МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

СОГЛАСОВАНО
Руководитель ОП

Р.И. Дремлюга

«17» июня .2019 г.

УТВЕРЖДАЮ
Директор Школы цифровой экономики

_____И.Г. Мирин

«17» _____2019 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

«ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ СИСТЕМ»
направления 09.04.01 Информатика и вычислительная техника
Магистерская программа «Кибербезопасность»
Форма подготовки очная

курс 2 семестр 3
лекции18 час.
практические занятия 36 час.
лабораторные работы 0 час.
всего часов аудиторной нагрузки 54 час.
самостоятельная работа 54 час.
контрольные работы программой не предусмотрены
курсовая работа/проект – не предусмотрено
зачет с оценкой – 3 семестр
экзамен – не предусмотрено учебным планом

Рабочая программа составлена в соответствии составлена в соответствии с требованиями Федерального государственного образовательного стандарта высшего образования по направлению подготовки/специальности 09.04.01 Информатика и вычислительная техника, утвержденного приказом Министерства образования и науки Российской Федерации от 19.09.2017 г. № 918.
Рассмотрена и утверждена на заседании Дирекции Школы цифровой экономики «17» июня 2019 года (протокол № 124-01-07-05).

Составитель(и): ст.пр. Кленин А.С.

Оборотная сторона титульного листа РПД

I. Рабочая программа пересмотрена на заседании Дирекции Школы цифровой экономики:
Протокол от «_____» _____ 20___ г. № _____

Заместитель директора ШЦЭ
по учебной и воспитательной работе _____ _____
(подпись) (И.О. Фамилия)

II. Рабочая программа пересмотрена на заседании Дирекции Школы цифровой экономики:
Протокол от «_____» _____ 20___ г. № _____

Заместитель директора ШЦЭ
по учебной и воспитательной работе _____ _____
(подпись) (И.О. Фамилия)

АННОТАЦИЯ
Б1.В.ДВ.01.03 ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ СИСТЕМ

Рабочая программа дисциплины «Программирование встроенных систем» предназначена для студентов, обучающихся по направлению подготовки 09.04.01 Информатика и вычислительная техника (уровень магистратуры), профиль «Кибербезопасность».

Дисциплина «Программирование встроенных систем» входит в часть, формируемую участниками образовательных отношений, блока «Дисциплины (модули) Б1» (Б1.В.ДВ.01) учебного плана подготовки магистров, модуль элективных дисциплин

Общая трудоемкость освоения дисциплины составляет 3 зачетных единицы или 108 часов. Дисциплина реализуется на 2 курсе в 3 семестре.

Семестр	Аудиторные Занятия			Самостоятельная работа	Контроль	Форма контроля	Всего по дисциплине	
	Лекции	Лабораторные занятия	Практические занятия				Часы	з.е.
3 семестр	18	-	36	54	-	Зачет с оценкой	108	3

Дисциплина «Программирование встроенных систем» имеет своей целью обучение базовым знаниям по организации процесса тестирования и отладки программных продуктов с использованием современных технологий и подходов.

Для достижения поставленной цели выделяются задачи курса:

- • Дать представление о встраиваемых системах.
- • Познакомить с аппаратными особенностями встраиваемых платформ.
- • Провести сравнительный обзор операционных систем, используемых во встраиваемых системах.
- • Провести обзор программных средств, используемых для разработки и отладки программного обеспечения встраиваемых систем.

- • Приобрести практические навыки для построения программных компонентов встраиваемых систем.
- • Приобрести практические навыки отладки программного обеспечения встраиваемой системы.

В результате изучения данной дисциплины у обучающихся формируются следующие общепрофессиональные и профессиональные компетенции (элементы компетенций).

Код и формулировка компетенции	**Этапы формирования компетенции**
ОПК-2. Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач	ОПК-2.1 Знать: современные информационно-коммуникационные и интеллектуальные технологии, инструментальные среды, программно-технические платформы для решения профессиональных задач ОПК-2.2 Уметь: обосновывать выбор современных информационно-коммуникационных и интеллектуальных технологий, разрабатывать оригинальные программные средства для решения профессиональных задач ОПК-2.3 Владеть: навыками разработки оригинальных программных средств, в том числе с использованием современных информационно-коммуникационных и интеллектуальных технологий, для решения профессиональных задач
ОПК-5. Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем	ОПК-5.1 Знать: современное программное и аппаратное обеспечение информационных и автоматизированных систем ОПК-5.2 Уметь: модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем для решения профессиональных задач ОПК-5.3 Владеть: навыками разработки программного и аппаратного обеспечения информационных и автоматизированных систем для решения профессиональных задач
ПК-1 Способен проводить контрольные проверки работоспособности и эффективности применяемых программно-аппаратных средств защиты информации	ПК-1.1 Знает: методы и методики оценки безопасности программно-аппаратных средств защиты информации; принципы построения программно-аппаратных средств защиты информации; принципы построения подсистем защиты информации в компьютерных системах; нормативно-правовые акты; национальные и международные стандарты в области защиты информации ПК-1.2 Умеет: определять параметры функционирования программно-аппаратных средств защиты информации; разрабатывать методики оценки защищенности программно-аппаратных средств защиты информации;

	анализировать программно-аппаратные средства защиты с целью определения уровня обеспечиваемой ими защищенности и доверия ПК-1.3 Владеет: методами и средствами оценки корректности и эффективности программных реализаций алгоритмов защиты информации; методами оценки эффективности политики безопасности, реализованной в программно-аппаратных средствах защиты информации; методами анализа программного кода с целью поиска потенциальных уязвимостей и недокументированных возможностей

I. СТРУКТУРА И СОДЕРЖАНИЕ ТЕОРЕТИЧЕСКОЙ ЧАСТИ КУРСА

Тема 1.1. Сложные системы (2 часа)

• Требования к отчётности. Обзор курса.

• Понятие сложности и сложной системы.

• Математические и физические основы теории сложных систем.

• Теория хаоса. Непредсказуемость сложных систем.

• Борьба со сложностью как фундаментальная проблема программирования.

Тема 1.2. Технологии (2 часа)

• Понятие и назначение технологии.

• Примеры технологий.

• Технологии программирования как частный случай технологий.

• Классификация и исторический обзор технологий программирования.

Тема 1.3. Коллективная разработка (2 часа)

• Классификация коллективной деятельности, особенности творческих коллективов.

• Сублинейный рост производительности труда как основная проблема крупных творческих коллективов.

• Методы и трудности объективной оценки и мотивации творческой деятельности.

• Конкретные критерии оценки производительности программистов и их недостатки, на примере технологии CMM.

• Открытая разработка как пример эффективной организации больших коллективов программистов.

Тема 1.4. Жизненный цикл программного продукта. (2 часа)

• Понятие жизненного цикла, основные этапы.

• Анализ предметной области.

• Проектирование программного продукта.

• Разработка.

• Тестирование.

• Сопровождение.

Тема 1.5. Технологии управления жизненным циклом (4 часа)

• Технология waterfall, кризис её применения.

• Повторяемость жизненного цикла.

• Инкрементная и итеративная разработка.

• Технологии короткого цикла, Agile, SCRUM.

Тема 1.6. Интеллектуальная собственность. (4 часа)

• Виды интеллектуальной собственности.

• Авторское право, лицензии.

• Закрытые и открытые лицензии.

• Свободное программное обеспечение.

• Этические и философские аспекты разработки и распространения программного обеспечения.

Тема 1.7. Контроль версий. (2 часа)

• Математический сопроцессор;

• Расширение MMX;

• Расширение SSE;

• Расширение SSE2;

• Расширение SSE3;

• Расширение SSE4.

II. СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

Практические работы организованы в виде лабораторных работ по выполнению этапов полного жизненного цикла сложной сетевой системы в рамках единого для каждой команды семестрового задания. Цель – практическая реализация изложенных на лекциях принципов коллективной разработки.

Лабораторная работа №1. Формирование творческих коллективов, выбор руководителей. Распределение заданий между коллективами. Взаимодействие с заказчиком. Описание и анализ предметной области. (4/0 час.)

Лабораторная работа №2. Формализация задания. Формирование требований (функциональных, программных, технических, к пользователю, к надежности и пр.) (4/0 час.)

Лабораторная работа №3. Распределение ролей этапа проектирования в коллективе, выбор средств проектирования программных систем. Принятие основных проектных решений, (4/0 час.)

Лабораторная работа №4. Выполнение проектных работ коллективом разработчиков. Ведение общей документации, управление проектированием со стороны руководителя коллектива (6/0 час.)

Лабораторная работа №5. Коллективная реализация проекта системы. Мобильное распределение ролей этапа реализации в коллективе с использованием системы контроля версий. Разработка протокола взаимодействия компонент системы. Контроль реализацией со стороны руководителя коллектива (6/0 час.)

Лабораторная работа №6. Разработка технологии тестирования реализованной системы. Авторское тестирование, отладка системы (2/0 час.)

Лабораторная работа №7. Сдача системы с документацией заказчику. Внедрение и сопровождение системы. (3/0 час.)

III. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Программирование встроенных систем» представлено в Приложении 1 и включает в себя:

план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;

характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;

требования к представлению и оформлению результатов самостоятельной работы;

критерии оценки выполнения самостоятельной работы.

IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

Изучение дисциплины «Программирование встроенных систем» предусматривает:

- предоставление теоретического материала в соответствии с программой, с использованием материала по списку электронных источников для каждой темы;
- выполнение домашних заданий;
- выполнение индивидуальных заданий;
- обязательная проработка материала, который будет разбираться на занятии с подбором дополнительных материалов.

Текущий контроль. Предусматривает учет посещения студентами занятий в течение периода обучения и оценку своевременности и качества изучения студентами темы и выполнения домашних заданий.

Итоговый контроль. Предусматривает рейтинговую оценку по учебной дисциплине в течение семестра и экзамен.

№ п/п	Контролируемые разделы / темы дисциплины	Коды и этапы формирования компетенций		Оценочные средства	
				текущий контроль	промежуточная аттестация
1	Языки программирования высокого уровня	ОПК-2 ОПК-5 ПК-1	знает докомпьютерные и компьютерные языки программирования, машинный код, автокод, язык ассемблера, современное развитие, взаимное влияние языков программирования высокого уровня, принципы формализации описания языков программирования, особенности структурного программирования.	Устный опрос	1 – 4
			умеет определять вид и уровень языка программирования, приводить примеры современных языков программирования указанного класса	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
			владеет языком формализации описания языков программирования	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
2	Синтаксический анализ	ОПК-2 ОПК-5 ПК-1	знает понятия формальных языков и формальных грамматик, основы иерархии грамматик по Хомскому, процесс синтаксического анализа, алгоритмы Кока-Янгера-Касами, табличные, нисходящего спуска, законы правления областями видимости	коллоквиум (УО-2).	*5 - 11*
			умеет устанавливать эквивалентность между формальными языками и алгоритмами, строить синтаксическое дерево,	Лабораторная работа (ПР-6)	Отчет по лабораторной работе

			создавать схемы записи грамматик, описывать грамматику в БНФ, реализовать контроль и преобразование типов.		
			владеет навыками анализа операторов и блоков, анализа процедур и функций, соответствия аргументов и параметров. подбора структуры данных для хранения синтаксических конструкций, реализации разбора выражений и описаний переменных, построения таблицы символов	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
3	Генерация кода реляционных БД	ОПК-2 ОПК-5 ПК-1	знает структуру и функции генератора кода, принципы эмуляции стековой архитектуры для простых выражений, понятие и назначение стекового фрейма, особенности локальной оптимизации, метод скользящего окна, алгоритм работы утилиты make	коллоквиум (УО-2).	12 – 14
			умеет осуществлять сборка программных проектов, выполнять оптимизирующие преобразования, классификацию и реализацию	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
			владеет навыками генерация кода для составных типов данных, управляющих операторов и операторов ввода-вывода, передачи аргументов и работы с параметрами, учитывать особенности	Лабораторная работа (ПР-6)	Отчет по лабораторной работе

			локальных и временных переменных		
4	Этапы программирования компиляторов	ОПК-2 ОПК-5 ПК-1	знает виды трансляторов, понятие прямой компиляции, назначение объектных модулей, редакторов связей, промежуточных ЯП , модели виртуальных машин, процесс эмуляции, теорию конечных автоматов	коллоквиум (УО-2).	15 – 17
			умеет применять технологию JIT, кросс-компиляции и раскрутки	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
			владеет способностью последовательного выполнения этапов компиляции, реализации лексических анализаторов по методам Lex и Flex, разбора регулярных выражений, навыками применения на практике принципа front-end/back-end, использования классов Scanner и Token	Лабораторная работа (ПР-6)	Отчет по лабораторной работе

Типовые контрольные задания, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература
(электронные и печатные издания)

1. Вирт, Н. Построение компиляторов [Электронный ресурс] / Н. Вирт. — Электрон. дан. — Москва : ДМК Пресс, 2010. — 192 с. — Режим доступа: https://e.lanbook.com/book/1262. — Загл. с экрана.
2. Разработка компиляторов [Электронный ресурс] / Н.Н. Вояковская [и др.]. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 374 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73654.html
3. Довек, Ж. Введение в теорию языков программирования [Электронный ресурс] / Ж. Довек, Ж.-. Леви. — Электрон. дан. — Москва : ДМК Пресс, 2013. — 134 с. — Режим доступа: https://e.lanbook.com/book/82826. — Загл. с экрана.
4. Теория и реализация языков программирования [Электронный ресурс] / В.А. Серебряков [и др.]. — 2-е изд. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 372 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73731.html
5. Серебряков, В.А. Теория и реализация языков программирования [Электронный ресурс] : учебное пособие / В.А. Серебряков. — Электрон. дан. — Москва : Физматлит, 2012. — 236 с. — Режим доступа: https://e.lanbook.com/book/5294. — Загл. с экрана.
6. FirebirdSQL Manuals .- Firebird Foundation Inc., 2013 http://www.firebirdsql.org/en/documentation/

Дополнительная литература
(печатные и электронные издания)

1. Gulutzan, Peter and Pelz, Trudy SQL-99 Complete, Really. - 2011. https://mariadb.com/kb/v/sql-99-complete-really/
2. MariaDB Documentation. SkySQL Corporation Ab., 2013 https://mariadb.com/kb/en/mariadb-documentation/

3. PostgreSQL Manuals, version 9.3 .- The PostgreSQL Global Development Group, 2013.
4. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework). — ISO/IEC, 2011 — 90 с.
5. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation). — ISO/IEC, 2011 — 1483 с.
6. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata). — ISO/IEC, 2011 — 316 с.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет»

1. Сообщество пользователей SQL [http://sql.ru].
2. Утилита для администрирования FlameRobin [http://flamerobin.org]

Перечень информационных технологий и программного обеспечения

1. Электронная презентация
2. Электронная рассылка
3. Электронный журнал успеваемости
4. Электронные поисковые приложения
5. Электронное тестирование

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

На изучение дисциплины отводится 54 часа аудиторных (лекционных и лабораторных) занятий. На занятиях перед выдачей индивидуальных заданий преподаватель объясняет теоретический материал по заданной теме. Вводит основные требования к его выполнению. Приводит примеры. Необходимо поддерживать непрерывный контакт с аудиторией, отвечать на возникающие у студентов вопросы. На практических занятиях преподаватель разбирает на примерах принципы и аспекты реализации задания по заданной теме.

По ряду тем студентам предлагается работать самостоятельно, выполняя полный обзор по теме. Преподаватель контролирует работу студентов, отвечает на возникающие вопросы, предоставляет список литературных источников для освоения темы, а также перечень вопросов для самопроверки. Если знаний, полученных в аудитории, оказалось недостаточно, студент может самостоятельно повторно просмотреть методические указания.

После выполнения задания, студент оформляет материал в форме программного кода и отправляет его на проверку преподавателю по электронной почте, либо предъявляет на компьютере во время занятия. Студент отвечает устно во время занятия по заданной теме.

По данному курсу разработаны учебные материалы. Для успешного достижения учебных целей занятий должны выполняться следующие основные требования:

-соответствие действий обучающихся ранее изученным на лекционных и семинарских занятиях методикам и методам.

-максимальное приближение действий студентов к реальным, соответствующим будущим функциональным обязанностям.

-поэтапное формирование умений и навыков, т.е. движение от знаний к умениям и навыкам, от простого к сложному и т.д..

-использование при работе на тренажерах или действующей технике фактических документов, технологических карт, бланков и т.п.

-выработка соответствующих индивидуальных и коллективных умений и навыков.

Студент должен:

-научиться работать с книгой, документацией и схемами, пользоваться справочной и научной литературой.

-научиться работать с электронными литературными источниками.

-формировать умение учиться самостоятельно, т.е. овладевать методами, способами и приемами самообучения, саморазвития и самоконтроля.

Рекомендации по подготовке к экзамену:

Рекомендуется еще раз самостоятельно ответить на вопросы для самопроверки, приведенные в методических материалах.

При ответе на каждый вопрос экзамена студент должен продемонстрировать знание определения указанного понятия, связанных с ним особенностей реализации и применения, умение реализовать указанную операцию, а также навыки иллюстрации теоретических принципов на предложенных простых примерах.

VII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Лекционная аудитория: мультимедийный проектор Optima EX542I – 1 шт.; аудио усилитель QVC RMX 850 – 1 шт.; колонки – 1 шт.; ноутбук; ИБП – 1 шт.; настенный экран; микрофон – 1 шт.

Компьютерные классы ДВФУ (кампус на о. Русском, Аякс 10, корпус D, ауд. 733, 733а) по 15 персональных компьютеров Extreme DOU E 8500/500 GB/ DVD+RW.

Системное и прикладное обеспечение ПЭВМ.

IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

Изучение дисциплины «Программирование встроенных систем» предусматривает:

– предоставление теоретического материала в соответствии с программой, с указанием материала по списку электронных источников для каждой темы;

– взаимодействие с членами творческого коллектива, выполнение индивидуальных заданий руководителя коллектива;

– обязательное согласование проектных, технических и прочих решений с руководителем коллектива.

Текущий контроль. Предусматривает учет руководителем коллектива своевременности и качества выполненных студентами

Итоговый контроль. Предусматривает рейтинговую оценку по учебной дисциплине в течение семестра (распределение баллов осуществлется руководителем коллектива) и экзамен.

№ п/п	Контролируемые разделы / темы дисциплины	Коды и этапы формирования компетенций		Оценочные средства	
				текущий контроль	промежуточная аттестация
1	Сложные программные системы	ОПК-5 ОПК-2 ПК-1	знает понятие сложности и особенности сложной системы, математические и физические основы теории сложных систем, требования к отчётности и документированию программных систем, непредсказуемость, теорию хаоса. сложных систем, фундаментальность проблемы борьбы со сложностью	Устный опрос	1 - 4
			умеет определять уровень сложности программной системы,	Лабораторная работа (ПР-6)	Отчет по лабораторной работе

			использовать современные методы борьбы со сложностью системы		
			владеет навыками коллективной разработки сложных программных систем	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
2	Технологии коллективной разработки сложных программных систем	ОПК-5 ОПК-2 ПК-1	Знает понятие и назначение технологии, в том числе технологии программирования, принципы классификации и исторические предпосылки современных технологий программирования	коллоквиум (УО-2).	*5 - 11*
			Умеет выполнять классификацию и представить исторический обзор технологий программирования, привести примеры современных технологий	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
			владеет языком и средствами формализации описания процесса разработки и модификации сложной программной системы	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
3	Технологии управления жизненным циклом программного продукта	ОПК-5 ОПК-2 ПК-1	•знает понятие и структуру жизненного цикла, основные его этапы, понимает свойство повторяемости жизненного цикла, инкрементный и итеративный характер жизненного цикла	коллоквиум (УО-2).	12 – 14
			умеет, осуществлять анализ предметной	Лабораторная работа	Отчет по лабораторн

			области, проектирование, разработку, тестирование, сопровождение программного продукта	(ПР-6)	ой работе
			владеет навыками использования технология waterfall, информацией о кризисе её применения, навыками технологии короткого цикла SCRUM	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
4	Аспекты создания интеллектуальной собственности	ОПК-5 ОПК-2 ПК-1	Знает виды интеллектуальной собственности, юридические аспекты установления авторского права, получения лицензии, этические и философские аспекты разработки и распространения программных продуктов	коллоквиум (УО-2).	15 – 17
			умеет применять и распространять свободное программное обеспечение	Лабораторная работа (ПР-6)	Отчет по лабораторной работе
			владеет навыками оформления открытых (закрытых) лицензий, соблюдения авторских прав на программные продукты,	Лабораторная работа (ПР-6)	Отчет по лабораторной работе

Типовые контрольные задания, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература
(электронные и печатные издания)

1. Технология программирования [Электронный ресурс]: учебное пособие/ Ю.Ю. Громов [и др.].— Электрон. текстовые данные.— Тамбов: Тамбовский государственный технический университет, ЭБС АСВ, 2013.— 173 с.— Режим доступа: http://www.iprbookshop.ru/63910.html.— ЭБС «IPRbooks»
2. Технология программирования: учебник / Г.С. Иванова. — Москва : КноРус, 2011. — 333 с. — ISBN 978-5-406-00519-4.
3. Котляров В.П. Основы тестирования программного обеспечения [Электронный ресурс]/ Котляров В.П.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 334 с.— Режим доступа: http://www.iprbookshop.ru/62820.html.— ЭБС «IPRbooks»
4. Сергеев С.Ф. Методы тестирования и оптимизации интерфейсов информационных систем [Электронный ресурс]: учебное пособие/ Сергеев С.Ф.— Электрон. текстовые данные.— СПб.: Университет ИТМО, 2013.— 117 с.— Режим доступа: http://www.iprbookshop.ru/68664.html.— ЭБС «IPRbooks»
5. **Информационные системы**: Учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - 2-е изд. - М.: Форум: НИЦ ИНФРА-М, 2014. - 448 с.: ил.; 60x90 1/16. - (Высшее образование). (переплет) ISBN 978-5-91134-833-5 - Режим доступа: http://znanium.com/catalog/product/435900
6. Битюцкая Н.И. Разработка программных приложений [Электронный ресурс]: лабораторный практикум/ Битюцкая Н.И.— Электрон. текстовые данные.— Ставрополь: Северо-Кавказский федеральный университет, 2015.— 140 с.— Режим доступа: http://www.iprbookshop.ru/63128.html.— ЭБС «IPRbooks»

7. Грэхем, Л. Разработка через тестирование для iOS [Электронный ресурс] / Л. Грэхем ; пер. с англ. Киселев А.Н.. — Электрон. дан. — Москва : ДМК Пресс, 2013. — 272 с. — Режим доступа: https://e.lanbook.com/book/63183. — Загл. с экрана.]

Дополнительная литература
(печатные и электронные издания)

1. Долженко А.И. Технологии командной разработки программного обеспечения информационных систем [Электронный ресурс]/ Долженко А.И.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 300 с.— Режим доступа: http://www.iprbookshop.ru/39569.html.— ЭБС «IPRbooks»
2. Введение в программные системы и их разработку [Электронный ресурс]/ С.В. Назаров [и др.].— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 649 с.— Режим доступа:

Перечень ресурсов информационно-телекоммуникационной сети «Интернет»

1. Литвиненко Н. А.Технология программирования на C++. Win32 API-приложения: Учебное пособие / Литвиненко Н.А. - СПб:БХВ-Петербург, 2010. - 280 с. ISBN 978-5-9775-0600-7 - Режим доступа: http://znanium.com/catalog/product/351463
2. Сайт AnandTech [http://anandtech.com].
3. Сайт Tom's Hardware [http://tomshardware.com].
4. Сайт Ars Technica CPU and Chipset Guide [http://arstechnica.com/cpu/index.html].

Перечень информационных технологий и программного обеспечения

1. Электронная рассылка
2. Электронный журнал успеваемости
3. Электронные поисковые приложения
4. Система контроля версий

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

На изучение дисциплины отводится 54 часа аудиторных (лекционных и лабораторных) занятий. На занятиях перед выдачей коллективных заданий преподаватель вводит основные требования к его выполнению, формирует творческие коллективы, назначает руководителей коллективов, озвучивает полномочия, права и обязанности руководителей и рядовых членов коллективов. Преподаватель объясняет теоретический материал об очередном аспекте разработки программной системы. Приводит примеры, поддерживает непрерывный контакт с аудиторией, отвечает на возникающие у студентов вопросы. На практических занятиях преподаватель контролирует деятельность коллектива через его руководителя, оказывает текущую организационную и профессиональную поддержку, уточняет особенности выполнения очередного этапа разработки системы.

По всем аспектам студентам предлагается проявлять самостоятельность в рамках коллективной работы. Преподаватель контролирует работу студентов, в части случаев отвечает на возникающие вопросы, в других случаях рекомендует выполнить самостоятельный обзор возможных источников информации.

После выполнения задания, коллектив оформляет отчет и документацию в соответствии с указанным в методических материалах требованиями, предоставляет на проверку преподавателю код программного продукта на компьютере во время занятия. Руководитель отвечает на вопросы преподавателя, дает необходимые пояснения по системе.

По данному курсу разработаны методические материалы. Для успешного достижения учебных целей занятий должны выполняться следующие основные требования:

- соответствие действий обучающихся ранее изученным на лекционных и семинарских занятиях методикам и технологиям.

- максимальное приближение действий студентов к реальным, соответствующим будущим функциональным обязанностям.

- формирование умений и навыков по каждому этапу разработки системы, т.е. каждый студент должен выполнять практические задания из всех этапов.

- при работе фактических документов, технологических карт, бланков и т.п.

- выработка соответствующих индивидуальных и коллективных умений и навыков.

Студент должен:

-научиться самостоятельно предлагать обоснованные варианты решений, работать с бумажными и электронными источниками, пользоваться справочной и научной литературой.

-формировать умение учиться самостоятельно, т.е. овладевать методами, способами и приемами самообучения, саморазвития и самоконтроля.

Рекомендации по подготовке к экзамену:

Рекомендуется еще раз самостоятельно ответить на теоретические вопросы для самопроверки, приведенные в учебных материалах, уже после приобретения практических умений и навыков в ходе разработки программной системы.

При ответе на каждый вопрос экзамена студент должен продемонстрировать знание определения указанного понятия, связанных с ним особенностей реализации и применения, умение реализовать указанную операцию, а также навыки иллюстрации теоретических принципов на предложенных простых примерах.

VII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Мультимедийная аудитория: Проектор DLP, 4000 ANSI Lm, 1920x1080, 2000:1 FD630u Mitsubishi; Проектор DLP, 2800 ANSI Lm, 1920x1080, 2000:1 GT1080 Optoma; Проектор DLP, 3000 ANSI Lm, WXGA 1280x800, 2000:1 EW330U Mitsubishi; Беспроводные ЛВС для обучающихся обеспечены системой на базе точек доступа 802.11a/b/g/n 2x2 MIMO(2SS).	690922, Приморский край, г. Владивосток, о. Русский, п. Аякс, 10, г. Владивосток, о. Русский, п. Аякс , корпус G, ауд. G470

Приложение 1

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

по дисциплине

«ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ СИСТЕМ»

Направление подготовки
09.04.01 Информатика и вычислительная техника

Форма подготовки очная

Владивосток
2019

План-график выполнения самостоятельной работы по дисциплине

№ п/п, название	Дата/сроки выполнения	Вид СРС	Примерные нормы времени на выполнение	Форма контроля
1. Сложные программные системы. Аспекты технологии их коллективной разработки	Четвертая неделя семестра	ИДЗ	2 недели	Коллоквиум
2. Технологии управления жизненным циклом программного продукта. Аспекты создания интеллектуальной собственности	Пятая неделя семестра	ИДЗ	1 неделя	Коллоквиум
3. Формирование требований к программной системе	Шестая неделя семестра	ИДЗ	2 недели	Документальный отчет
4. Разработка проекта программной системы	Восьмая неделя семестра	ИДЗ	3 неделя	Документальный отчет
5. Реализация, тестирование и отладка проекта программной системы	Одиннадцатая неделя семестра	ИДЗ	4 недели	Программный код
6. Подготовка программной системы к внедрению	Сессия	ИДЗ	1 неделя	Итоговое тестирование кода

Характеристика заданий самостоятельной работы

Содержание самостоятельной работы студентов по дисциплине

1. Изучение необходимых для реализации системы технических аспектов и программных средств, выходящий за рамки предыдущих курсов обучения.

2. Разработка алгоритмов и программ при выполнении лабораторных заданий.

3. Подготовка системы к тестированию.

4. Подготовка студента к экзамену.

Текущая СРС.

– работа с лекционным материалом, поиск и обзор литературы и электронных источников информации по программным и техническим средствам реализации программных продуктов;

– изучение тем, вынесенных на самостоятельную проработку;

– подготовка к лабораторным работам;

– подготовка к экзамену.

Творческая проблемно-ориентированная самостоятельная работа (ТСР).

ТСР направлена на развитие интеллектуальных умений, комплекса универсальных (общекультурных) и профессиональных компетенций, повышение творческого потенциала бакалавров и заключается в:

– поиске, анализе, структурировании и документации информации;

– разработке сложного программного продукта;

– исследовательской работе и участии в творческих студенческих коллективах;

– анализе научных публикаций по заранее определенной руководителем коллектива теме.

Требования к представлению и оформлению результатов самостоятельной работы

1. Устный ответ по вопросу экзамена.
2. Отчет по коллективному проекту системы, оформленный согласно установленным стандартам.
3. Исходный код разработанной системы.

Критерии оценки выполнения самостоятельной работы

Оценка результатов самостоятельной работы организуется как единство двух форм: самоконтроль и контроль со стороны преподавателя.

Критерии оценки:

1. Анализ предметной области и обзор существующих решений (10 баллов)
2. Коллективный выбор и обоснование проектных решений (10 баллов)
3. Разработка коллективного проекта системы (10 баллов)
4. Процесс и качество реализации проекта (10 баллов)
5. Качество документации реализованной системы (10 баллов)
6. Надежность системы и соответствие требованиям заказчика (10 баллов)

Приложение 2

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине

«ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ СИСТЕМ»

Направление подготовки

09.04.01 Информатика и вычислительная техника

Форма подготовки очная

Владивосток
2019

Результаты обучения (компетенции из ФГОС)	Знает	Умеет	Владеет
ОПК-5 ОПК-2 ПК-1	Методы проектирования, разработки и сопровождения промышленных информационных систем Современных инструментальных CASE–средств автоматизированного проектирования информационных систем	Проектировать промышленные информационные системы, используя современные инструментальные среды.	Навыками применения методов и средств анализа и проектирования промышленных информационных систем; навыками разработки сетевых прикладных программ
Эталонный	Основной и дополнительный материал, предусмотренный компетенцией, без ошибок и погрешностей	**Умеет** в полном объеме …	всеми навыками, демонстрируя их не только в стандартных ситуациях, но и при решении нестандартных задач
Продвинутый	основной материал, предусмотренный компетенцией, без ошибок и погрешностей	**Умеет** с незначительными погрешностями …	основными навыками, демонстрируя их в стандартных ситуациях, в том числе при решении дополнительных задач
Пороговый	большинство основных понятий, изучаемых в рамках дисциплины	**Умеет** с погрешностями …	некоторыми основными навыками, демонстрируя их в стандартных ситуациях

Критерии оценивания

В течение семестра студентам последовательно выдаются практические задания по семи темам, каждая из которых имеет вес от 10%. Своевременность выполнения заданий также учитывается и имеет вес 10%. Для получения оценки «отлично» необходимо иметь итоговый балл не ниже 80%, оценки «хорошо» – 60 %, оценки «удовлетворительно» – 50 %.

Вопросы к зачету по дисциплине
«Проектирование промышленных информационных систем»

1. Сложные системы. Понятие сложности. Источники сложности.
2. Свойства работоспособных сложных систем.
3. Методы борьбы со сложностью.
4. Понятие технологии. Особенности технологии программирования.
5. Жизненный цикл программного обеспечения.
6. Методологии управления разработкой программного обеспечения: обзор и история.
7. Методологии CMM/CMMI, RUP, ГОСТ.
8. Методологии Agile/Scrum, TDD, XP.
9. Управление коллективом разработчиков.
10. Требования к программной системе, управление требованиями.
11. Методы и инструменты проектирования программных систем.
12. Визуальное проектирование, язык UML: назначение и общие принципы.
13. Статические диаграммы UML.
14. Динамические диаграммы UML.
15. Риски, оценка и управление рисками.
16. Системы контроля версий: назначение, классификация, принципы работы.
17. Распределённые системы контроля версий на примере Git.
18. Управление изменениями и релизами.
19. Системы управления инцидентами: назначение, принципы работы.
20. Структура и жизненный цикл инцидента.
21. Управление качеством кода. Технический долг.
22. Формальные методики обеспечения корректности.
23. Неформальные методики обеспечения корректности: тестирование и отладка.
24. Документирование программных систем.

Вопросы к коллоквиуму по дисциплине
«Программирование встроенных систем»

1. Предмет и объекты изучения дисциплины «Проектирование промышленных информационных систем».
2. Пояснить понятие Аспектно-ориентированное сборочное программирование

3. Пояснить принципы технологии Восходящего программирования. Описать метод восходящего проектирования
4. Пояснить идею технологии Программирование "снизу вверх"
5. Диаграмма функционального моделирования – назначение и правила построения
6. Назначение и преимущества Заглушек
7. Пояснить принципы технологии Императивное программирование
8. Инструментарий технологии программирования
9. Инструментарий технологии программирования –назначение, примеры
10. Сборочное программирование – назначение технологии.
11. Общая концепция и область применения Компонентного сборочного программирования
12. Компьютерный дарвинизм
13. Описать подход –Компьютерный дарвинизм
14. Основные характеристики Логического программирования
15. Описать метод расширения ядра
16. Модульное программирование – основные концепции, преимущества
17. Модульное сборочное программирование – отличительные свойства.
18. Методика Нисходящего программирования – преимущества и недостатки
19. Технология Программирование "сверху вниз"
20. Методика Нисходящего программирования
21. Объектно-ориентированный подход к программированию
22. Объектно-ориентированное сборочное программирование – суть технологии
23. Идея Синтезирующего программирования
24. Основы Структурного программирования
25. Пояснить понятия: Компонент, Инкапсуляция, Полиморфизм, Наследование, Контейнер
26. Сфера применения технологии называемой "чёрный ящик".
27. Парадигма Распределенные вычисления

Методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности

Глоссарий дисциплины

«Программирование встроенных систем»

- Технология программирования - дисциплина, изучающая технологические процессы программирования и порядок их прохождения.
- Аспектно-ориентированное сборочное программирование - разновидность сборочного программирования, основанная на сборке полнофункциональных приложений из многоаспектных компонентов, инкапсулирующих различные варианты реализации.
- Восходящее программирование
- Программирование "снизу вверх"
- Восходящее программирование - методика разработки программ, при которой крупные блоки собираются из ранее созданных мелких блоков.
- Восходящее программирование начинается с разработки ключевых процедур и подпрограмм, которые затем постоянно модифицируются.
- Диаграмма функционального моделирования
- Structured analysis and design technique (SADT)
- Диаграмма функционального моделирования - инструмент разработки функциональных спецификаций в виде диаграмм, фрагментов текста и глоссария, связанных перекрестными ссылками. В состав диаграммы входят:
- - блоки, изображающие активность моделируемой системы; и
- - дуги, связывающие блоки вместе и изображающие взаимодействия и взаимосвязи между ними.
- Место соединения дуги с блоком определяет тип интерфейса:
- - управляющая информация входит в блок сверху;
- - входная информация, подвергающаяся обработке, показывается с левой стороны блока;
- - выходная информация показывается с правой стороны;
- - механизм, осуществляющий операцию, представляется дугой, входящей в блок снизу.

- Заглушка - структурном программировании - процедура, представленная точной спецификацией заголовка и пустым телом. Заглушка позволяет компилировать и выполнять программу в отладочном режиме.
- Императивное программирование - технология программирования, характеризующаяся принципом последовательного изменения состояния вычислителя пошаговым образом. При этом управление изменениями полностью определено и полностью контролируемо.
- Инструментарий технологии программирования
- Инструментарий технологии программирования - программные продукты, предназначенные для поддержки технологии программирования.
- Компонентное сборочное программирование
- Компонентное сборочное программирование - объектно-ориентированное сборочное программирование, основанное на распространении классов в бинарном виде и предоставление доступа к методам класса через строго определенные интерфейсы.
- Компонентное сборочное программирование поддерживают технологические подходы COM, CORBA, .Net.
- Компьютерный дарвинизм
- Компьютерный дарвинизм - подход к разработке программных систем, основанный на принципе восходящей разработки при интенсивном тестировании. Подход состоит из трех основных процессов: макетирования, тестирования и отладки.
- Логическое программирование
- Логическое программирование - программирование в терминах фактов и правил вывода, с использованием языка, основанного на формальных исчислениях.
- Метод восходящего проектирования
- Метод восходящего проектирования - подход, при котором в первую очередь определяются вспомогательные модули, которые потребуются для проектируемой программы.
- Метод расширения ядра
- Метод расширения ядра - метод восходящего программирования, при котором основное внимание уделяется выявлению множества вспомогательных модулей, а не определению функции всей программы в целом.
- Модульное программирование

- Модульное программирование - метод разработки программ, предполагающий разбиение программы на независимые модули. Считается, что:
- - оптимальный по размерам модуль целиком помещается на экране дисплея;
- - разделение большой программы на модули облегчает ее разработку, отладку и сопровождение.
- Модульное сборочное программирование
- Модульное сборочное программирование - разновидность сборочного программирования, основанная на процедурах и функциях методологии структурного императивного программирования.
- Нисходящее программирование
- Программирование "сверху вниз"
- Нисходящее программирование - методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.
- Объектно-ориентированное программирование - технология программирования, при которой программа рассматривается как набор дискретных объектов, содержащих, в свою очередь, наборы структур данных и процедур, взаимодействующих с другими объектами.
- Объектно-ориентированное сборочное программирование
- Объектно-ориентированное сборочное программирование - разновидность сборочного программирования:
- - основанная на методологии объектно-ориентированного программирования; и
- - предполагающая распространение библиотек классов в виде исходного кода или упаковку классов в динамически компонуемую библиотеку.
- Сборочное программирование
- Сборочное программирование - технология программирования, при которой программа собирается посредством повторного использования уже известных фрагментов программ.
- Синтезирующее программирование - программирование, предполагающее синтез программы по ее спецификации.
- Структурное программирование - методология и технология разработки программных комплексов, основанная на принципах:
- - программирования "сверху-вниз";

- - модульного программирования.
- При этом логика алгоритма и программы должны использовать три основные структуры: последовательное выполнение, ветвление и повторение.
- Компонент - Составная часть распределенного приложения
- Инкапсуляция, encapsulation - механизм, который объединяет данные и код, манипулирующий зтими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В объектно-ориентированном программировании код и данные могут быть объединены вместе; в этом случае говорят, что создаётся так называемый "чёрный ящик". Когда коды и данные объединяются таким способом, создаётся объект (object). Другими словами, объект - это то, что поддерживает инкапсуляцию.
- Внутри объекта коды и данные могут быть закрытыми (private). Закрытые коды или данные доступны только для других частей этого объекта. Таким образом, закрытые коды и данные недоступны для тех частей программы, которые существуют вне объекта. Если коды и данные являются открытыми, то, несмотря на то, что они заданы внутри объекта, они доступны и для других частей программы. Характерной является ситуация, когда открытая часть объекта используется для того, чтобы обеспечить контролируемый интерфейс закрытых элементов объекта.
- На самом деле объект является переменной определённого пользователем типа. Может показаться странным, что объект, который объединяет коды и данные, можно рассматривать как переменную. Однако применительно к объектно-ориентированному программированию это именно так. Каждый элемент данных такого типа является составной переменной.
- Полиморфизм, polymorphism - (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Например для языка Си, в котором полиморфизм поддерживается недостаточно, нахождение абсолютной величины числа требует трёх различных функций: abs(), labs() и fabs().

Эти функции подсчитывают и возвращают абсолютную величину целых, длинных целых и чисел с плавающей точкой соответственно. В C++ каждая из этих функций может быть названа abs(). Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. В C++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций (function overloading).

- В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор. Вам, как программисту, не нужно делать этот выбор самому. Нужно только помнить и использовать общий интерфейс. Пример из предыдущего абзаца показывает, как, имея три имени для функции определения абсолютной величины числа вместо одного, обычная задача становится более сложной, чем это действительно необходимо.
- Полиморфизм может применяться также и к операторам. Фактически во всех языках программирования ограниченно применяется полиморфизм, например, в арифметических операторах. Так, в Си, символ + используется для складывания целых, длинных целых, символьных переменных и чисел с плавающей точкой. В этом случае компилятор автоматически определяет, какой тип арифметики требуется. В C++ вы можете применить эту концепцию и к другим, заданным вами, типам данных. Такой тип полиморфизма называется перегрузкой операторов (operator overloading).
- Ключевым в понимании полиморфизма является то, что он позволяет вам манипулировать объектами различной степени сложности путём создания общего для них стандартного интерфейса для реализации похожих действий.
- Наследование, inheritance - это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов (hierarchical classification).

- Применение иерархии классов делает управляемыми большие потоки информации. Например, подумайте об описании жилого дома. Дом - это часть общего класса, называемого строением. С другой стороны, строение - это часть более общего класса - конструкции, который является частью ещё более общего класса объектов, который можно назвать созданием рук человека. В каждом случае порождённый класс наследует все, связанные с родителем, качества и добавляет к ним свои собственные определяющие характеристики. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающи его определяли. Однако при использовании наследования можно описать объект путём определения того общего класса (или классов), к которому он относится, с теми специальными чертами, которые делают объект уникальным. Наследование играет очень важную роль в ООП.
- Контейнер в программировании - структура, позволяющая инкапсулировать в себя объекты разных типов.
- Среди "широких масс" программистов наиболее известны контейнеры, построенные на основе шаблонов, однако существуют и реализации в виде библиотек (наиболее широко известна библиотека GLib). Кроме того, применяются и узкоспециализированные решения. Примерами контейнеров являются контейнеры из стандартной библиотеки (STL) - map, vector и др. В контейнерах часто встречается реализация алгоритмов для них. В ряде языков программирования (особенно скриптовых типа Perl или PHP) контейнеры и работа с ними встроена в язык.
- Контейнер, в отличие от коллекции, в общем случае, обычно не допускает явного задания числа элементов и обычно не поддерживает ветвистой структуры. Впрочем, это сильно зависит от реализации, поскольку многие реализации (особенно ориентированные на persistant storage) позволяют задавать размеры при создании контейнера.
- Распределенные вычисления - Парадигма организации приложений, в которой различные части программы могут исполняться на разных компьютерах в сети.
- Active Directory - Сетевая служба каталогов Microsoft, которую корпорация включила в состав Windows с версии 2000.
- ActiveX - Предлагаемый Microsoft способ разработки программных компонентов; название группы технологий, разработанных Microsoft для

программирования компонентных объектных приложений на основе модели COM.

- ActiveX control - управляющий элемент ActiveX; введенное в 1996 г. Microsoft новое название независимых программируемых компонентов, ранее называемых OLE controls, OCXs, OLE custom controls; в отличие от последних позволяют работать с Internet.
- CDN, Content Delivery Network, Content Distribution Network, Сеть доставки и дистрибуции контента - географически распределённая сетевая инфраструктура, позволяющая оптимизировать доставку и дистрибуцию контента конечным пользователям в сети Интернет. Использование контент-провайдерами CDN способствует увеличению скорости загрузки интернет-пользователями аудио-, видео-, программного, игрового и других видов цифрового контента в точках присутствия сети CDN.
- Простыми словами, CDN-это промежуточный хостинг. Контент вашего сайта сначала загружается на CDN хостинг, а затем отдается пользователю. Часто используется в криминальных целях воровства контента, информационного подавления неугодных интернет-ресурсов, перехвата пользовательского трафика и позиций в поисковых системах.
- Популярные бесплатные CDN-сервисы: CloudFlare, Incapsula.
- COM, Component Object Model - Программная архитектура Microsoft, поддерживающая компонентный подход к разработке приложений; модель компонентных объектов Microsoft; стандартный механизм, включающий интерфейсы, с помощью которых одни объекты предоставляют свои сервисы другим; является основой многих объектных технологий, в том числе OLE и ActiveX).
- COM+ - Модернизация COM и Microsoft Transaction Server, который упрощает разработку сложных распределенных приложений.
- Common Object Request Broker Architecture, CORBA - Основной конкурент DCOM в области построения распределенных программных систем.
- DLL, Dynamic Link Library - динамически подключаемая библиотека, понятие операционной системы Microsoft Windows; динамическая библиотека, позволяющая многократное применение различными программными приложениями. К DLL иногда причисляют также элементы управления ActiveX и драйвера. В мире UNIX аналогичные функции выполняют т. н. shared objects (<разделяемые объекты>).

Формат файлов *.dll придерживается тех же соглашений, что и формат исполняемых файлов *.exe, сочетая код, таблицы и ресурсы.

- Microsoft Transaction Server, MTS - Усовершенствование в составе COM, которое реализует поддержку транзакций баз данных.
- OLE, Object Linking and Embedding - общее название (до 1996 г.) группы объектно-ориентированных технологий Microsoft на основе COM (OLE 1, OLE 2, OLE automation, OLE Database и др.).
- OCX, OLE Custom eXtension - перемещаемые элементы управления, OLE custom control, OLE control. Упрощенно можно сказать, что файлы *.ocx - это элементы управления ActiveX, выполняющие примерно те же функции, что и файлы *.dll.
- - OLE custom control
- специализированный управляющий элемент OLE, OLE control.
- OLE control - управляющие элементы OLE, программируемые компоненты-приложения с интерфейсом на базе OLE, позволяющим легко включать их в другие приложения; с 1996 г. называются ActiveX control. Синонимы: OCX, OLE custom control.
- Remote Procedure Call, RPC - Удаленный вызов процедуры - сообщение, посылаемое по сети, которое позволяет программе, установленной на одном компьютере, инициировать выполнение необходимой операции на другом.

Пример выполнения задания

«Разработка протокола взаимодействия компонент системы»

FEFU MMORPG Protocol — FEMP/0.3

Requirements

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC 2119.

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be unconditionally compliant; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be conditionally compliant.

Introduction

Client and server communicate with request and response messages. Each message MUST be represented by a single JSON object. Messages are sent via either HTTP/1.1 or WebSocket protocol.

Request message MUST contain a key action with a corresponding string value determining required action.

Each request message MUST be answered with a corresponding response message.

Response message MUST contain a key result with a corresponding value describing result.

Each response MUST contain key action with a value of the same key action of corresponding request. This one serves for describing response type.

If server does not handle the request regardless of underlying transport and with regard to value of 'action' key then server MUST respond with a key result of value badAction.

Both request and response messages MAY contain any other key/value pairs specific for particular request/response.

Terminology

TBD — http://en.wiktionary.org/wiki/TBD

Key in context of JSON message stands for name in name/value pair which is the same as key/value pair or attribute/value pair. JSON RFC uses name in such context.

Authorization

An authorization stage of communication MUST use HTTP/1.1 as an underlying transport. A method of HTTP request MUST be POST. Message MUST be HTTP message-body. Content-Type header of HTTP response MUST be application/json.

Register

The requirements for user credentials are as follows:

login: [a-zA-Z0-9]{2,36} i.e. minimal length is 2 symbols and maximum length is 36 symbols. Allowed charset is latin symbols and numbers.

password: .{6, 36} i.e. minimal length is 6 symbols and maximum length is 36 symbols. Any character is allowed except characters indexed from 0 to 31 in ASCII.

class: see Player Classes

Request

action: register
login: <new client's login>
password: <new client's password>
class: <player class name>

Response

result: one of: ok, badPassword, badLogin, badClass, loginExists

Login

Request

action: login
login: <client's login>
password: <client's password>

Response

result: one of: ok, invalidCredentials
sid: <string representation of session identifier>
webSocket: <WebSocket server URI>
id: <player ID for use with Game Interaction requests>
fistId: <fists' id for use it when no weapon equipped>

Logout

Request

action: logout
sid: <client's sid>

Response

result: one of: ok, badSid

Game Interaction

All communication other than authorization stage (with an exception of logout) is done via WebSocket protocol as underlying transport.

Common Invariants

Those rules apply to any client-server communication of Game Interaction section. Explicit inclusion of these rules may be ommited. If there is an exception from the rules, the corresponding section shall state it explicitly.

Each request message sent after client has been logged in MUST have a key sid with a string value of client sid provided by server. In case of invalid sid the result key of response MUST have a value of badSid.

In case of successful response result key of respone MUST have a value ok.

Destroy Item

Destroys an item. It is possible if distance between item and player centers doesn't exceed pickUpRadius or item is already in player's inventory. Otherwise result MUST be badId. If object with this id is not an item result MUST be badId.

If item has consumable class or expendable type (see Items) request body MUST contain amount field that specify amount of destroyed items.

When amount field contain negative value or value that is greater than count of items in stack or destroyed item is not stackable, result MUST be badAmount.

Request

action: destroyItem
id: <item's id>
amount: <items amount>

Drop

Drop item from inventory to the ground. Item's x and y now MUST be the same as client's player ones.

If dropped item has consumable class or expendable type (see Items) request body MUST contain amount field that specify amount of dropped items.

When amount field contain negative value or value that is greater than count of items in stack or dropped item is not stackable, result MUST be badAmount.

Request

action: drop
id: <id>
amount: <dropped items amount>

Response

result: one of: ok, badSid, badId, badAmount

Equip

Equips item of given id onto specified slot. If it is impossible to equip this item onto that slot badSlot is returned. If slot is already occupied then item which is already there gets automatically unequipped and takes of the item being equipped.

If item is on the ground and it is possible to Pick Up it with Pick Up request then it is also possible to try to equip this item from the ground. If the result of such equipping is ok than item is automatically gets picked up.

When equipping a weapon of subtype two-handed or bow while wielding a some item in other hand — last one MUST be unequipped automatically.

If equipping has modified contents of other slots, then response MUST contain slots field, that is mapping from modified slot to id of item, that now is equipped in this slot.

Request

action: equip
id: <item's id>
slot: <slot to equip onto>

Response

slots: { <slot> : <item id>, ... }
result: one of: ok, badId, badSid, badSlot

Examine

There are three versions of Examine. One is for single id, other is for map cell and last is OPTIONAL for multiple id's.

Single id Examine

Request

action: examine
id: <actor's identifier>

Response

id: <actor's id>
type: <actor's type. One of `player`, `monster`, `item`, `projectile`>
x: <x coordinate>
y: <y coordinate>
result: one of: ok, badSid, badId

If type is item response MUST contain the following:

item: {<Item Description*>}

See Item Description

If type is either player or monster response MAY contain the following:

health: <actor's current number of health points>
maxHealth: <actor's maximum number of health points>

mana: <actor's current number of mana points>
maxMana: <actor's maximum number of mana points>
inventory: [<item description>, ...]
stats: {<stat name> : <stat value>, ...}

Field stats MUST contain all stats, described in Stats with corresponding values.

Under various circumstances inventory field MAY be present or not. However if given id is an id of a player corresponding to client issued an Examine request and this player's inventory is not empty the field inventory MUST always be present.

If present, inventory is an array of item ids.

If type is monster response MAY contain these fields:

name: <name of a monster>
mobType: <string describing the type of a monster>

If type is player response MAY contain these fields:

slots: {<slot name>: <item description>, ...}
login: <player's login>

Rules for presence of field slots are the same as for field inventory. If slot is empty it MAY be omitted.

For possible slot names see Slots

Map Cell Examine

Request

(x, y) pair belongs to a cell. This cell is about to be examined.

action: examine
x: <x coordinate on map>
y: <y coordinate on map>

Response

Response JSON object MUST contain a single field data which is an array of objects each representing a result from single id version of Examine. data MUST contain information for all objects belonging to given cell.

data: [{<single-id-examine's result}]

Multiple id Examine

The following is OPTIONAL. Examine request could be issued to examine multiple ids. In such case id field of request is an array of item's ids. Then response is an array of objects each containing fields from Examine response for single id. e.g.

```json
"data":
[
  {
    "id": 42,
    "type": "player",
```

```
    "login": "John",
    "x": 0,
    "y": 100,
    "result": "ok"
  },
  {
    "id": 43,
    "type": "player",
    "login": "Snow",
    "x": 1,
    "y": 99,
    "result": "ok"
  },
  {
    "result": "badId"
  }
]
```

Get Dictionary

Dictionary is a JSON object describing mapping from game map cell (recieved via look action) to string value of cell type e.g.

```
"dictionary":
{
  ".": "grass",
  "#": "wall"
}
```

Request

action: getDictionary

Response

dictionary: {...}

Logout

See Logout

Look

A request for server to provide information for a map area around the client's player. Such information MUST be provided via keys map and actors.

TBD: If size of such area must be standardized

map key MUST have a value of an array of array of strings. Such strings are decoded via dictionary got using getDictionary request. map 2d array has row-major order. E.g. for a 4x6 (4 rows, 6 columns) map area:

```
"map":
[
  ["#", "#", ".", "#", "#", "#"],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
```

```
  ["#", "#", "#", ".", "#", "#"]
]
```

actors key MUST have a value of an array of objects type. Each element of the array MUST describe a single actor present at the provided area in the following form:

type: <actor's type. One of `player`, `monster`, `item`, `projectile`>
id: <actor's id>
x: <global map space x coordinate of actor>
y: <global map space y coordinate of actor>

If type is not item actor description MUST contain:

health: <actor's current number of health points>
maxHealth: <actor's maximum number of health points>
mana: <actor's current number of mana points>
maxMana: <actor's maximum number of mana points>

If type is monster actor description MUST contain these fields:

race: <string describing the race of a monster>

If type is player actor description MUST contain theese fields:

class: <string describing the class of a player>

for more information about class field see Player Classes

Request

action: look

Response

map: [[...]]
actors: [{...}, ...]
x: <global map space x coordinate of player's center>
y: <global map space y coordinate of player's center>

See that actors don't contain current player itself

Move

If total weight of items in Player's inventory exceeds this Player's carrying capacity then result is tooHeavy.

Request

action: move
direction: one of: west, north, east, south
tick: <tick number for move action>

Pick Up

Request for client player to pick up an item with given ID. Object with this ID MUST have type item and distance between player's center and item's center MUST be less or equal than constant pickUpRadius. Otherwise nothing is picked up and result is badId.

If item is contained in someone's inventory than result is badId.

If total weight of Player's inventory exceeds Player's carrying capacity after picking up an item and it is possible to pick up item then result MUST be tooHeavy.

Request

action: pickUp
id: <item-to-pick-up's id>

Tick

For each simulation tick server MUST broadcast current tick to all the clients. Tick message is neither request nor response message therefore implicit rules for keys sid, action, result don't apply to it.

Tick message is a JSON object. It MUST contain key tick with a value of server current tick number.

Tick numbers are required to grow monotonously by 1 for each tick.

Tick message MAY also contain an array events of JSON objects describing events occured at a given tick.

Server MAY decide to send some events only for a subset of clients. e.g. attack action only to ones for which it is visible.

Possible Events

Attack

If there are multiple targets for a single attack then server MUST produce multiple attack events for each target.

Field killed may be omitted if it is false.

event: "attack"
attacker: <attacker's id>
target: <target's id>
blowType: <string describing attack type e.g. "CLAW", "BITE", etc>
dealtDamage: <damage dealt>
killed: <whether target was killed or not>

Effect

x: <global-space x coordinate of effect center>
y: <global-space y coordinate of effect center>
radius: <visual effect radius>
type: <type of effect>

Unequip

Unequips item with given id. badId is returned if there is no such item equipped on the client.

Request

action: unequip
slot: <slot to be unequipped>

Response

result: one of: ok, badSlot

badSlot MUST be returned if slot field contain invalid slot specificator or doesn't exist in request.

Use

Use an object by given id. In general object can be used if it is contained in the player's inventory or equipped onto player. Objects can also be used if laying on the ground and can be picked up with Pick Up request.

Some item require certain additional data to be specified. e.g. if using equipped weapon client MUST specify x and y.

badSlot MUST be returned if using item required to be equipped e.g. using weapon present in inventory but unequipped.

If no item equipped as a weapon and player want to attack, fistId (see Login) MUST be used in request.

Request

action: use
id: <item's id>

Optional data for various items:

x: <global map space x coordinate of target point>
y: <global map space y coordinate of target point>

This section is to be completed.

Response

message: <text describing what's happened as a result of usage>
result: one of: ok, badId, badSlot, badPos

Testing

There are a number of request messages available only when server is in the testing stage. Such messages are marked with "Testing stage only." If such message to be sent while testing stage is not active, server MUST respond with "result": "badAction".

All testing API provided via web-socket. All testing requests need SID field in request body and can return badSid if SID is invalid.

Get Constants

Get a set of current constant values.

Request

action: getConst

Response

result: ok
playerVelocity: <value>
slideThreshold: <value>
ticksPerSecond: <value>
screenRowCount: <value>
screenColumnCount: <value>
pickUpRadius: <value>

Put Item

Testing stage only.

Request

action: "putItem"
x: <x coordinate of item's center>
y: <y coordinate of item's center>
item: {<Item Description*>}

See also:

- Item Description

Response

result: ok, badPlacing, badItem
id: <item's id>

Put Mob

Testing stage only.

Put specified mob onto the level map.

Request

action: "putMob"
x: <mob's x coordinate>
y: <mob's y coordinate>
stats: {<Stats*>}
inventory : [{<Item Description*>}, ...]
flags: [<Flags*>, ...]
race: <Race*>
dealtDamage: <mob's dealt damage by single attack>

Field dealtDamage MUST exist in request. This field is specified by string of the form NdM, where N is count of verges dice and M is count of dice pops.

See also:

- Stats
- Item Description
- Flags

- Race

Response

id: <mob's id>
result: one of: ok, badPlacing, badFlag, badRace, badInventory, badStats, badDamage

Put Player

Testing stage only.

Put player instance onto the level map, specifying inventory, slots and stats. Player has CAN_MOVE and CAN_BLOW flags by default.

Request

action: "putPlayer"
x: <player's x coordinate>
y: <player's y coordinate>
inventory: [{<Item Description*>}, ...]
slots: {<Slot name. Slots*> : {<Item Description*>}, ...}
stats: {<Stats*>}

See also:

- Stats
- Item Description
- Slots

Response

sid: <player's session id>
id: <player's id>
fistId: <player's fist id>
inventory: [<items' id generated in inventory>]
slots: {<Slot name. Slots*> : <items' id generated for slot>, ...}
result: one of: ok, badPlacing, badInventory, badSlot, badStats

If there was not items in inventory/slots field in putPlayer request then corresponding fields in response MAY be omitted.

Enforce

Testing stage only.

This request tell server perform some player's action.

Request

action: "enforce"
enforcedAction: <object that represent any action of player>

Response

actionResult: <object that represent result of perfoming requested action>
result: one of: ok, badSid, badEnforcedAction

Set Location

Testing stage only.

Set x, y coordinates of player.

Request

action: "setLocation"
x: <desired player's x coordinate>
y: <desired player's y coordinate>

Response

result: on of: ok, badPlacing, badSid

Set Up Constants

Testing stage only.

Upload a set of constants for a server to immediately set up. There is a reference set of constants for the purposes of cross server testing:

```
{
  "action": "setUpConst",
  "playerVelocity": 1.0,
  "slideThreshold": 0.1,
  "ticksPerSecond": 60,
  "screenRowCount": 7,
  "screenColumnCount": 9,
  "pickUpRadius": 1.5
}
```

Request

action: setUpConst
playerVelocity: <portion of tile player travels through the world per second>
slideThreshold: <portion of tile which gets ignored when moving towards it>
ticksPerSecond: <number of simulation cycles per second>
screenRowCount: <number of tile rows in a rectangle get via `look`>
screenColumnCount: <number of tile columns in a rectangle get via `look`>
pickUpRadius: <maximum distance between player's center and item's one for latter to be picked up>

Response

result: one of: ok, badAction

Set Up Map

Testing stage only.

Upload map to server and set it up as currently active. If server assumes uploaded map to be invalid (e.g. malformed map array or wrong cell value) then the request MUST be responded with badMap. Innermost values of map key are those found in Dictionary.

Minimal size of map is 1x1. Empty map is badMap.

Request

```
{
   "action": "setUpMap",
   "map": [
   [<column count number of values>],
   [<column count number of values>],
   [<column count number of values>],
   ... <total of row count arrays with row values>
   [<column count number of values>]
   ]
}
```

Response

result: one of: ok, badMap, badAction

Start Testing

This request MUST be sent each time at the beginning of testing stage. Once this message is responded with "result": "ok", it is valid to state that testing stage is now active.

It is invalid to request Start Testing when testing stage is already active. In such case request MUST be answered with "result": "badAction".

Request

action: startTesting

Response

result: one of: ok, badAction

Stop Testing

Testing stage only.

Each testing stage MUST be closed with this request. Once responded with "result": "ok" it is valid to state that server is no more in the testing stage.

Request

action: stopTesting

Response

result: one of: ok, badAction

Data Invariants

This section describes involved data.

All units in game are measured in tiles, assuming one tile width and height to be 1.0. Upscaling this for rendering is up to client.

Game Objects

Game Objects are the contents of actors array of look response. Items in inventory of creatures are also game objects. Hovewer those don't appear in look's actors.

The type of game object may be one of:

- player
- monster
- item
- projectile

All of those MUST have square shape. Player's and monster's size MUST be 1. Item's and projectile's size MUST be 0.

Player

Classes

- warrior
- rogue
- mage

Slots

- left-hand - one-handed weapon or shield slot
- right-hand - one-handed weapon or shield slot
- ammo - slot for weapon expendables. e.g. arrows for bow.
- left-finger - ring
- right-finger - ring
- neck - amulet
- body - armor
- head - helmet
- forearm - gloves
- feet - boots

Monster

name : <monster's name>
x: <mob's x coordinate>
y: <mob's y coordinate>
stats: {<Stats*>}
race: <Race*>

Monsters have a simple logic realized by flags (see Flags). Describe main concepts of monsters' behavior logic.

1. If monster found target, it has to go to it.
2. If monster gets attacked by player, it MUST get player as new target.
3. If monster gets attacked by another monster, first one has to options:

- if it was fighting with another monster, one MAY switch its target to attacker
- if it wasn't, one MUST gets attacker as new target

Item

Every item MUST have a weight. Number of items in inventory and slots of any Creature (Monster or Player) MUST be limited by maximum weight Creature can handle.

If weight of item to be picked up exceeds maximum Player's weight then pickUp request MUST result in tooHeavy.

If total weight of items in inventory of any Creature somehow exceeds its carrying capacity then Creature becomes immobilized. If Player is to be immobilized then move request MUST result in tooHeavy.

Item Description

Some requests from Testing section MAY induce a need to describe an item. Examine returns itemData in the same form.

Item description is a set of JSON fields:

```
name: <item's name>
weight: <item's weight>
class: <Item Class*>
type: <Item Type*>
subtype: <Item Subtype*>
bonuses : [{<Bonus Description*>}, ...]
effects : [{<Effect Description*>}, ...]
```

Based on the context those fields may be wrapped in JSON object.

Bonus application order is determined via order of bonuses elements. Same applies to effects.

Field subtype MUST be omitted if there is no subtypes specified for item type. Otherwise it MUST be present.

No shield can be equipped with bow or two-handed. arrows can only be equipped with bow.

Item Class

- garment
- consumable

Item Type

types for garment class:

- amulet
- ring
- armor
- shield
- helm
- gloves

- boots
- weapon
- expendable

Item Subtype

subtypes for weapon type of garment class:

- one-handed
- two-handed
- bow

subtypes for expendable type of garment class:

- arrows

Bonus description

Only equipment may have bonuses.

TBD: REFORMULATE section

Json-object represent bonus description. There is a one kind of bonuses in Angband: constant bonus to some stat. Our team added something like it: bonus to any stat of active object (mob, player), but it can be calculated as percent bonus. So, we have two kinds of bonuses: constant bonus and percent bonus. Calculation for constant bonus: += Calculation for percent bonus: += * Value of bonus can be a negative.

stat: <stat modified by bonus>
effectCalculation: <const|percent>
value: <value of bonus>

Effect description

TBD: REFORMULATE section

Json-object respresent effect description. Effects in Angband are something hazy, so I invented my own. Effect is something, that modified some only stat during some time. There is two kinds of effects: OnGoingEffect (modify stat continuously with some fixed value) and BonusEffect (add some bonus).

Ongoing Effect Description

stat : <stat modified by effect>
duration : <effect duration in seconds>
type: "ongoing"
value: <value of effect>

Bonus Effect Description

duration: <effect duration in seconds>
effectType: "bonus"
bonus: <bonus description>

Projectile

name: <projectile's name>

type: 'projectile'
x : <projectile's x coord>
y : <projectile's y coord>

Race

Both players and monsters have race. Player always have PLAYER race.

Possible races are:

- ORC
- EVIL
- TROLL
- GIANT
- DEMON
- METAL
- DRAGON
- UNDEAD
- ANIMAL
- PLAYER

Races aree taken from Angband (except for the PLAYER race).

Flags

The following flags are available:

- CAN_MOVE
- CAN_BLOW
- HATE_<race>

Angband has many flags. Of our flags CAN_MOVE and CAN_BLOW have relation to Angband's NEVER_MOVE and NEVER_BLOW ones.

TBD: REFORMULATE the following: If mob hasn't flag NEVER_MOVE (NEVER_BLOW), it has CAN_MOVE (CAN_BLOW) flag by default. Otherwise, if mob has never-flag in angband, it hasn't corresponding flag in our MMO.

Angband have no HATE_<race> flag. E.g. HATE_ORC, HATE_PLAYER etc.

TBD: describe thoroughly what does it mean to HATE_<race> etc

Stats

Both players and monsters have stats. Stat describes to what extent a character possesses a natural, in-born characteristic.

Stats in Angband:

- STR
- INT

- WIS
- DEX
- CON
- SPEED
- STEALTH
- SEARCH
- LIGHT
- TUNNEL

Our game does not implement all of those. Instead we are using the following stats:

- STRENGTH (STR)
- INTELLIGENCE (INT)
- DEXTERITY (DEX)
- SPEED (SPEED)
- DEFENSE
- MAGIC_RESISTANCE
- CAPACITY
- HP
- MAX_HP
- MP
- MAX_MP

Some requests from Testing section require to specify stats field. Stats field MUST be JSON object containing zero or more of the aforementioned stats.

If any particular stat as well as the whole stats field of request is not required for testing it MAY be omitted.

TBD: rework stats and describe which stats affect which.