Современные средства разработки ПО

Введение. Состав курса. Сложное ПО. C++. ООП.

Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru

Суть курса

- Дать основные приёмы при создании:
  - сложного ПО,
  - качественного ПО,
  - в условиях гибкой разработки (Agile).
- Введение в современные технологии
  - с акцентом на тематику нашей кафедры (распределённые программно-аппаратные системы),
  - начальные знания по параллелизму, как основе распределённых систем.

Что такое «сложное ПО»?

Основные признаки сложного ПО

- **Комплексность**,
  - то есть, собственно, сложность, которая проявляется в большом количестве объектов, их вложенности друг в друга и связями между ними
- **Длительный жизненный цикл**,
  - то есть, необходимость не только разработать ПО, но и поддерживать его развитие на протяжении длительного времени
- **Работа в команде**,
  - что подразумевает невозможность разработать и поддерживать сложное ПО в одиночку за приемлемое время, а также устойчивость проекта в условиях текучки команды разработки

Дополнительные признаки сложного ПО

- **Многопоточность**,
  - усугубляет сложность за счёт необходимости согласования параллельных потоков
- **Многопроцессность**,
  - усугубляет сложность за счёт необходимости введения протоколов взаимодействия процессов
- **Распределённость**,
  - усугубляет сложность за счёт необходимости введения протоколов взаимодействия узлов

Примеры сложных проектов

Проблемы при разработке сложного ПО

- Нечёткие или меняющиеся требования
- Большое количество взаимодействующих плохо разделяемых и/или нечётких понятий предметной области
- Необходимость периодического внесения изменений в исходный код ПО
- Большое количество разработчиков

Каким образом можно компенсировать эти проблемы с минимальными издержками

- Техники формирования устойчивых к изменениям архитектуры, кода и всего проекта(*адаптивная архитектура*, *адаптивный код*).
- Методологии, позволяющие повысить качество определения и контроля требований на всём жизненном цикле разработки (*Enterprise*: MSF, RUP и др.) или сокращающие время итерации жизненного цикла, привлекая заказчика к проектированию и верификации (*Agile*: SCRUM и др.)
- Автоматизация разработки, коллективной разработки (*CI/CD/CD*).
- Инструменты, поддерживающие и объединяющие эти и другие решения (*автоматические тесты, автоматическое документирование и др.*).
- Другие техники (*метрики проекта, соглашения о стиле кодирования*, регламенты проектов и др.).

Состав курса

1. Современные принципы применения ООП
   - Л: 14ч., С: 7ч., ЛР: -, СР: 11, РК
2. Паттерны проектирования и принципы SOLID
   - Л: 10ч., С: 5ч., ЛР: 2, СР: 18, РК, ДР
3. Основы параллельных вычислений
   - Л: 10ч., С: 5ч., ЛР: 2, СР: 17, РК
4. Экзамен (СР: 30)
   - Три вопроса – по одному для каждого модуля. Два устных и задача на диаграмму или код на C++

Оценка результатов обучения

Неделя	Форма контроля	Оценка мин	Оценка макс
7	РК	12	20
12	ДЗ	3	5
	РК	12	20
17	РК	15	25
	Экзамен	18	30
	ИТОГО	60	100

Лабораторные и домашняя работы

М	Практикум	Зависимость	«Команда»	Отчёт
2	ДР Использование парадигмы ООП	-	-	CI
2	ЛР №1 Изоляция предметной области	-	+	CI
2	ЛР №2 Изменение в предметной области	ЛР №1	+	CI
3	ЛР №3 Нагрузочное тестирование	ЛР №2	+	CI
3	ЛР №4 Многопоточность	ЛР №3	+	CI

Литература (общая)

1. Фаулер М. UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с.
2. Бек Кент, Фаулер Мартин, Брант Джон. Рефакторинг. Улучшение проекта существующего кода. – Вильямс. 2017
3. Стив Макконел. Совершенный код. Мастер-класс. — М. : Издательство «Русская редакция», 2010. — 896с.
4. C++ reference. - URL: https://en.cppreference.com
5. Портал C++. - URL: http://www.cplusplus.com
6. Методичка по данному курсу

Литература (модуль 1)

1. Мейер Бертранд. Почувствуй класс. — М.: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011. —775 с.
2. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — 304 с.
3. Герб Саммер, Андрей Александреску. Стандарты программирования на C++. 101 правило и рекомендация. – Вильямс. 2017. – 224с.
4. Meyer, Bertrand. Object-Oriented Software Construction, second edition. - Prentice Hall, 1997

Литература (модуль 2)

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368 с.
2. Гэри Маклин Холл. Адаптивный код: гибкое кодирование с помощью паттернов проектирования и принципов SOLID. – Вильямс, 2017
3. Роберт С. Мартин. Гибкая разработка программ на Java и C++. Принципы, паттерны и методики. – Вильямс, 2016
4. Андрей Александреску Современное проектирование на C++: Обобщённое программирование и прикладные шаблоны проектирования. - Addison-Wesley, 2001 (оригинальное издание) [Loki]

Основные инструменты

- C++17
- UML 2.0 (классы, последовательности, пакеты)
- GNU/Linux
- Git/GitLab: https://bmstu.codes
- Рекомендации:
  - GNU/Linux: Ubuntu 20.04 LTS или Alt Linux p10 K
  - Visual Studio Code или Qt Creator
  - GCC (g++) или clang

Расширения Visual Studio Code

- C/C++ Extension Pack
  - Microsoft. *Popular extensions for C++ development in Visual Studio Code.*
- Spelling Checker for Visual Studio Code (+Russian)
  - Street Side Software. *A basic spell checker that works well with camelCase code*

Почему C++

- C++ позволяет писать адаптивный код
- Высокоуровневый ЯП с хорошей поддержкой ООП
- Высокоэффективный высокоуровневый ЯП
- Широко распространён
- Активно развивается, особенно, начиная со стандарта 11
- Наиболее открытый ВЯП, т. е. содержит минимальное количество встроенных конструкций
- C++ позволит заглянуть в детали реализации некоторых конструкций и понять различия их реализации в других ЯП
- Вы его проходили

Светлая и тёмная стороны C++

- Один из постулатов философии C++: «Если разработчик хочет выстрелить себе в ногу, ЯП не будет ему мешать».
- Эти и другие особенности C++ делает его мощнейшим инструментом разработки, но для его использования необходимо иметь самодисциплину и соблюдать некоторые приёмы.
- Можно сказать, что C++ имеет две стороны:
  - светлая и тёмная,
  - доступная и скрытая для большинства,
  - доктор Джекил и мистер Хайд C++,
  - Гарри Поттер и Воландеморт C++.
- Наш путь — по светлой стороне C++, но
  - иногда, балансируя на границе между светом и тьмой, мы будем осторожно изучать тёмную сторону C++, рассматривать компромиссы и оценивать потерю эффективности, в том числе, в сравнении с другими ЯП

Тёмная сторона C++, основные элементы

Тёмная сторона	Светлая сторона
Указатели	Интеллектуальные указатели STL C++11: **`std::unique_ptr`**, **`std::shared_ptr`**, **`std::weak_ptr`**; передача по значению, ссылки
`new` / **`delete`**	Интеллектуальные указатели STL C++11, методы **`std::make_unique`** и **`std::make_shared`**
Ссылки	Интеллектуальные указатели STL C++11 (частично), передача по значению, семантика перемещения, ссылки
Массивы	Контейнерные классы STL C++11, ссылки
Оператор **`goto`**	Не используем.
Глобальные переменные	Объекты, передаваемые в параметрах, паттерн Одиночка
Макросы	Используем спецификаторы **`const`** и **`constexpr`** при объявлении переменной
Макрос **`NULL`**	Не используем. Ключевое слово **`nullptr`** в C++11

Тёмная сторона указателей и ссылок C++

- Может быть NULL (`nullptr`)
  - **крах программы**
- Можно забыть инициировать
  - **неопределённое поведение, крах программы**
- Может указывать на уже несуществующий объект
  - **неопределённое поведение, крах программы**
- Можно забыть выделить память
  - **неопределённое поведение, крах программы**
- Можно забыть освободить память
  - **утечка памяти**

Идиома RAII

- RAII — Resource Acquisition Is Initialization (бук. «получение ресурса есть инициализация»)
- RAII — Использование механизма контроля зоны видимости объекта для управления ресурсами, которые он содержит
- Самостоятельно повторить/изучить:
  - как работает (принцип работы),
  - чем RAII отличается от «сборки мусора».

Что ещё необходимо самостоятельно повторить/изучить из C++

- Умные указатели стандартной библиотеки C++
  - unique_ptr и функцию make_unique,
  - shared_ptr и функцию make_shared.
- Обобщённые контейнеры стандартной библиотеки C++
  - array, vector, list, set, map.
- Реализация динамического полиморфизма с использованием C++

ООП

- Доминирующая парадигма при разработке сложного ПО
- Позволяет проектировать адаптивный код
- Имеет широкое распространение
- Хорошо описана во многих источниках
- Вы её проходили

Определение объектно-ориентированного программирования

- **Объектно-ориентированное программирование** (ООП) — парадигма программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию абстрагирования

  (немного изменил определение Гради Буча)

- Нужно уметь ответить:
  - Что такое «парадигма программирования»?
  - Что такое «объект»?
  - Что такое «класс»?
  - Что такое «иерархия абстрагирования»?

Основные принципы ООП

- **Абстрагирование**
  - *упрощение* — выделение значимой информации и исключение из рассмотрения незначимой;
  - *обобщение* — форма превращения понятия путём мысленного перехода от частного к общему в некоторой модели мира, что обычно соответствует и переходу на более высокую ступень абстракции.
- **Инкапсуляция**
  - — свойство, позволяющее объединить данные и методы, работающие с ними, в классе.
- **Наследование**
  - — свойство, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.
- **Полиморфизм**
  - — свойство, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

Уточнение определений

- «Объект» = «Экземпляр класса»
- «Инкапсуляция» ≠ «сокрытие»
- **Сокрытие** — принцип проектирования, заключающийся в разграничении доступа различных частей программы к внутренним компонентам друг друга

Состояние объекта

- **Состояние объекта** — совокупность значений (состояний) членов класса и состояний базовых классов
- **Изменение состояния объектов** — изменение значения (состояния) любого члена класса или состояния базового класса
- **Некорректное / недопустимое состояние объекта** — недопустимая комбинация значений (состояний) членов класса и / или состояний базовых классов

Инвариант класса

- **Инвариант класса** — утверждение, определяющее непротиворечивое состояние объектов этого класса
- **Нарушение инварианта класса** — объект класса имеет некорректное состояние
- **Способы контроля инварианта класса** — проверка инварианта в случаях, когда объект мог изменить своё состояние
  - см. «контрактное программирование», Бертран Майер

Нарушение инварианта класса

- Приводит к нарушению целостности кода, который использует данный класс
- Приводит к недопустимому состоянию классов, которые его используют
- А значит к нарушению работы всей программы

Контроль инварианта класса на языке D

```
class Date {
  int day;
  int hour;

  invariant() {
    assert(1 <= day && day <= 31);
    assert(0 <= hour && hour < 24);
  }
}
```

- Метод `invariant` вызывается каждый раз при потенциальном изменении состояния объекта
- (В чем отличие от C++?)

Вариативность состояния класса

- способность объекта заданного класса изменять своё состояние без нарушения инварианта при выполнении стандартных операций над объектами (копирование, параллельный доступ и т.д.)

Типы вариативности состояний класса

- **Неизменяемый (immutable)** — объект не изменяет своего состояния после создания
  - для таких объектов инвариант достаточно проверить в конструкторе
- **Копируемый** — при копировании объект не нарушает своего состояния:
  - при создании (конструктор копирования),
  - при передаче в качестве параметра методу (конструктор копирования),
  - при присваивании (оператор присваивания).
- **Некопируемый** — объект может быть только переносим без нарушения своего состояния

Вопросы?

Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru