C++企业软件开发实践

实验1–高质量编程基础

HUAWEI

前言

欢迎参加C++企业软件开发实践课程。本课程旨在通过完整开发案例，分享企业软件开发中的实践、经验和要求，帮助在校学生提升软件开发技能，养成良好的软件开发习惯。

实践课程共有4次实验，本课程为实验1，您将学习高质量编程的基础知识，包括：

- 开发者测试：掌握确保代码正确性和需求完整性的测试方法
- 命名：了解提高代码可读性的命名原则、规范和最佳实践
- 代码版本管理：学习使用git进行代码版本管理的规范和实践

期待您在课程中的精彩表现！

目标

通过本课程的学习，您将能够：

- 掌握测试的基本理论和方法
- 实践有效的开发者测试技术
- 掌握使用git进行代码版本管理的规范和实践

了解C++编程的可读性要求，建立高质量编程的意识，养成良好的开发习惯

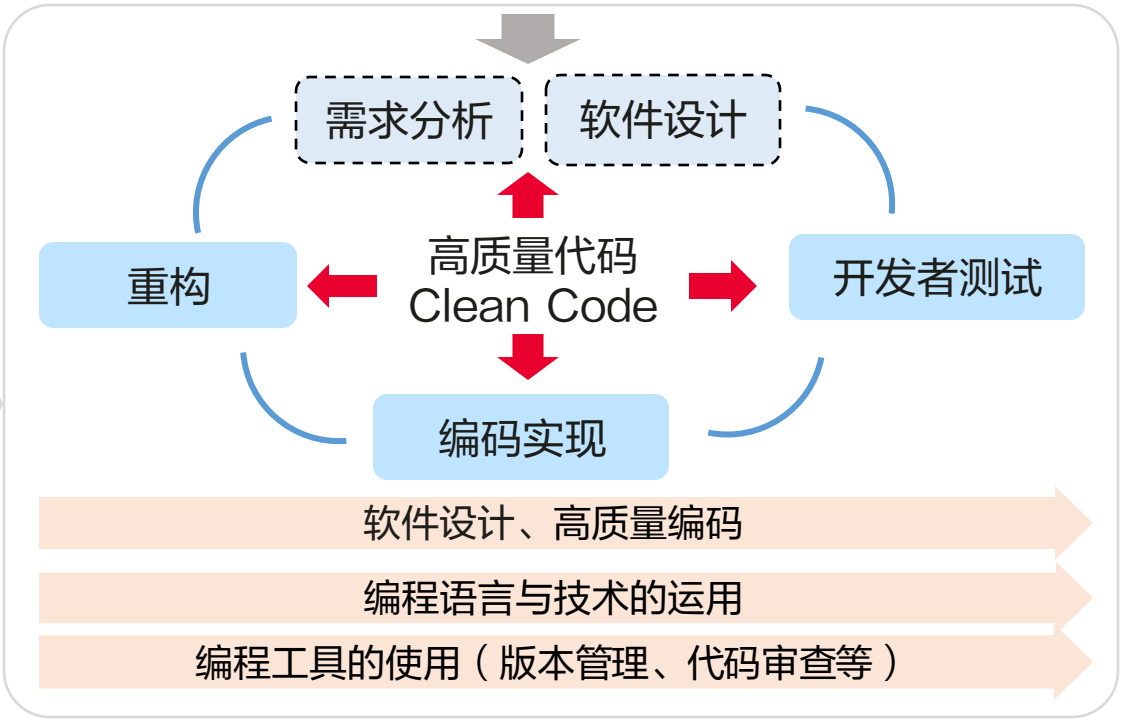
目录

企业软件开发的特点

- 企业高质量产品对软件开发的要求：企业软件规模大、生命周期长、可靠性要求高，软件代码不光满足功能和性能要求，还需要满足代码质量和安全要求，软件开发人员需要掌握良好的软件开发技能，遵循企业软件开发规范，应用软件代码实现的最佳实践
- 《企业软件开发实践》系列课程通过完整的需求开发，帮助学生提升满足企业软件开发要求的软件实践能力：
  - 企业软件开发工程化思维，软件开发不仅仅是编写代码，还包括版本管理、开发者测试、代码审查、自动化构建等；
  - 语言编程实践动手能力，遵循编码规范，合理运用语言特性，学习编写高质量代码（可读、可维护、安全、高效等）的方法，养成良好的编程习惯。

业务特点	对软件技术的诉求	高质量代码要求
质量要求高	1. 可靠性、韧性要求高，异常处理完善 2. 并发性能高，资源利用率高；	安全、 可靠性、 高效
客户众多， 全球化分布	1. 适应不同的环境和场景； 2. 快速地响应不同用户的差异化需求；	可维护-可修改/ 扩展、 可复用、 可测试
产品生命周期长，迭代演进	1. 长期开发和运营，需要不断地进行迭代和优化，以适应市场的变化，保持产品的竞争力； 2. 软件规模大，需要能够方便地进行修改和扩展，能够有效地进行复用和测试，能够高效地进行维护和更新；	可读、 可维护、 可复用、 可测试

综合实践能力



HUAWEI

课程整体框架

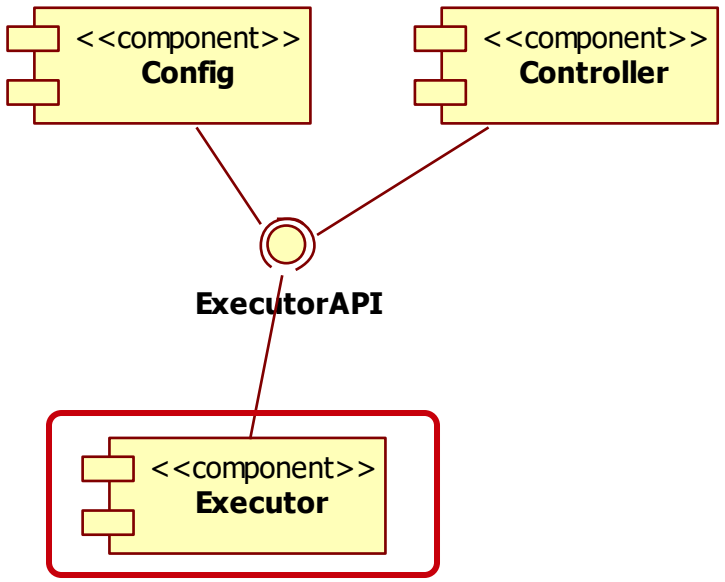
Level1 课程模块	Leve2 课程目录	学时 （h）	Leve3 主要内容	内容要点
实验1	高质量编程基础	4	开发者自测试	如何确保代码的正确性和需求的完整性
			命名	代码可读性的原则、规范、实践
			代码版本管理	使用git进行代码版本管理的规范和实践
实验2	面向对象编程	4	封装	掌握面向对象编程的基本概念，包括类和对象的封装，以及如何通过封装实现代码的复用性
			继承与多态	掌握面向对象中，继承、多态等特性，实现代码的良好扩展性
实验3	面向对象编程	4	表驱动和STL容器	有效地使用STL容器和智能指针，提升代码的可扩展性和灵活性
			解耦循环依赖	如何通过分层设计和抽象化来有效地解耦代码中的循环依赖
	函数式编程		函数式编程	掌握C++新特性带来的代码简洁性收益，写出简洁优雅的代码
			对象生命周期安全	关注C++使用过程中编程陷阱可能导致的问题，对背后原理究根，学习C++特性整体体系
实验4	综合实践	4	面向接口编程	通过C++泛型编程技术，提升代码效率和可扩展性
			任务编排	将复杂问题分解为原子操作，逐步实现，提升代码扩展性
			目录结构划分	理解和应用物理设计原则，优化代码结构和组织
课后作业	课程内容巩固	0.5~4	课程内容巩固	all

课程大纲

Level1 课程模块	Leve2 课程目录（小节）	学时（h）	Leve3 主要内容	内容要点	学时（h）	教学目的	案例		
							需求描述	演练考察点	实训前知识点储备
实验1	实验1-高质量编程基础	4	高质量测试用例构建	接口确定，建立高质量测试防护网	2	掌握如何在程序设计中： •应用测试的基本理论和方法 •实践有效的开发者测试技术	需求1 控制指令	运用正交分解设计UT用例	gtest、cmake
			命名	代码可读性的原则、规范、实践和评估。	1.5	了解C++编程可读性基本要求，建立高质量编程的基本意识，养成良好的开发习惯。		代码可读性	static、switch、for、namesp noexcept、Initialization、= 、explicit、PIMPL、std::tie
			代码版本管理	git代码版本控制	0.5	了解使用git进行代码版本管理的规范和实践		代码版本控制	git
实验2	实验2-面向对象编程	4	封装	C++中的类和对象，包括封装、构造函数、析构函数、成员变量、成员函数等	1.5	掌握C++面向对象编程的基本概念和优势，主要包括类和对象的封装，以及如何通过封装实现代码的复用性。	需求2 支持加速功能	Executor组件面向对象封装实现	final、const、noexcept
			继承与多态	C++中的继承和多态，包括虚函数、抽象类、派生类、重写、重载等	2.5	掌握C++面向对象中，继承、多态等特性，实现代码良好的可扩展性。		通过OO继承和多态，MLR指令处理统一	overvide、=default、=delete
实验3	实验3-面向对象编程	4	表驱动	STL容器、智能指针	0.5	掌握C++STL库的融合使用，利用表驱动方法减少复杂循环，结合多态特性提升代码灵活性和可扩展性	需求3 支持倒车功能	消减指令分支处理	std::unordered_map、std::un
			解耦循环依赖	分层设计和抽象化	1	通过数据抽离和代码分层策略，解耦循环依赖，显著提升代码的可扩展性和可维护性		通过状态数据的抽离，解决循环依赖问题	
			状态抽象提升可读性	状态抽象和计算属性	0.5	通过状态抽象，简化复杂状态流转问题，将指令数字化处理，提高代码的可维护性和可扩展性		业务问题抽象建模	
	实验3-函数式编程		函数式编程	函数式编程、lambda	1.5	掌握C++新特性带来的代码简洁性收益，写出简洁、优雅的代码。		简化指令子类实现的代码	std::function、lambda
			对象生命周期安全	lambda封装、生命周期安全	0.5	关注C++使用过程中编程陷阱可能导致的问题，对背后原理究根，学习C++特性整体体系。		消减lambda生命周期安全风险	操作符重载
实验4	实验4-综合实践	4	面向接口编程	模板类：基于模板的单体类	2	通过泛型编程技术，提升可扩展性	需求4 支持掉头功能	基于接口编程，策略表驱动归一管理，提升效率及代码可扩展性；同时通过using特性，提升代码可读性	template、=delete、std::lis using
			任务编排	抽象、分层	1	掌握复杂问题处理的思维		将复杂问题分解为原子操作，实现任务编排，提升代码扩展性	
			目录结构设计	工程文件管理	1	树立有效管理工程文件的意识，掌握目录结构设计的原则，学习企业实践		按功能划分，确保独立和职责明确，清晰结构，便于理解和维护	
课后作业	课程内容巩固	0.5~4	课程内容巩固	all	0.5~4	课后作业，巩固实训编程各知识点	课后作业 支持不同的车辆类型		enum class

目录

课程实训案例引导



ADAS系统介绍

Advanced Driver Assistance System，ADAS是辅助驾驶员安全驾驶车辆的系统，可以增加车辆安全和道路安全，同时可以明显减轻汽车驾驶的操作负担。

ADAS Executor组件主要功能：

- 由Config组件进行初始化及配置
- 接收Controller组件的各种移动控制命令（如前进、转向等），维护车的位置状态

课程实训需求1 支持基本控制指令

设计一个C++程序，模拟自动驾驶车辆的执行器Executor组件功能。

Executor组件提供初始化接口，负责将车初始化在指定位置，输入数据为:(x, y, heading)，其中：

- x,y对应地点位置（均为int32类型）
- heading对应四个方向（N、S、E、W）(均为char类型)

Executor组件可以执行如下的移动指令：

- M: 前进，1次移动1格

Executor组件可以执行如下的转向指令：

- L: 左转90度，位置不变
- R: 右转90度，位置不变

Executor组件提供获取车当前的坐标位置和朝向接口，如果Executor未被初始化位置，则接口默认返回(0,0,N)。

X轴移动的方向为EW方向，Y轴移动的方向为NS方向。

要求

- 设计Executor组件对外的接口，实现上述功能
- 设计测试用例，构建上述功能的测试防护网

约束

- 整个过程中坐标(x,y)的值都在int32范围内，不考虑整数溢出场景
- 调用者保证了传给Executor接口的参数都是合法的，不考虑非法参数场景

需求示意图

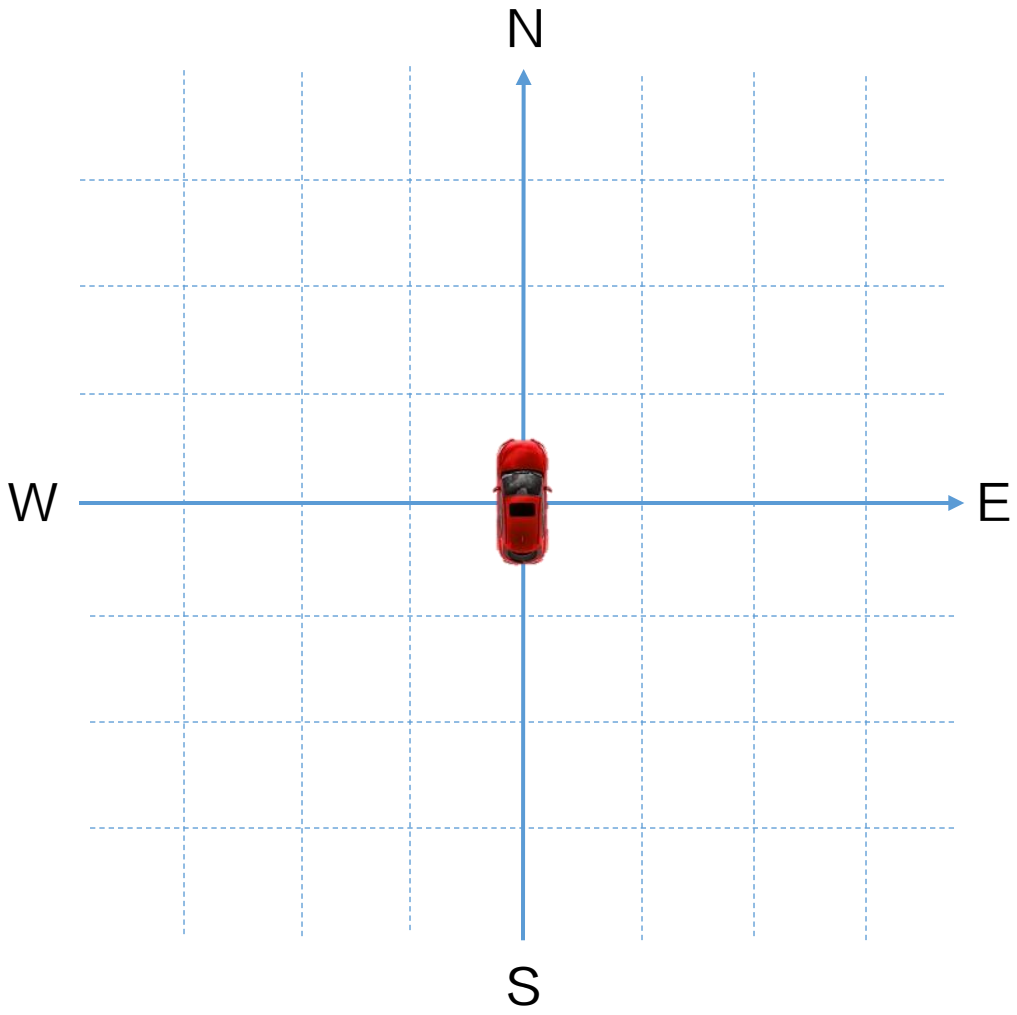
一辆车在二维坐标平面上行驶，车的位置始终为整数表示的坐标点，朝向有4种。

车的控制系统通过以字母表示的指令控制汽车。

指令序列：　MLMMRM

当前位置：　(-2,0)

当前朝向：　W



目录

Executor需要对外提供接口，接口如何设计?

让我们来思考一下这个问题：

- **需要定义哪些接口?**
- **接口参数、返回值分别是什么?**

请将上述问题的思考表达出来，例如:

接口	功能	参数	返回值
**接口	***	***	***
**接口	***	***	***

课程实训需求1 支持基本控制指令

设计一个C++程序，模拟自动驾驶车辆的执行器Executor组件功能。

Executor组件提供初始化接口，负责将车初始化在指定位置，输入数据为：(x, y, heading)，其中：

- x,y对应地点位置（均为int32类型）
- heading对应四个方向（N、S、E、W）(均为char类型)

Executor组件可以执行如下的移动指令：

- M: 前进，1次移动1格

Executor组件可以执行如下的转向指令：

- L: 左转90度，位置不变
- R: 右转90度，位置不变

Executor组件提供获取车当前的坐标位置和朝向接口，如果Executor未被初始化位置，则接口默认返回(0,0,N)。

X轴移动的方向为EW方向，Y轴移动的方向为NS方向。

一起来通过对需求的动名词分析（Verb-Noun Analysis），理解分解需求。
蓝框：动词，接口
红框：名词，数据

接口设计

通过动名词分析，已明确对应的接口及其操作的数据：

接口	功能	参数	返回值
初始化接口	负责将车初始化在指定位置	x, y, heading	NA
指令执行接口	执行移动、转向指令	指令字符串	NA
获取位置和朝向接口	获取当前的坐标位置和朝向	NA	x, y, heading

讨论（10分钟）

现需将这些接口及数据用C++代码实现，该如何实现？即这些接口如何命名？

目录

命名规范

- 使用正确的英文单词并符合英文语法，不要使用拼音
- 仅使用常见或领域内通用的业务术语（Domain Language）的单词缩写
- 命名风格保持统一，推荐使用驼峰命名风格:

类别	命名风格	形式
命名空间，类类型，结构体类型，联合体类型，枚举类型，typedef定义的类型，类型别名	大驼峰	AbbBbb
函数（包括成员函数，作用域内函数，全局函数）	大驼峰	AbbBbb
全局变量（包括全局、文件、namespace域下的变量以及相应作用域下的静态变量）	带’g_’前缀的小驼峰	g_aaaBbb
类成员变量	小驼峰	aaaBbb
局部变量，函数参数，宏参数，结构体和联合体中的成员变量	小驼峰	aaaBbb
枚举值，常量	全大写，下划线分割	AAA_BBB

减少符号依赖

- 尽量避免使用”extern”关键字，除非引用某些汇编或者编译器生成符号
- 不要在头文件中使用匿名namespace或static定义非外部可见符号
- 尽可能多使用private和匿名命名空间隐藏符号
- 使用P-IMPL手法，将头文件中无需暴露的细节转移到实现文件中

匿名命名空间是一种特殊的命名空间，它没有名字。匿名命名空间的主要作用是限制其作用域，使得其中定义的符号（例如变量、函数、类等）仅在定义它们的翻译单元（即源文件）中可见。这样可以有效地避免命名冲突并防止符号的外部链接。

P-IMPL：Pointer to Implementation是一种用于隐藏类的实现细节的技术。通过将实现细节放在一个单独的类中，并通过一个不透明的指针来访问它，可以减少编译依赖性并提高封装性。

头文件设计原则

- 自满足原则：是指C++头文件应该能够被独立地解析和理解，不依赖于任何源文件或其他头文件的内容。头文件要独立可编译通过（头文件通过放到自身实现文件第一行可自行验证自满足性）
- 最小依赖原则：头文件在可独立编译的前提下基础上尽可能少依赖其它头文件和符号（尽量弱符号依赖）
- 最小可见原则：头文件暴露最小信息，不让客户看到不需要的信息
  - 接口隔离：讲头文件按照不同客户进行拆分，避免上帝头文件（God's single include file）
  - 将客户无需看到的宏、类型、静态私有成员和函数搬迁到实现文件
  - 减少头文件中不必要的inline函数
  - 适可使用PIMPL技巧隐藏更多的头文件中信息
- 接口隔离原则：面向不同客户的接口拆分到不同文件中，避免不同客户因共享头文件产生不必要的耦合

命名实践：接口定义

```
#pragma once 防止头文件被多次包含，简化代码，避免重复定义和编译错误
#include <string>
namespace adas                  namespace：组织代码并防止命名冲突，使代码更易于管理和阅读
{
struct Pose {                   在计算和计算机视觉领域，pose (或 spatial pose) 表示物体的位置和方向
   int x;
   int y;
   char heading;
};                              定义结构体，便于数据的组织和传递

class Executor抽象类，提供接口定义，隐藏功能实现细节
{
public:
   // Caller should delete *executor when it is no longer needed.
   static Executor* NewExecutor(const Pose& pose = {0, 0, 'N'}) noexcept    静态成员函数，无需实例化即可调用，工厂方法模式

public:
   Executor(void) = default;                                  =default简化代码，明确表示使用编译器生成的默认实现
   virtual ~Executor(void) = default;

   Executor(const Executor&) = delete;                            =delete防止对象的拷贝
   Executor& operator=(const Executor&) = delete;

public:
   virtual void Execute(const std::string& command) noexcept      = 0;
   virtual Pose Query(void)    const noexcept      = 0;           noexcept指定函数不会抛出异常，提高代码的安全性和性能，便于编译器优化
};
} // namespace adas
```

好的命名规范和合理应用C++特性，提供简洁、清晰的接口约定，同时确保了代码的安全性和可维护性

目录

思考

需求已澄清，接口已经定义，接下来的开发过程中，我们如何确保需求的完整性和代码的正确性?

开发者测试是确保代码正确性和需求完整性的良好开发习惯。

为什么做开发者测试?

如何做好开发者测试?

关键在于快速有效地构建测试防护网。那么，如何快速有效地构建测试防护网呢?

什么是好的测试防护网

测试框架与原则

- FIRST原则：掌握测试用例编写的基本原则，确保测试用例的独立性、可重复性和及时性
- 正交分解法：设计高效测试用例的方法，覆盖更多的测试场景
- 测试命名规则：理解并应用“测试即文档”的概念，使测试用例名称清晰明了，便于理解和维护

测试防护网：FIRST原则

Fast（快速）

测试的运行速度将直接决定自动化的反馈速度，速度越快，越能提早发现问题。

反例：一个测试用例需要跑10分钟。

测试防护网：FIRST原则

Isolated（独立）

每一个测试用例需要保证完全独立运行，互不依赖，这样才能保证测试运行不被干扰，从而能够进行并行运行甚至分布式运行，使得测试运行更加Fast。

- 可独立运行任何一个测试用例
- 可任何顺序运行测试

反例：一个测试用例调用了另外一个测试用例

- 当测试相互依赖时，一个用例未通过会导致一连串的测试失败，使问题的诊断变得困难

测试防护网：FIRST原则

Repeatable（可重复）

每一个测试用例在运行条件不变的情况下，不论重复运行多少次，运行结果必须完全一致（幂等）。

反例：在没有任何代码变更的情况下，一个测试用例跑了999次都通过了，但第1000次出错了。

测试防护网：FIRST原则

Self-Validating（自验）

测试用例必须能够自动告知（断言）运行结果，而不依赖人工进行判断。

反例：测试只能打印运行日志，需要依赖人工判断运行是否符合预期。

测试防护网：FIRST原则

Timely（及时）

编写自动化测试是一种好的习惯，为了保证能够及时得到反馈，必须及时编写自动化测试而不拖延，一旦拖延，就很难补回来，为了能够将这件事情做到极致，那么就要做到“测试先行（Test First）”，直至“测试驱动开发（Test-Driven Development）”。

反例：现在太忙，后面找个时间集中补测试。

测试防护网：正交分解法

方向 / 指令	E	W	N	S
M	当前朝向E 执行M X+1	当前朝向W 执行M X−1	当前朝向N 执行M Y+1	当前朝向S 执行M Y−1
L	当前朝向E 执行L 朝向N	当前朝向W 执行L 朝向S	当前朝向N 执行L 朝向W	当前朝向S 执行L 朝向E
R	当前朝向E 执行R 朝向S	当前朝向W 执行R 朝向N	当前朝向N 执行R 朝向E	当前朝向s 执行R 朝向W

测试防护网：测试命名原则

- 清晰：清晰，不牺牲可读性和表达力
- 一致：遵循一致的命名规范，例如使用驼峰或下划线，使用动词或名词等
- 描述：能够清晰地描述测试的目的和内容，例如使用Expected-Behavior-Condition格式或使用Given-When-Then格式等
- 区分：能够区分不同的测试场景和结果，例如使用不同的前缀或后缀，或者使用不同的参数等

测试用例命名示例

- Executor组件提供初始化接口，负责将车初始化在指定位置，如果未接收到指令，则返回初始化指定的位置

  should_return_init_pose_when_without_command

- 如果Excutor未被初始化位置，则接口默认返回(0,0,N)

should_return_default_pose_when_without_init_and_command

测试用例命名示例

移动指令

- should_return_x_plus_1_given_command_is_M_and_facing_is_E
- should_return_x_minus_1_given_command_is_M_and_facing_is_W
- should_return_y_plus_1_given_command_is_M_and_facing_is_N
- should_return_y_minus_1_given_command_is_M_and_facing_is_S

左转指令

- should_return_facing_N_given_command_is_L_and_facing_is_E
- should_return_facing_W_given_command_is_L_and_facing_is_N
- should_return_facing_S_given_command_is_L_and_facing_is_W
- should_return_facing_E_given_command_is_L_and_facing_is_S

右转指令

- should_return_facing_S_given_command_is_R_and_facing_is_E
- should_return_facing_W_given_command_is_R_and_facing_is_S
- should_return_facing_N_given_command_is_R_and_facing_is_W
- should_return_facing_E_given_command_is_R_and_facing_is_N

测试用例的命名应反映其业务场景，并在出错时能够迅速定位到问题代码：

- **业务场景明确**：测试用例名称应清晰描述其覆盖的业务场景，在名称中包含业务场景关键词，便于理解测试目的
- **清晰明了**：名称应清晰描述测试内容
- **一致性**：遵循统一的命名风格和格式
- **可读性**：名称应易于理解，便于团队成员快速识别
- **易于错误定位**：确保名称中包含足够信息，以便在测试失败时快速定位问题

本节小结

好的测试防护网：

- **FIRST原则：**测试防护网应当易于更新和维护
- **正交分解法：**确保测试覆盖尽可能多的代码路径，以最大限度地发现潜在问题
- **命名原则：**代码和测试用例应当清晰易懂，便于团队成员理解和使用

目录

思考

通过合理应用C++特性，定义了简洁、清晰的接口，并学习了开发者测试的原则、规范和实践，设计了测试防护网。这些措施能帮助我们在开发中高效、高质量地实现需求代码。

然而，在实际开发过程中，我们可能会遇到以下问题：

- **代码回滚**：如何在出现代码问题后迅速恢复到之前的稳定状态?
- **团队高效协作**：如何实现团队成员之间的高效协作?

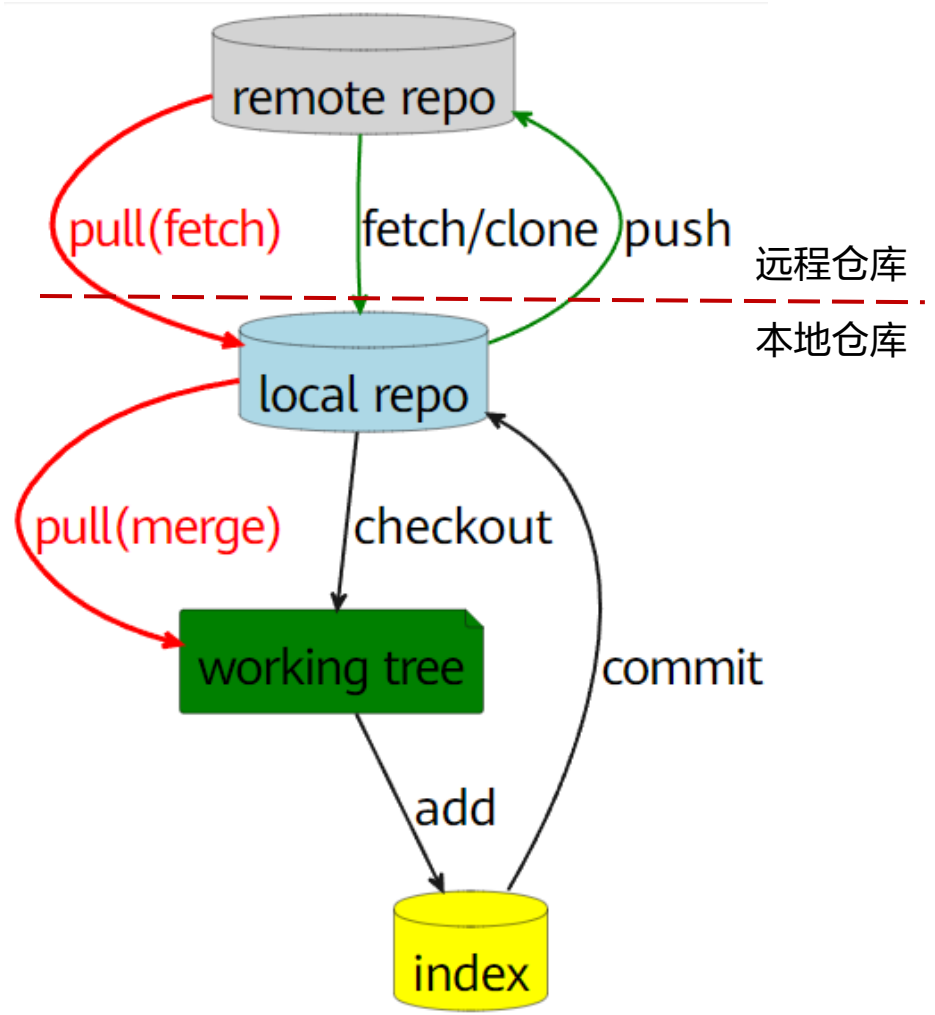
在企业软件开发中，Git能够显著提升开发效率，解决上述问题：

- 快速恢复代码
  - •使用Git的版本控制功能，可以轻松回滚到之前的稳定版本，快速恢复代码
  - •例如，使用git checkout命令可以切换到指定的提交记录，恢复代码状态
- 高效协作
  - •Git支持多人协作开发，通过分支和合并功能，团队成员可以并行工作，避免代码冲突。
  - •使用git pull和git push命令同步团队成员的代码变更，确保所有人都在最新的代码基础上工作

代码管理

git命令执行后，文件在不同的区域(remote repo、local repo、working tree、index)中变化：

命令	作用	remote repo	local repo	working tree	index
pull	从远程仓库获取最新版本并合并到本地分支		更新并合并	更新并合并	更新并合并
fetch	从远程仓库获取最新版本，但不合并到本地分支		更新		
clone	克隆远程仓库到本地		新建并初始化	新建并初始化	新建并初始化
push	将本地分支的提交推送到远程仓库	更新			
checkout	切换分支或恢复工作区文件			更新	
add	将工作区的修改添加到暂存区				更新
commit	将暂存区的修改提交到本地仓库		更新		清空



git常用命令

新建仓库

- git init初始化仓库，可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的

代码提交

- git add . 提交修改过或新增的文件到暂存区
- git add <file> 提交指定的修改过或新增的文件到暂存区
- git commit -m "commit info" 将暂存区的文件提交到本地仓库中

创建和合并分支

- git checkout -b dev创建一个新的分支：dev，并且会切换到dev分支
- git branch dev，新建分支是新建指针,指向当前commit
- git checkout dev切换到dev分支

查询配置信息

- git config --list 列出当前配置

常用查看指令

- git status 查看仓库当前的状态

bug分支

git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

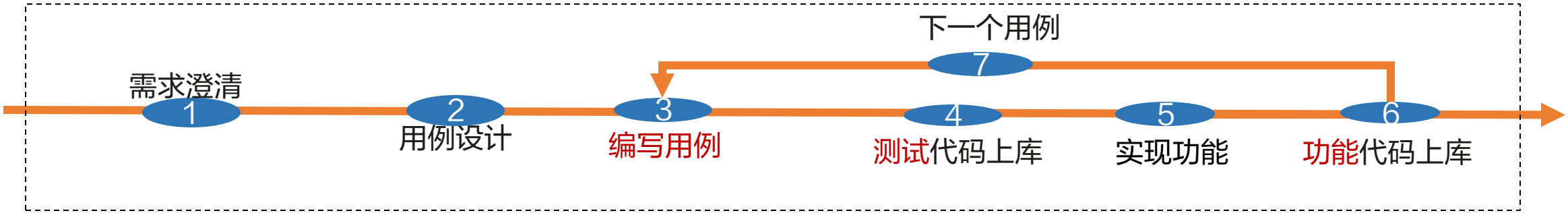
- git stash保存工作现场
- git stash list 查看工作现场
- git stash apply恢复工作现场，但是恢复后，stash内容并不删除，有多个工作现场时可以git stash apply stash@{0}恢复特定的现场
- git stash drop删除stash的内容
- git stash pop恢复的同时也把stas内容删除

本节小结

- 版本控制：
  - Git可以跟踪每一次代码变更，方便回溯历史版本，查看代码的变更记录
  - 通过分支管理，可以同时进行多个功能的开发和测试，而不会影响主分支的稳定性
- 协同工作：
  - 支持多人协作，多个开发者可以同时对同一个项目进行修改和提交，大大提高了开发效率
  - 通过Pull Request功能，团队成员可以进行代码审查，确保代码质量

目录

功能开发主要环节



需求澄清	用例设计及编写	测试代码上库	实现功能	功能代码上库
需求澄清确保团队和客户对项目目标和期望一致，从而提高效率和质量，减少风险	首先用例设计（正交分解法+最小等价类）， 然后编写测试用例，只关注需求，测试程序的输入输出，不关注中间过程。这一步确保了开发过程中的每个功能都能满足需求，并且减少了后期发现问题的风险	编写测试用例，编译通过后，代码入库，确保测试用例的独立性	用最简单的方式满足当前需求，不考虑其他需求	实现功能，编译运行通过后，代码入库，确保功能代码和测试代码的修改是分开的

请参见实验一学生引导StepbyStep.pptx

本节小结

- **测试失败的意义：**确认我们对代码行为的假设是否正确
- **小步迭代虽然看似缓慢，但有助于：**
  - **降低风险**：每次变更都较小，容易定位问题
  - **提高代码质量**：每次提交都经过充分测试，确保稳定性
  - **养成良好习惯**：在实训阶段，小步迭代有助于培养良好的开发习惯
- **代码命名的重要性**
  - **功能代码**：清晰的命名有助于理解代码逻辑和功能
  - **测试代码**：明确的测试命名可以帮助快速定位和理解测试目的
  - **git提交注释**：详细的提交注释可以记录每次变更的业务背景和目的，方便日后追溯

目录

本章总结

通过本课程的学习，您已经掌握了高质量编程的基础知识，特别是在以下几个方面：

- **开发者自测试：**您学会了如何确保代码的正确性和需求的完整性，掌握了测试的基本理论和方法，并实践了有效的开发者测试技术
- **命名：**您了解并实践了提高代码可读性的命名原则、规范
- **代码版本管理：**学习使用git进行代码版本管理的规范和实践

这些技能和知识将为您在未来的企业软件开发中打下坚实的基础，帮助您成为一名更加专业和高效的软件开发者。

感谢您参与本课程，期待您在未来的软件开发工作中取得更大的成就！

作业

完成课程实训需求1支持基本控制指令所有功能：

- **测试先行**：优先编写测试代码，确保每一步都有测试覆盖
- **小步提交**：每次提交代码时，确保变更小且独立，便于回溯和审查
- **清晰的提交注释**：每次提交时，使用明确且无歧义的注释，准确描述本次提交的业务逻辑和变更内容
- **一周内完成**：确保在一周内完成所有作业并提交

学习推荐

在线参考资料网站，涵盖了C++基本概念到高级特性、标准库函数、类和模板等各个方面：

- 搜索功能：有强大的搜索功能，可以快速找到需要的函数、类或概念
- 示例代码：可以帮助理解如何使用C++特性
- 网址：https://en.cppreference.com/w/

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and organization for a fully connected, intelligent world.

HUAWEI