выражение-типа = начальное-значение {строка-свойств}
где видимость принимает одно из следующих значений: +общедоступный (public) #защищенный (protected) -частный (private) Глобальные для класса атрибуты отображаются с помощью подчеркивания имени и выражения типа, иначе атрибут является локальным для экземпляра. Неизменяемые атрибуты описываются с помощью свойства {frozen} ({заморожен}). В отсутствии указателя множественности атрибут принимает точно одно значение. Множественность может быть задана помещением указателя множественности в скобках после имени.
Диаграмма действий (Activity diagram)
Диаграмма активности является частным случаем диаграммы состояний, в которой все или большинство состояний являются состояниями действий и, в которой переходы, инициируются завершением действий в состояниях источниках. Диаграмма активности связывается (через модель) с классом, с реализацией операции или с использования. Назначение этой диаграммы сконцентрироваться на потоках управляемых внутренней обработкой (в противоположность внешним событиям).
Диаграмма классов (Class diagram)
Диаграмма классов это набор декларативных (статических) элементов модели, таких как классы, интерфейсы и их отношения, соединенные друг с другом в виде графа. Диаграмма классов представляет собой граф из элементов соединенных различными постоянными отношениями.

Диаграмма компонентов (Component diagram)
Компонентная диаграмма показывает зависимости между программными компонентами, включая компоненты исходного текста, компоненты двоичного кода и выполнимые компоненты. Компонентная диаграмма содержит только представление типа, а не экземпляра. Чтобы показать экземпляры компонентов, используется диаграмма развертывания. Компонентная диаграмма изображается как граф компонентов, соединенных отношениями зависимости. Компонент может также быть присоединен к компонентам с помощью физического включения, представляющего отношение композиции. Содержащая типы компонентов и типы узлов диаграмма может использоваться для показа зависимостей компилятора, которые показываются пунктирными стрелками от компонента клиента к компоненту поставщику, от которого он неким образом зависит. Виды зависимостей являются специфическими для языка и могут показываться как стереотипы зависимостей. Диаграмма может использоваться так же для показа интерфейсов и зависимостей вызова между компонентами, с помощью пунктирных стрелок от компонентов к интерфейсам других компонентов.
Диаграмма объектов (Object diagram)
Диаграмма объектов - это граф экземпляров, включая объекты и значения данных. Статическая диаграмма объектов - это экземпляр диаграммы классов; она показывает моментальный снимок детального состояния системы в определенный момент времени. Использование диаграммы объектов довольно ограничено. В основном она показывает примеры структуры данных.
Диаграмма последовательности (Sequence diagram)
Диаграмма последовательности представляет собой взаимодействие, которое является набором передаваемых между объектами сотрудничества сообщений для достижения желаемой операции или цели. Диаграмма последовательности имеет два измерения: вертикальное измерение представляет время, горизонтальное измерение представляет различные объекты. Обычно время распространяется вниз страницы. (При необходимости размерность может быть обращена.) Обычно важна только последовательность событий, однако в приложениях реального времени ось времени может иметь реальную метрику. Горизонтальное упорядочение объектов не имеет никакого значения. На диаграмме объекты могут быть сгруппированы в "дорожки процессов".
Диаграмма прецедентов использования (Use-case diagram)
Диаграммы прецедентов использования показывает отношение акторов (действующих лиц) и прецедентов использования внутри системы. С помощью такой диаграммы представляются функциональные возможности системы или класса как они проявляются при внешнем взаимодействии с системой. Диаграмма изображается в виде графа акторов, набора прецедентов использования, заключенных в границы системы, ассоциаций связи (участия) между акторами и прецедентами, и обобщений между прецедентами.
Диаграмма размещения (Deployment diagram)
Диаграммы размещения показывают конфигурацию обрабатывающих элементов времени выполнения и программных компонентов, процессов и объектов, размещенных в них. Компоненты, которые не являются сущностями времени выполнения показываться не должны. Диаграмма развертывания представляет граф узлов соединенных ассоциациями связи. Узлы могут содержать экземпляры компонентов; они показывают, что компоненты размещены или работают в узле. Компоненты соединяются с другими компонентами пунктирными стрелками зависимостей (возможно через интерфейс). Это значит, что один компонент использует сервисы другого компонента; для показа точного вида зависимости может использоваться стереотип.
Диаграмма состояний (State diagram)
Диаграмма состояний показывает последовательность состояний, через которые проходит объект или взаимодействие за период жизни. Диаграмма состояния представляет собой конечный автомат. Состояния показываются символами состояний, переходы - соединенными с ними стрелками. Состояния могут также содержать поддиаграммы путем физического включения и разделения на ячейки. Состояние - ситуация в жизни объекта или взаимодействия за период которой, они удовлетворяют некоторым условиям,

выполняют некоторые действия или ждут некоего события. Объект находится в состоянии конечное (не нулевое) время. Действия являются атомарными и не могут прерываться.

Диаграмма сотрудничества (Collaboration diagram)

Диаграмма сотрудничества показывает взаимодействие организованное вокруг объектов во взаимодействии и их связи друг с другом. Поведение обеспечивается множеством объектов, которые для достижения цели обмениваются сообщениями. Для понимания используемых в проекте механизмов важно видеть только объекты и сообщения, связанные с достижением цели. Такая статическая конструкция называется сотрудничеством. Сотрудничество может быть присоединено к операции или прецеденту использования для описания окружения, в котором возникает их поведение. Параметризованное сотрудничество представляет проектную структуру, которая может многократно использоваться в различных проектах.

Диаграмма сотрудничества представляет **сотрудничество**, которое является множеством объектов связанных в конкретном окружении, и **взаимодействие**, которое является множеством обмениваемых между объектами сотрудничества сообщений для достижения требуемых операции или результата.

Сотрудничество показывает окружение, в котором происходит взаимодействие. Динамическое поведение обмениваемой между объектами для достижения конкретной цели последовательности сообщений называется взаимодействием. Сотрудничество показывается диаграммой сотрудничества без сообщений. Добавляя сообщения, мы получаем взаимодействие. Различные совокупности сообщений могут применяться к одному и тому же сотрудничеству для создания различных взаимодействий. Диаграмма сотрудничества является графом ссылок на объекты и связи с потоками сообщений присоединенных к этим связям.

Зависимость (Dependency)

Зависимость показывает семантическое отношение между двумя (или более) элементами модели. Она указывает ситуацию, в которой изменение целевого элемента может потребовать изменения элемента источника зависимости. Зависимость показывается пунктирной стрелкой между двумя элементами модели. Элемент модели на хвосте стрелки зависит от элемента модели на острие стрелки. Стрелка может быть помечена необязательным стереотипом и необязательным названием. Следующие виды зависимости предопределены и могут обозначаться ключевыми словами:

Trace - След. Историческое соединение между двумя элементами, которые представляют одинаковое понятие на разных уровнях абстракции.

Refine - Уточнение. Историческое или возникшее соединение между двумя элементами с отображением (не обязательно полным) между ними. Описание отображения может быть присоединено к зависимости в примечании.

Uses - Использование. Ситуация в которой элемент требует присутствия другого элемента для корректной реализации и функционирования. Может стереотипироваться дальше для указания точного характера зависимости, например, вызов операций другого класса, предоставление прав доступа, иллюстрирующий объект другого класса, и т.д.

Bind - Связывание. Связывание параметров шаблона с действительными значениями для создания не параметризованного элемента.

Интерфейс (Inteface)

Интерфейс описывает видимые снаружи операции класса, компоненты или другие сущности без задания их внутренней структуры. Каждый интерфейс часто описывает только ограниченную часть поведения реального класса. Интерфейсы не имеют реализаций - они не имеют атрибутов состояний или ассоциаций, а только операции. Интерфейс может иметь отношение обобщения. Интерфейс формально эквивалентен абстрактному классу без атрибутов и методов и имеющему только абстрактные операции.

Интерфейс может быть показан как символ прямоугольника, разделенный на секции и имеющий соответствующий стереотип. Секция атрибутов может опускаться, так как она всегда пуста.

Интерфейс также может быть отображен как маленькая окружность с помещенным под символом названием интерфейса. Окружность может соединяться сплошной линией с

классами, которые поддерживают интерфейс. Это означает, что класс поддерживает все операции интерфейса (и возможно другие). Класс, который использует или требует поддерживаемые интерфейсом операции, может быть соединен с окружностью пунктирной стрелкой, указывающей на окружность.
Отношение реализации классом интерфейса изображается пунктирной линией с треугольным терминатором.

Класс (Class)

Класс - это описатель для набора объектов с общей структурой, поведением и отношениями. UML представляет нотацию для объявления классов и определения их свойств. Некоторые моделирующие элементы, которые имеют общую с классом форму (такие как интерфейсы, сигналы или утилиты), описываются с использованием соответствующих стереотипов. Классы объявляются на диаграмме классов и используются в большинстве других диаграмм.
Название класса ограничено пакетом, в котором он объявлен, и оно должно быть уникальным (среди названий классов) в этом пакете. Класс изображается как сплошной прямоугольник, разделенный горизонтальными линиями на три секции. Верхняя секция названия содержит название класса и другие основные свойства класса (включая стереотип); средняя секция списка содержит список атрибутов; нижняя секция списка содержит список операций. Чтобы показать ссылку на класс, определенный в другом пакете, используется следующий синтаксис:

Package-name::Class-name (Название_пакета::Название_класса)

В качестве расширения программное обеспечение может добавлять дополнительные секции, показывающие предопределенные или определенные пользователем свойства модели, например, для показа бизнес правил, обязанностей, изменений, управления событиями, вызовов исключений и т.п. Большинство подобных секций представляют собой просто список строк.

Классов диаграмма (Class diagram)

Диаграмма классов это набор декларативных (статических) элементов модели, таких как классы, интерфейсы и их отношения, соединенные друг с другом в виде графа. Диаграмма классов представляет собой граф из элементов соединенных различными постоянными отношениями.

Композиция (Composition)

Композиция - это форма агрегирования с сильным отношением собственности и совпадающим сроком жизни ее части со сроком жизни целого. Множественность цели композиции не может превышать единицы. Композиция может изображаться с помощью терминитора в виде сплошного заполненного ромба. UML также обеспечивает вложенную форму, когда композиция может показываться графическим вложением символов элементов частей в символ элемента целого. Вложенный элемент может иметь множественность в составном элементе. Атрибуты, в действительности, являются отношениями композиции между классом и классами этих атрибутов.

Компонентов диаграмма(Component diagram)

Компонентная диаграмма показывает зависимости между программными компонентами, включая компоненты исходного текста, компоненты двоичного кода и выполнимые компоненты. Компонентная диаграмма содержит только представление типа, а не экземпляра. Чтобы показать экземпляры компонентов, используется диаграмма развертывания. Компонентная диаграмма изображается как граф компонентов, соединенных отношениями зависимости. Компонент может также быть присоединен к компонентам с помощью физического включения, представляющего отношение композиции. Содержащая типы компонентов и типы узлов диаграмма может использоваться для показа зависимостей компилятора, которые показываются пунктирными стрелками от компонента клиента к компоненту поставщику, от которого он неким образом зависит. Виды зависимостей являются специфическими для языка и могут показываться как стереотипы зависимостей. Диаграмма может использоваться так же для показа интерфейсов и зависимостей вызова между компонентами, с помощью пунктирных стрелок от компонентов к интерфейсам других компонентов.

Обобщение (Generalization)

Обобщение является таксономическим отношением между общим элементом и специфичным элементом, который полностью непротиворечив первому элементу и добавляет к нему дополнительную информацию. Оно используется для классов, пакетов, прецедентов использования и других элементов. Обобщение показывается как сплошная линия от специфичного элемента (например, подкласса) к общему элементу (например, суперклассу), с большим полым треугольником на конце пути в точке соединения с общим элементом. Для указания семантических ограничений среди подклассов могут использоваться предопределенные ограничения. Разделенный запятыми список ключевых слов помещается в скобки либо рядом с разделяемым треугольником (если несколько путей имеют один общий треугольник) либо рядом с пунктирной линией, которая пересекает все включаемые строки обобщения. Следующие ограничения (среди прочих) могут использоваться:

Overlapping - перекрывающийся. Потомок может происходить более чем из одного подкласса.

Disjoint - непересекающийся. Потомок не может происходить более чем из одного подкласса.

Complete - завершенный. Все подклассы определены (во всяком случае, показаны). Дополнительных подклассов не ожидается.

Incomplete - незавершенный. Некоторые подклассы определены, но известно, что их список незавершен. Имеются дополнительные подклассы, которые пока отсутствуют в модели.

Объект (Object)

Объект представляет собой отдельный экземпляр класса. Он имеет значения атрибутов. Объект изображается как класс подчеркиванием элементов уровня экземпляра, но в прямоугольнике с только двумя секциями. Верхняя секция показывает название объекта и его класс (оба подчеркнутые), м.б. включая полный путь содержащего его пакета. Например:

display_window: WindowingSystem::GraphicWindows::Window.

Вторая секция показывает список атрибутов объекта и их значений. Тип атрибутам объекта уже присвоен при объявлении атрибута класса и поэтому может быть опущен.

Возможно определение анонимного объекта заданного класса.

Объектов диаграмма (Object diagram)

Диаграмма объектов - это граф экземпляров, включая объекты и значения данных. Статическая диаграмма объектов - это экземпляр диаграммы классов; она показывает моментальный снимок детального состояния системы в определенный момент времени. Использование диаграммы объектов довольно ограничено. В основном она показывает примеры структуры данных.

Ограничение (Constraint)

Ограничение - это семантическое отношение между элементами модели, определяющее всегда истинные условия и теоремы. Основные виды ограничений (такие как ограничение - соединение "или") предопределены в UML, другие определяются пользователем. Для описания определенных пользователем ограничений в конкретном программном обеспечении системы моделирования может использоваться один или более формальных языков. Предопределенным языком для записи ограничений является OCL. При отсутствии поддержки формального языка ограничений, они описываются на естественном языке. Ограничение изображается как текстовая строка в фигурных скобках ({ }).

Операция (Operation)

Операция - это сервис, выполнение которого может быть запрошено для экземпляра класса. Она имеет название и список аргументов. visibility name (parameter-list): return-type-expression {property-string} видимость имя (список-параметров): выражение-возвращаемого-типа {строка-свойств} где список-параметров - это разделенный запятыми список формальных параметров, для описания которых используется следующий синтаксис:

kind name : type-expression = default-value
вид имя : выражение-типа = значение-по-умолчанию
где вид - это in, out или inout, по умолчанию in; где имя - название формального параметра; Глобальные для класса операции отображаются с помощью подчеркивания имени и выражения типа. По умолчанию операции являются локальными для экземпляра и не маркируются. Операция, которая не модифицирует состояние системы (не имеет побочных эффектов) описывается свойством {query}; иначе, операция может изменять состояние системы. Семантика параллелизма операций определяется строкой свойств с одним из следующих имен: sequential, guarded, concurrent. В отсутствии описания семантика параллелизма неопределенна или в худшем случае должна быть назначена последовательная.
Сигнатура операции в базовом классе, наследуется всеми его наследниками). Если класс не реализует операцию (т.е. отсутствует метод), то операция либо маркируется как {abstract} либо сигнатура операции пишется курсивом, чтобы показать ее абстрактность. Любое последующее появление сигнатуры операции показывает, что подчиненный класс реализует метод операции.
Текст или алгоритм метода может быть показан в сноске, присоединенной к записи операции. Запись операции со стереотипом {signal} показывает, что класс принимает определенный сигнал. Описание поведения операции дается как присоединенная к ней сноска. Если описание сделано на формальном языке, то его текст должен быть оформлен как ограничение, иначе, если для описания поведения используется естественный язык, то должен применяться простой текст (комментарий).
Пакет (Pakage)
Пакеты являются основой управления и организации модели в UML. Пакеты предназначены для группировки элементов модели. В них могут быть упакованы все виды элементов и диаграмм UML. В качестве элемента может выступать другой пакет, т.е. пакеты могут быть вложенными друг в друга. Пакет изображается как большой прямоугольник с маленьким прямоугольником ("ярлыком"), присоединенным к одному из углов (обычно слева/сверху) большого прямоугольника. Для пакетов предопределены следующие стереотипы.
System (система) - стереотип пакета, который представляет набор моделей одной и той же моделируемой системы. Модели описывают моделируемую систему с различных точек зрения, которые не обязательно непересекающиеся. Поэтому система содержит всестороннее описание моделируемой системы и является самым верхним структурным элементом спецификации
Facade (фасад) является стереотипом пустого пакета, который ссылается на элементы модели, содержащиеся в другом пакете. Он обеспечивает общедоступное представление содержимого пакета.
Framework (скелет) является стереотипом пакета, в основном содержащего шаблоны.
Последовательности диаграмма(Sequence diagram)
Диаграмма последовательности представляет собой взаимодействие, которое является набором передаваемых между объектами сотрудничества сообщений для достижения желаемой операции или цели. Диаграмма последовательности имеет два измерения: вертикальное измерение представляет время, горизонтальное измерение представляет различные объекты. Обычно время распространяется вниз страницы. (При необходимости размерность может быть обращена.) Обычно важна только последовательность событий, однако в приложениях реального времени ось времени может иметь реальную метрику. Горизонтальное упорядочение объектов не имеет никакого значения. На диаграмме объекты могут быть сгруппированы в "дорожки процессов".
Прецедент использования (Use-case)
Прецедент использования - это блок функциональных возможностей обеспеченных системой или классом, которые проявляются при обмене сообщений между системой и одним или более внешними объектами (называемыми акторами) одновременно с действиями выполняемыми системой. Прецедент использования изображается как эллипс, содержащий название. Точка расширения - место в случае использования, в которое

может быть вставлена последовательность действий из других случаев использования. Каждая точка использования должна иметь уникальное название внутри прецедента использования. Точки расширения могут быть перечислены в секции с заголовком extension points.

Прецедентов использования диаграмма (Use-case diagram)

Диаграммы прецедентов использования показывает отношение акторов (действующих лиц) и прецедентов использования внутри системы. С помощью такой диаграммы представляются функциональные возможности системы или класса как они проявляются при внешнем взаимодействии с системой. Диаграмма изображается в виде графа акторов, набора прецедентов использования, заключенных в границы системы, ассоциаций связи (участия) между акторами и прецедентами, и обобщений между прецедентами.

Размещения диаграмма (Deployment diagram)

Диаграммы размещения показывают конфигурацию обрабатывающих элементов времени выполнения и программных компонентов, процессов и объектов, размещенных в них. Компоненты, которые не являются сущностями времени выполнения показываться не должны. Диаграмма развертывания представляет граф узлов соединенных ассоциациями связи. Узлы могут содержать экземпляры компонентов; они показывают, что компоненты размещены или работают в узле. Компоненты соединяются с другими компонентами пунктирными стрелками зависимостей (возможно через интерфейс). Это значит, что один компонент использует сервисы другого компонента; для показа точного вида зависимости может использоваться стереотип.

Роль ассоциации (Role)

Роль ассоциации описывает конец ассоциации, которым она присоединяется к классу. Роль может включать следующие элементы.

Множественность - определяется с помощью текстового описания .

Упорядочение - если множественность больше единицы, то набор связанных элементов может быть упорядоченным или нет. Виды упорядочения могут определяться в виде ограничения.

Сортировка. Правило сортировки, определяется как отдельное ограничение.

Навигация показывается присоединенной к концу линии стрелкой.

Название роли - это строка названия недалеко от конца линии. Показывает роль играемую классом, присоединенным к ближайшему от названия роли концу.

Изменчивость. Свойство {frozen} показывает, что связи объекта не могут быть добавлены, удалены или перемещены после того, как объект создан и инициализирован. Свойство {addOnly} показывает, что связи могут быть добавлены , но не могут быть модифицированы или удалены.

Видимость определяется индикатором видимости перед названием роли. Определяет видимость ассоциации со стороны заданного наименования роли.

Спецификатор - атрибут или список атрибутов, чьи значения служат для разделения множества объектов связанных с объектом с помощью ассоциации. Спецификаторы - атрибуты ассоциации. Спецификатор показывается как маленький прямоугольник, присоединенный к концу пути ассоциации между заключительным сегментом линии и символом класса, к которому он присоединяется.

Сноска (Note)

Сноска - это содержащий текстовую информацию (возможно и изображения) графический символ. Сноска используется для представления различных видов текстовой информации из метамодели, таких как ограничения, комментарии, тела методов и теговые значения. Сноска изображается в виде прямоугольника с подогнутым верхним правым углом. Она содержит произвольный текст. Сноска появляется на диаграммах и может быть, как присоединена к одному или более элементам модели пунктирными линиями, так и не присоединена ни к одному из элементов.

Состояний диаграмма (State diagram)

Диаграмма состояний показывает последовательность состояний, через которые проходит объект или взаимодействие за период жизни. Диаграмма состояния представляет собой конечный автомат. Состояния показываются символами состояний, переходы -

соединенными с ними стрелками. Состояния могут также содержать поддиаграммы путем физического включения и разделения на ячейки. Состояние - ситуация в жизни объекта или взаимодействия за период которой, они удовлетворяют некоторым условиям, выполняют некоторые действия или ждут некоего события. Объект находится в состоянии конечное (не нулевое) время. Действия являются атомарными и не могут прерываться.

Сотрудничества диаграмма (Collaboration diagram)

Диаграмма сотрудничества показывает взаимодействие организованное вокруг объектов во взаимодействии и их связи друг с другом. Поведение обеспечивается множеством объектов, которые для достижения цели обмениваются сообщениями. Для понимания используемых в проекте механизмов важно видеть только объекты и сообщения, связанные с достижением цели. Такая статическая конструкция называется сотрудничеством. Сотрудничество может быть присоединено к операции или прецеденту использования для описания окружения, в котором возникает их поведение. Параметризованное сотрудничество представляет проектную структуру, которая может многократно использоваться в различных проектах.

Диаграмма сотрудничества представляет **сотрудничество**, которое является множеством объектов связанных в конкретном окружении, и **взаимодействие**, которое является множеством обмениваемых между объектами сотрудничества сообщений для достижения требуемых операции или результата.

Сотрудничество показывает окружение, в котором происходит взаимодействие. Динамическое поведение обмениваемой между объектами для достижения конкретной цели последовательности сообщений называется взаимодействием. Сотрудничество показывается диаграммой сотрудничества без сообщений. Добавляя сообщения, мы получаем взаимодействие. Различные совокупности сообщений могут применяться к одному и тому же сотрудничеству для создания различных взаимодействий. Диаграмма сотрудничества является графом ссылок на объекты и связи с потоками сообщений присоединенных к этим связям.

Стереотип (Stereotype)

Стереотип - новый класс моделирующих элементов, который вводится в процессе моделирования. Он представляет собой подкласс уже существующих элементов с одинаковой формой (атрибутами и отношениями), но с различными целями. Название стереотипа заключается в согласованные кавычки. Допускается связывать со стереотипом особую пиктограмму (icon) или графический маркер (такой как текстура или цвет).

Теговые значения (Tagged values)

Многие виды элементов имеют подробные свойства, которые не имеют визуальной нотации. Теговое значение представляет собой пару ключевое слово - значение, которая может быть присоединена к любому элементу модели (включая как элементы диаграмм, так и семантические элементы модели). Ключевое слово называется ярлыком (tag). Каждый ярлык представляет собой особый вид свойств, подходящий к одному или более элементу модели. И ярлык, и значение кодируются строками. Теговые значения являются механизмом расширения UML, позволяющим присоединять к модели произвольную информацию. Свойства (либо атрибуты метамодели, либо связанные значения) изображаются как заключенная в фигурные скобки ({ }), разделенная запятыми последовательность описаний свойств. Описание свойства имеет форму:

keyword = value (ключевое слово = значение)

где ключевое слово - это название свойства, а значение - произвольная строка, которая показывает его значение. Если тип свойства - логический, то в случае, когда значение не указано по умолчанию принимается "true" (истина). Это значит, что для описания истинного значение достаточно включить только ключевое слово, а для описания ложного значения допустимо совсем его не включать. Для других типов свойств требуется явное указание значения.

{author = "Joe Smith", deadline = 31-March-1997, status = analysis}

{abstract}

Утилита (Utility)

Утилита - группировка глобальных переменных и процедур в форме объявления класса.

Это не фундаментальная конструкция, а средство повышения удобства программирования. Все атрибуты и операции утилиты являются глобальными переменными и процедурами. Утилита моделируется как соответствующий стереотип класса.

Шаблон (Template)

Шаблон - это описатель для класса с одним или более несвязанными параметрами. Он определяет семейство классов, каждый из которых определяется связыванием параметров с реальными значениями (инстанцированием). Параметры представляют собой типы или объекты. Атрибуты и операции в шаблоне определяются в терминах формальных параметров.

Так как шаблон содержит несвязанные параметры, то он непосредственно не используется как класс. Для создания шаблонного класса (класса инстанцированного по шаблону), параметры должны быть связаны с реальными значениями. Например, VArray обозначает класс, описанный шаблоном VArray. Количество и типы значений должны соответствовать количеству и типам параметрам шаблона с заданным названием.

Параметризация может применяться и к другим элементам модели, таким как сотрудничества или пакетам.

Для изображения шаблона в верхнем правом углу прямоугольника класса (или символа другого моделирующего элемента) добавляется маленький пунктирный прямоугольник. Он содержит список формальных параметров класса и типы их реализаций.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ПРЕПОДАВАТЕЛЯ ПО ДИСЦИПЛИНЕ

Основной целью изучения дисциплины «Программная инженерия» является освоение современных технологий разработки программного обеспечения.

Форма промежуточной аттестации знаний — **зачёт**.

Методические принципы и приемы построения учебной дисциплины «Программная инженерия». Ключевыми методическими способами подачи учебного материала по дисциплине «Программная инженерия» являются лекции и подгрупповое занятие.

Лекционное занятие — это систематическое, последовательное, устное изложение лектором учебного материала. Занятие «лекция» носит, прежде всего, обзорный характер, охватывая весь круг выносимых на изучение учебных вопросов. При проведении такого типа занятий очень важно живое слово лектора, его педагогическое мастерство как педагога, который дает студентам информационную базу. Лекции являются важной формой передачи преподавателем студентам общетеоретических знаний.

Лекции, как правило, читаются не по всем, а по наиболее сложным темам курса, не дублируют учебники, а содержат новейшие научные данные и примеры, которых может не быть в учебных пособиях. Для лучшего усвоения материала на лекционных занятиях целесообразно предварительно перед лекцией ознакомиться с положениями лекционной темы в конспекте лекций, содержащемся в данном учебно-методическом пособии либо в рекомендуемых учебниках.

Практическое (подгрупповое) занятие — другая важная форма учебного процесса. Они способствуют закреплению и углублению знаний, полученных студентами на лекциях и в результате самостоятельной работы над научной и учебной литературой и нормативными источниками. Они призваны развивать самостоятельность мышления, умение делать выводы, связывать теоретические положения с практикой, формировать профессиональное правовое сознание. На занятиях вырабатываются необходимые каждому бакалавру навыки и умения публично выступать, логика доказывания, культура профессиональной речи. Кроме того, занятия — это средство контроля преподавателей за самостоятельной работой студентов, они непосредственно влияют на уровень подготовки к итоговым формам отчетности — зачетам и экзаменам. В работе и выступлении на занятии должны содержаться следующие элементы:

- четкое формулирование соответствующего теоретического положения в виде развернутого определения;
- приведение и раскрытие основных черт, признаков, значения и роли изучаемого явления или доказательства определенного теоретического положения;
- подкрепление теоретических положений конкретными фактами.

Для качественного и эффективного изучения актуальных проблем теории необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Работа с научной литературой, в конечном счете, должна привести к выработке у бакалавра умения самостоятельно размышлять о предмете и объекте изучения, которое должно проявляться:

- в ясном и отчетливом понимании основных понятий и суждений, содержащихся в публикации, разработке доказательств, подтверждающих истинность тех или иных положений;

- в понимании студентами обоснованности и целесообразности, приводимых в книге и статье примеров, поясняющих доказательства и выводы автора. При этом будет уместно, если бакалавр самостоятельно приведет дополнительные примеры к этим выводам;
- в отделении основных положений от дополнительных, второстепенных сведений;
- в способности студента критически разобраться в содержании публикации, определить свое отношение к ней в целом, дать ей общую оценку, характеристику.

Другим важнейшим методическим приемом в учебном процессе является самостоятельная работа студента.

Самостоятельная работа в высшем учебном заведении, является важной организационной формой индивидуального изучения студентами программного материала.

В современных условиях дидактическое значение самостоятельной подготовки неизмеримо возрастает, а ее цели состоят в том, чтобы:

- повысить ответственность самих обучаемых за свою профессиональную подготовку, сформировать в себе личностные и профессионально-деловые качества;
- научить студентов самостоятельно приобретать знания, формировать навыки и умения, необходимы для программистской деятельности;
- развивать в себе самостоятельность в организации, планировании и выполнении заданий, определяемых учебным планом и указаниями преподавателя.

Достигнуть этих целей в ходе самостоятельной работы при изучении дисциплины возможно только при хорошей личной организации своего учебного труда, умении использовать все резервы имеющегося времени и подчинить их профессиональной подготовке.

Самостоятельная работа как метод обучения включает:

- изучение и конспектирование обязательной литературы в соответствии с программой дисциплины;
- ознакомление с литературой, рекомендованной в качестве дополнительной;
- изучение и осмысление специальной терминологии и понятий;
- изучение и отработка нормативных актов, комментариев к ним, проведение сравнительного анализа с предыдущим;
- сбор материала и написание контрольных, конкурсных и дипломных работ;
- изучение указанной литературы для подготовки к зачету.
- Основными компонентами содержания данного вида работы являются:
- творческое изучение учебных пособий и научной литературы;
- умелое конспектирование;
- участие в различных формах учебного процесса, научных конференциях, в работе кружков и т. д.;
- получение консультаций у преподавателя по отдельным проблемам курса;
- получение информации и опыта о работе профессионалов в процессе производственно-учебной практики;
- знакомство со специальной литературой при формировании своей личной библиотеки и др.

Данный комплекс рекомендаций позволяет студентам овладеть многими важными приемами самостоятельной работы и успешно использовать их при подготовке контрольных по дисциплине.

Важнейшей формой учебной отчетности студента являются индивидуальные задания по практическим работам. Выполнение индивидуальных заданий по практическим работам является промежуточной формой отчетности по изучаемой дисциплине и преследует цель лишь оценить способность студента к самостоятельному поиску источников, формированию содержания и его письменного изложения по указанной проблеме. Это важная составляющая изучения дисциплины, а также эффективная форма контроля знаний.

Как правило, индивидуальных заданий по практическим работам по дисциплине сугубо индивидуальны, то есть их тематика персонифицирована. Однако в отдельных

случаях темы работ могут быть адресованы и сразу нескольким бакалаврам, и группе в целом. Таким приемом преподаватель выявляет степень усвоения какой-то важной учебной проблемы и определяет необходимость проведения дополнительных занятий по какой-либо теме.

Игра позволяет влиять на правовые установки студентов. Учебно-правовые ситуации относятся к тем методическим средствам, которые позволяют осуществлять взаимосвязь понятийно-категориального уровня правосознания с поведенческим. В результате достигается не только интеллектуальный, но и эмоциональный уровень усвоения правовых понятий и идей.

Учебно-тренировочные ситуации являются специфическим методическим приемом, одним из основных видов проблемно-развивающего обучения, благодаря которому усиливается практический интерес бакалавров к теоретико-правовым вопросам.

Эффективность применения учебных ситуаций зависит от соблюдения следующих условий: знание студентами теоретического материала и наличие достаточного личного опыта и жизненного опыта вообще.

Важными в методическом плане на семинарских занятиях являются проводимые **тестовые опросы** и решение задач, которые содействуют превращению знаний в глубокие убеждения, дают простор для развития творческо-эмоциональной сферы, позволяют сделать выводы об эффективности занятий с учащимися, что в итоге повышает интерес к овладению знаниями.

Только сочетая дидактически и органически все методические способны и приемы в их диалектическом единстве и взаимосвязи мы можем добиться должного уяснения учебного материала со стороны студентов.

Методические рекомендации для преподавателей

Тема занятия	**Виды учебных занятий**	**Способы учебной деят-сти**	**Методы обучения, формы педагогического общения**	**Средства обучения**	**Формы контроля**
Жизненный цикл программного обеспечения	Лекция, подгрупповое занятие	Коллективный	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Устный опрос Проверка файлов общих и индивидуальных заданий, устный опрос, оперативное исправление ошибок
Анализ и проектирование программного обеспечения	Лекция, подгрупповое занятие	Коллективный	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Устный опрос Проверка файлов общих и индивидуальных заданий, устный опрос,

					оперативное исправление ошибок
Метрология программного обеспечения	Лекция, подгрупповое занятие	Индивидуально-групповой	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Проверка индивидуальных заданий
Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения	Лекция, подгрупповое занятие	Коллективный, Индивидуально-групповой	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Проверка индивидуальных заданий

Приложение

Тематический план изучения дисциплины «Программная инженерия»

Год набора с 2019 форма обучения очная

Наименование разделов и тем	**Всего**	**Трудоемкость по дисциплине**					**Формируемые компетенции**
		контакт. работа	в т.ч.			СРС	
			лекции	лаб. работы	практ./ сем. \ИЗ		
Жизненный цикл программного обеспечения	18	6	4	2		12	ОПК-7, ОПК-9
Анализ и проектирование программного обеспечения	22	8	4	4		14	ОПК-7, ОПК-9
Метрология программного обеспечения	36	20	4	16		16	ОПК-7, ОПК-9
Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения	32	18	4	14		14	ОПК-7, ОПК-9
Зачет							
Итого по дисциплине	**108**	**52**	**16**	**36**		**56**	
Зачетных единиц	**3**						
Курсовая работа	**-**						

Тематический план изучения дисциплины «Программная инженерия»
Год набора с 2020 форма обучения заочная

Наименование разделов и тем	**Всего**	**Трудоемкость по дисциплине**					**Формируемые компетенции**
		контакт. работа	в т.ч.			СРС	
			лекции	лаб. работы	практ./ сем. \ИЗ		
Жизненный цикл программного обеспечения	28	4	2	2		24	ОПК-7, ОПК-9
Анализ и проектирование программного обеспечения	26	2		2		24	ОПК-7, ОПК-9
Метрология программного обеспечения	24	2		2		22	ОПК-7, ОПК-9
Стандартизация и сертификация процесса разработки информационных технологий и программного обеспечения	26	2	2			24	ОПК-7, ОПК-9
Контроль	**4**	**4**					
Итого по дисциплине	**108**	**14**	**4**	**6**		**94**	
Зачетных единиц	**3**						
Контрольная работа	**+**						