中南大学
CENTRAL SOUTH UNIVERSITY

软件工程

Software Engineering

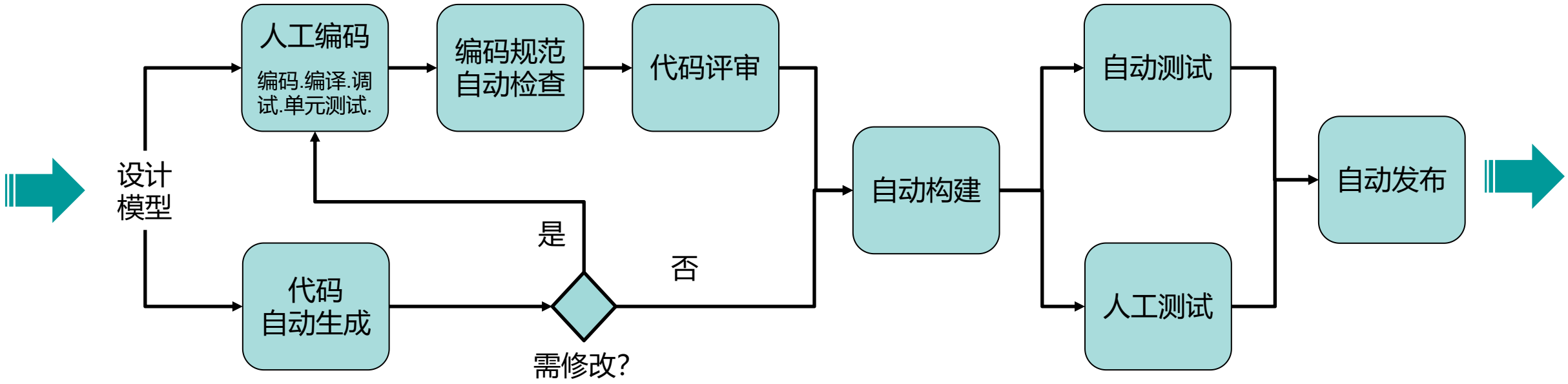
龙 军

jlong@csu.edu.cn 18673197878

计算机学院 | 大数据研究院

软件设计后...

高质量和高效能的编码！



- 通过持续集成工具进行工具链集成

从 宏观上 看百度程序员的工作

大 纲

第八章 编码

01-编码

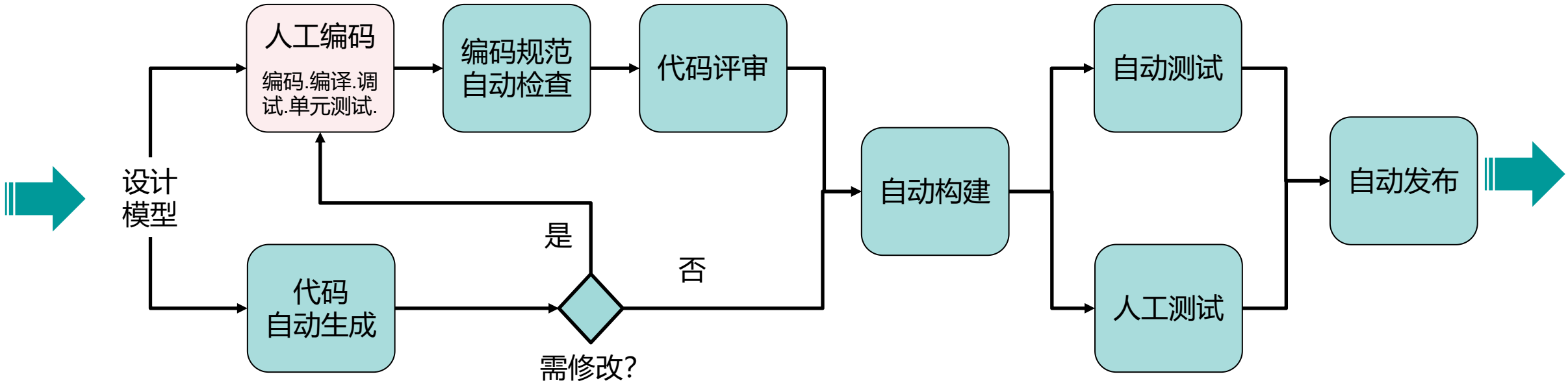
02-代码静态检查和评审

03-代码自动生成

04-软件持续集成

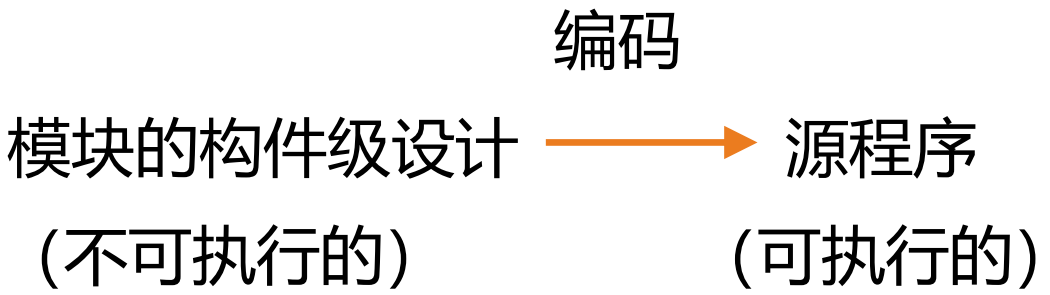
@第7章.教材

编码



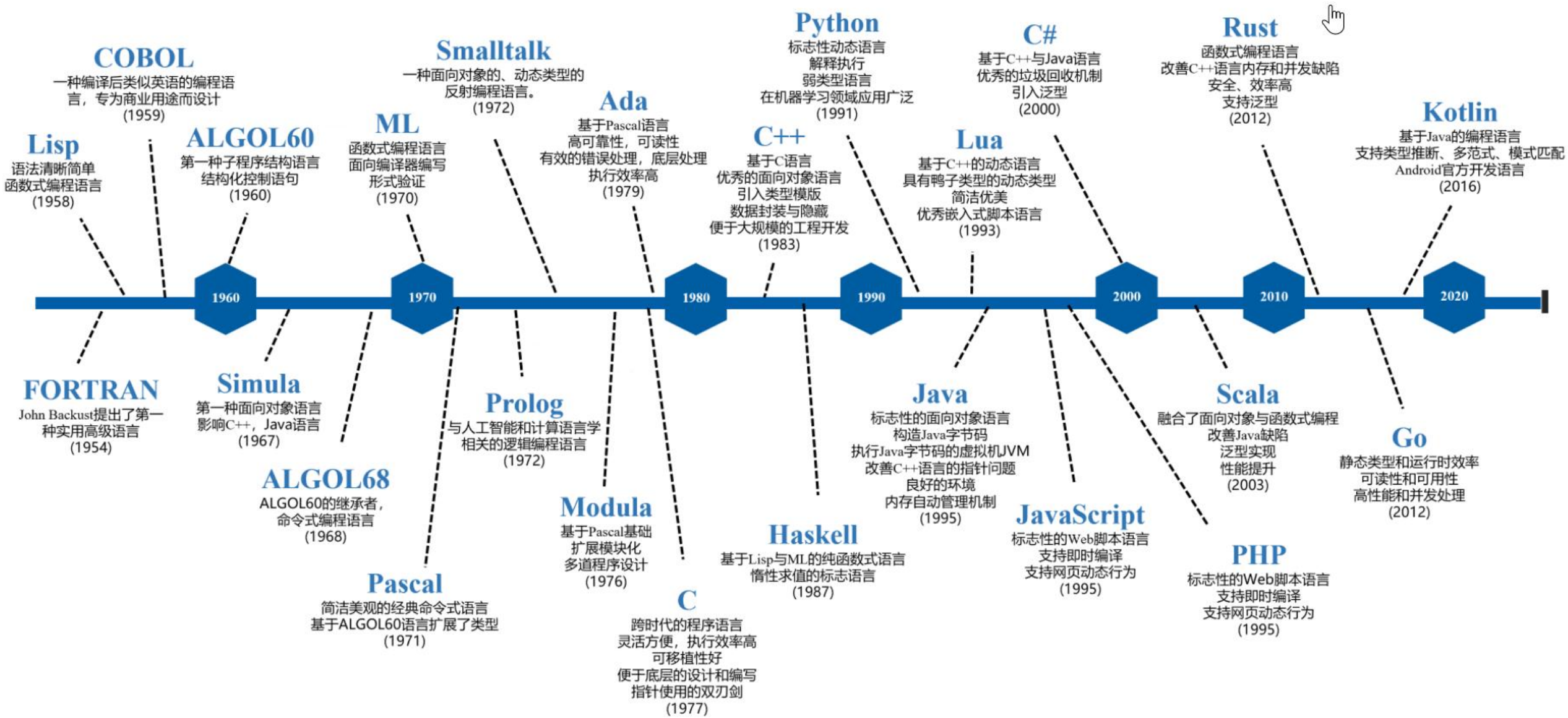
- 通过持续集成工具进行工具链集成

编码的目的和质量要求

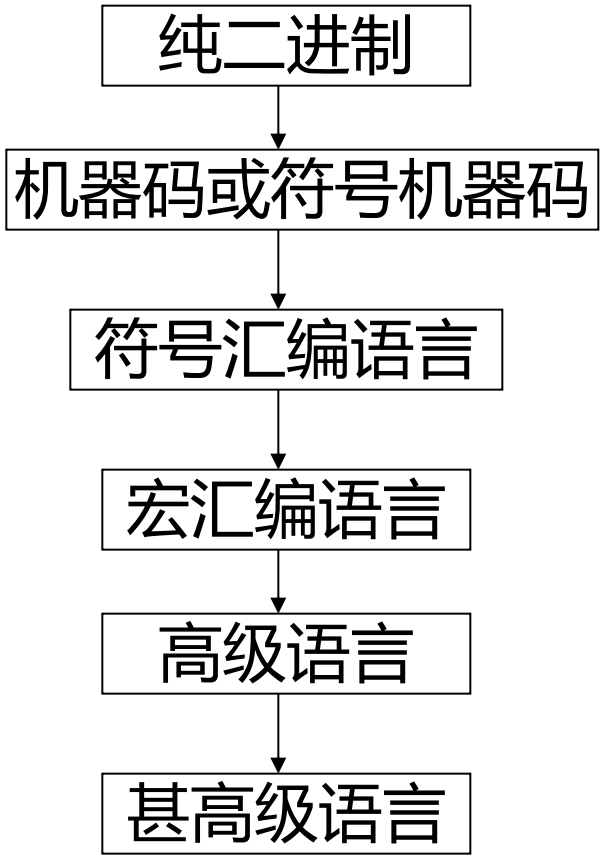


- 编程语言的特性和编程风格会深刻地影响软件的质量和可维护性。
- 为了保证程序编码的质量，程序员必须深刻理解、熟练掌握并正确地运用编程语言的特性。此外，还要求源程序具有良好的结构性和良好的编程风格。

程序设计语言的发展


COBOL
一种编译后类似英语的编程语言，专为商业用途而设计
(1959)
Lisp
语法清晰简单
函数式编程语言
(1958)
ALGOL60
第一种子程序结构语言
结构化控制语句
(1960)
ML
函数式编程语言
面向编译器编写
形式验证
(1970)
Smalltalk
一种面向对象的、动态类型的
反射编程语言。
(1972)
Ada
基于Pascal语言
高可靠性，可读性
有效的错误处理，底层处理
执行效率高
(1979)
Python
标志性动态语言
解释执行
弱类型语言
在机器学习领域应用广泛
(1991)
C++
基于C语言
优秀的面向对象语言
引入类型模版
数据封装与隐藏
便于大规模的工程开发
(1983)
C#
基于C++与Java语言
优秀的垃圾回收机制
引入泛型
(2000)
Lua
基于C++的动态语言
具有鸭子类型的动态类型
简洁优美
优秀嵌入式脚本语言
(1993)
Rust
函数式编程语言
改善C++语言内存和并发缺陷
安全、效率高
支持泛型
(2012)
Kotlin
基于Java的编程语言
支持类型推断、多范式、模式匹配
Android官方开发语言
(2016)
1960
1970
1980
1990
2000
2010
2020
FORTRAN
John Backust提出了第一种实用高级语言
(1954)
Simula
第一种面向对象语言
影响C++，Java语言
(1967)
ALGOL68
ALGOL60的继承者，
命令式编程语言
(1968)
Pascal
简洁美观的经典命令式语言
基于ALGOL60语言扩展了类型
(1971)
Prolog
与人工智能和计算语言学
相关的逻辑编程语言
(1972)
Modula
基于Pascal基础
扩展模块化
多道程序设计
(1976)
C
跨时代的程序语言
灵活方便，执行效率高
可移植性好
便于底层的设计和编写
指针使用的双刃剑
(1977)
Haskell
基于Lisp与ML的纯函数式语言
惰性求值的标志语言
(1987)
Java
标志性的面向对象语言
构造Java字节码
执行Java字节码的虚拟机JVM
改善C++语言的指针问题
良好的环境
内存自动管理机制
(1995)
JavaScript
标志性的Web脚本语言
支持即时编译
支持网页动态行为
(1995)
PHP
标志性的Web脚本语言
支持即时编译
支持网页动态行为
(1995)
Scala
融合了面向对象与函数式编程
改善Java缺陷
泛型实现
性能提升
(2003)
Go
静态类型和运行时效率
可读性和可用性
高性能和并发处理
(2012)

计算机上语言的层次



Byte或word，指令、数据不分

用一些符号来代表指令，如sub代表减，Add代表加等，
机器地址用十进制。有时汇编语言等同于符号机器码。

变量名用符号，地址也可用符号而非数字。
编制的程序称为汇编语言程序。

用户可以定义新指令及子程序

源程序编译为目标程序，或解释执行

高级程序语言的高层规约语言，提供比高级
程序语言更高级的语言设施。又称为“可执行的规约语言”。有时也不区别于高级语言。

高级程序设计语言的分类

- 命令式（imperative） 语言
  - 结构化语言 C, Ada, Fortran …
  - 面向对象语言 Smalltalk, Eiffel, C++, Java …
  - 脚本语言 Perl, Python, PHP…
- 说明式（declarative）语言
  - 函数式 Lisp/Scheme, ML, Haskell, Clean, Erlang, Miranda…
  - 数据流 Id, Val …
  - 逻辑式 或基于约束的 Prolog, spreedsheets …
  - 基于模板的 XSLT …
- 量子（quantum）编程语言
  - Q#, Quipper, Sliq

语言的选择

- 选择编码语言的标准
  - 应用领域
  - 算法、计算、数据结构的复杂性
  - 运行效率、开发效率、可移植性、安全性等考虑
  - 库
  - 生态、大厂的支持

TIOBE Programming Community Index

（https://www.tiobe.com/tiobe-index/ ）

语言选择举例

- 如果编写对性能要求苛刻，或和操作系统结合紧密的程序，必然选择c。
- 如果需要跨平台，又要广泛的支持的话，选择java。
- 编写服务器端程序，用Java 、JavaScript 、php、perl、python、asp。
- 编写客户端程序，最常用的是JavaScript。
- 编写机器学习软件采用python、R、Java
- 编写数据库程序最简单的语言是vb或delphi。
- 编写知识的处理程序用prolog。
- 编写最灵活，最模糊的程序用lisp。
- 编写office程序用vba。
- 如果只作为简单应用的工具语言，python和ruby是更好的选择，他们的跨平台移植性好，应用也比较广泛，语言简单，生产率高。

编码准则

- 降低复杂性
  - 好的代码应是简单、整洁（clean）的代码，容易理解、测试和维护
- 预测并拥抱变化
  - 预测变化，将灵活性和适应性构建到软件中，实现可维护的、可扩展的软件，从而能在不破坏架构的情况下修改软件
- 为验证而编码
  - 更容易地发现软件中的错误。
- 代码复用
  - 框架、库、模块、构件、源代码和商业现货（COTS）资产的复用
- 采用开发标准
  - 遵循文档模板、编码规范、接口标准、模型标准等

Clean Code

- 追求“聪明”和“技巧” → 提倡“简明”和“直接”
- 清晰的前提下求取效率
  - Make it right before you make it faster.
  - Make it clear before you make it faster.
  - Keep it right when you make it faster.

    (求快不忘保持程序正确)
  - Keep it simple to make it faster.

    (保持程序简单以求快)
  - Don’t sacrifice clarity for “efficiency”.

    (书写清楚，不要为“效率”牺牲清楚)

符合编码规范

- 目的：编写出简洁、可维护、可靠、可测试、高效、可移植的代码
- Google编码规范

  英文：https://github.com/google/styleguide

  中文：https://github.com/zh-google-styleguide/zh-google-styleguide

示例：Google C++ 编程规范

- 编程规范的大纲

1. 头文件
2. 作用域
3. 类
4. 函数
5. 来自 Google 的奇技
6. 其他 C++ 特性
7. 命名约定
8. 注释
9. 格式

- 编程规范规则的例子

- 函数体尽量短小, 紧凑, 功能单一.
- 所有按引用传递的参数必须加上 const.
- 函数命名, 变量命名, 文件命名要有描述性; 少用缩写.
- 每个类的定义都要附带一份注释, 描述类的功能和用法.

源程序的文档化 （code documentation）

- 有意义的变量名称和规范的命名形式

命名形式	形式特点	例子	常用情况
大驼峰	大写字母开头；多个单词时，每个单词首字母大写，其他字母小写	CourseOffered	类名、接口名、注解、枚举类型
小驼峰	小写字母开头；多个单词时，除了第一个单词全小写，其他每个单词首字母大写、其余字母小写	selectedCourse	类的属性名、局部变量名、方法名、方法参数
全大写	所有字母大写	MANDATORY、IN_PROGRESS	枚举值，静态常量

- 适当的注释
- 标准的书写格式
  - 用分层缩进的写法显示嵌套结构的层次；
  - 在注释段与程序段、以及不同程序段之间插入空行；
  - 每行只写一条语句；
  - 书写表达式时，适当使用空格或圆括号等作隔离符.

好的源程序本身
就是详细设计文档！

案例：低质量的代码（图书借阅功能）

```
1   public void borrowBook(String id1, String id2) {
2     if (! id1.equals(""))) {
3       if (checkBooks(id1) < 5) {
4         if (!(hasOverdueBooks(id1))) {
5           if (! id2.equals(""))) {
6           if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7             updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8           throw new IllegalArgumentException("This book can't be borrowed");
9           } else
10            throw new IllegalArgumentException("Book id can't be empty");
11        }
12        else
13          throw new IllegalArgumentException("Overdue books exist");
14      } else
15        throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16    } else
17      throw new IllegalArgumentException("Student id can't be empty");
18  }
```

这段代码有哪些问题？

案例：低质量的代码（图书借阅功能）

标识符命名不规范，难以理解其含义

```
1   public void borrowBook(String id1, String id2) {
2     if (! id1.equals(""))) {
3       if (checkBooks(id1) < 5) {
4         if (!(hasOverdueBooks(id1))) {
5           if (! id2.equals(""))) {
6           if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7             updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8           throw new IllegalArgumentException("This book can't be borrowed");
9           } else
10            throw new IllegalArgumentException("Book id can't be empty");
11        }
12        else
13          throw new IllegalArgumentException("Overdue books exist");
14      } else
15        throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16    } else
17      throw new IllegalArgumentException("Student id can't be empty");
18  }
```

未考虑id1、id2为null的情况

“魔法”数字（代码中含义不明的数字或字符串）

五层条件语句嵌套，逻辑难以理解

语句缩进排版不规范，容易造成误解

if条件第二部分少了“非”操作

缺少else

异常抛出不当，需要使用自定义业务异常

案例：改进后的代码（图书借阅功能）

标识符规范命名，含义自解释

自定义业务异常类并根据具体情况建立异常类继承层次

```
1       public void borrowBook(String studentID, String bookID) throw ServiceException {
2           if (studentID == null || "".equals(studentID))
3               throw new StudentIDEmptyException();
4           if (getBorrowedBookCount(studentID) >= BOOK_BORROW_LIMIT)
6               throw new BookBorrowExeedLimitException();
7           if (hasOverdueBooks(studentID))
8               throw new OverdueBooksException();
9           if (bookID == null || "".equals(bookID))
10              throw new BookIDEmptyException();
11          if (getBook(bookID).isAllOut() || getBook(bookID).blocked())
12              throw new BookUnavailableException();
13          updateStudentStatus(studentID, bookID);
14          updateBookStatus(bookID, studentID);
15      }
```

对可能为null的参数进行判断

通过针对各种规则检查的异常抛出消除了深层嵌套的条件语句

代码结构变清晰之后各种缺陷也更容易被发现和纠正

案例：进一步改进（图书借阅功能）

通过通用方法notEmpty判断字符串是否为空（如果为null或空字符串则抛出异常）

```
1     public void borrowBook(String studentID, String bookID) throw ServiceException {
2         notEmpty(studentID, "Student ID can't be empty");
3         notEmpty(bookID, "Book ID can't be empty");
4
5         checkStudentCanBorrowBook(studentID);
6         checkBookCanBeBorrowed(bookID);
7         updateBorrowingStatus(studentID, bookID);
…         ……
```

通过专门的check方法和update方法分别执行相应的条件检查和最终的借书信息更新

整个代码逻辑非常清楚，代码本身的标识符（参数名、方法名）基本都是自解释的

大 纲

中南大學
CENTRAL SOUTH UNIVERSITY

第八章 编码

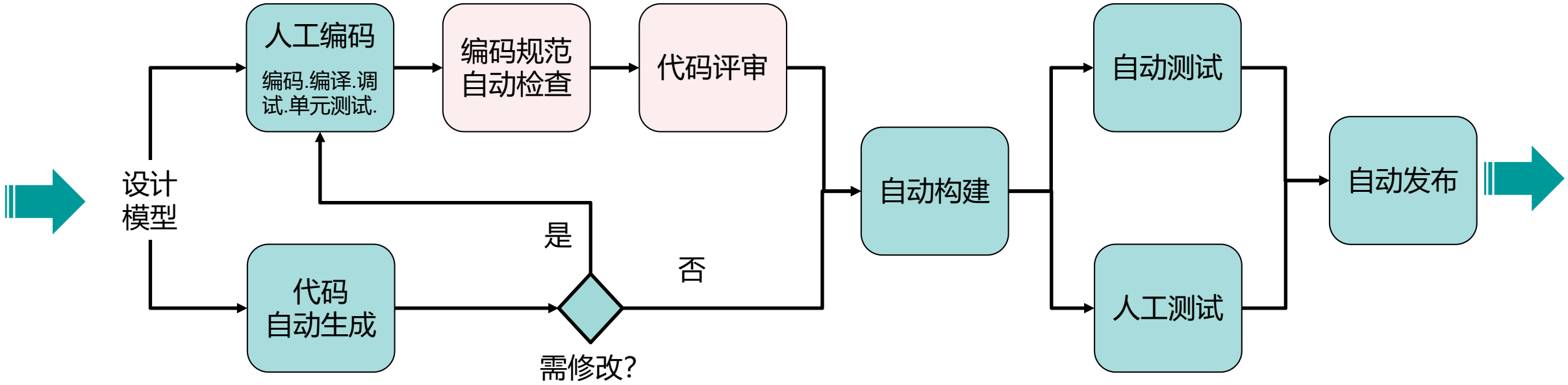
01-编码

02-代码静态检查和评审

03-代码自动生成

04-软件持续集成

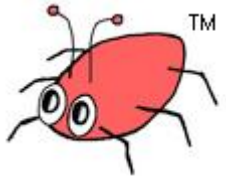
编码规范和代码评审



- 通过持续集成工具进行工具链集成

编码规范的自动检查

- 普遍都利用代码静态分析技术来发现潜在的代码质量问题
- 一般都使用基于规则的检测方法来发现各种代码质量问题
- 常用工具
  - PMD (Java)
  - flake8和pylint (Python)
  - SonarQube (多语言)
  - SpotBugs （Java, 前身是FindBugs ）
  - CheckStyle (Java)
  - .....



举例：SonarCube

问题类型：

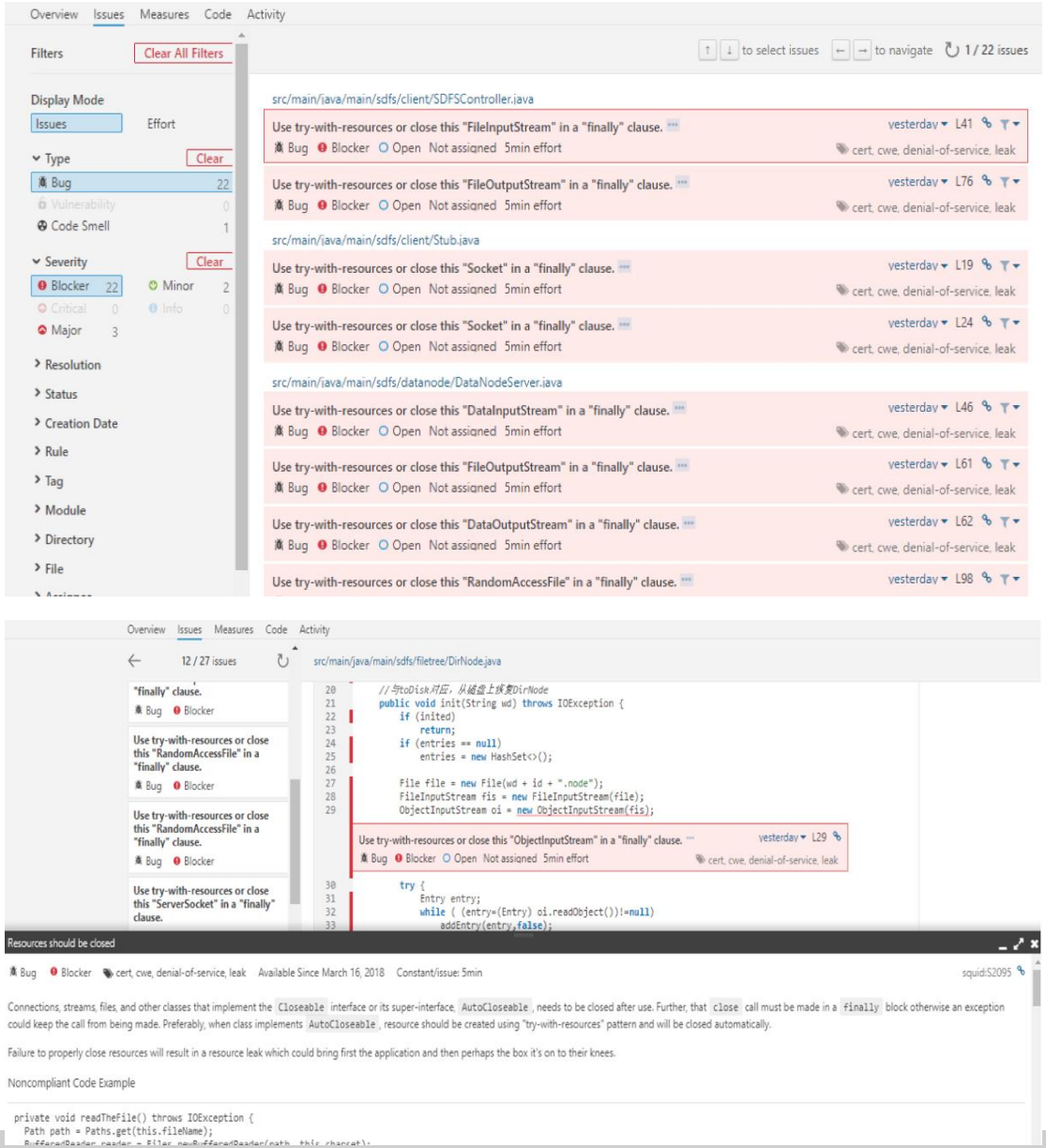
- 缺陷（Bug）
- 漏洞（Vulnerability）
- 代码坏味道（Code Smell）

严重程度（从高到低）：

- Blocker
- Critical
- Major
- Minor
- Info等

问题详细信息：

- 所在位置、问题类型、严重程度、问题解释



何时进行编码规范的自动检查?

- 个人本地检测
  - 开发人员在进行编码活动时利用IDE中的代码静态扫描插件实时扫描代码中可能存在的质量问题并及时进行改进
  - 对于提升个人编写代码的质量有重要的作用
- 作为持续集成工具链的一部分
  - 自动执行检测，作为持续集成结果的一部分进行反馈
- 作为代码提交或合并之前的质量门禁，例如
  - 要求提交或合并的代码中不允许存在严重的代码质量问题（即严重程度为Critical或更高）
  - 要求所发现的代码质量问题的数量不能增加

代码评审(code review)

- Why：有些代码质量问题很难通过测试或静态检查来发现，但有经验的开发人员很容易看出来
- 代码评审：通过代码阅读的形式来判断源代码是否符合指定的代码质量标准，同时发现潜在的代码质量问题
- 正式的代码评审：在代码提交、合并的流程中进行控制，评审不通过的代码不能进入代码库
- 评审内容：从设计、功能、复杂性、测试、实现逻辑、异常处理、编码规范等方面评价代码的质量
- 代码评审工具：Phabricator、Collaborator、Gerrit等
  - 目的是集成配置管理和版本管理工具、便于阅读和理解代码修改（例如高亮差异部分）及添加评审反馈（打分、备注等）

大 纲

第八章 编码

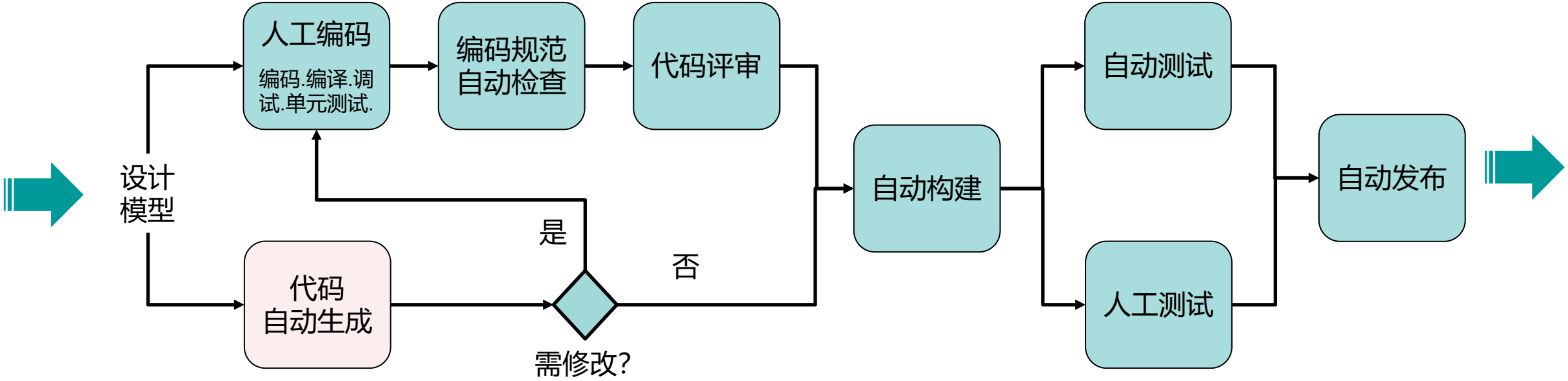
01-编码

02-代码静态检查和评审


03-代码自动生成

04-软件持续集成

代码自动生成

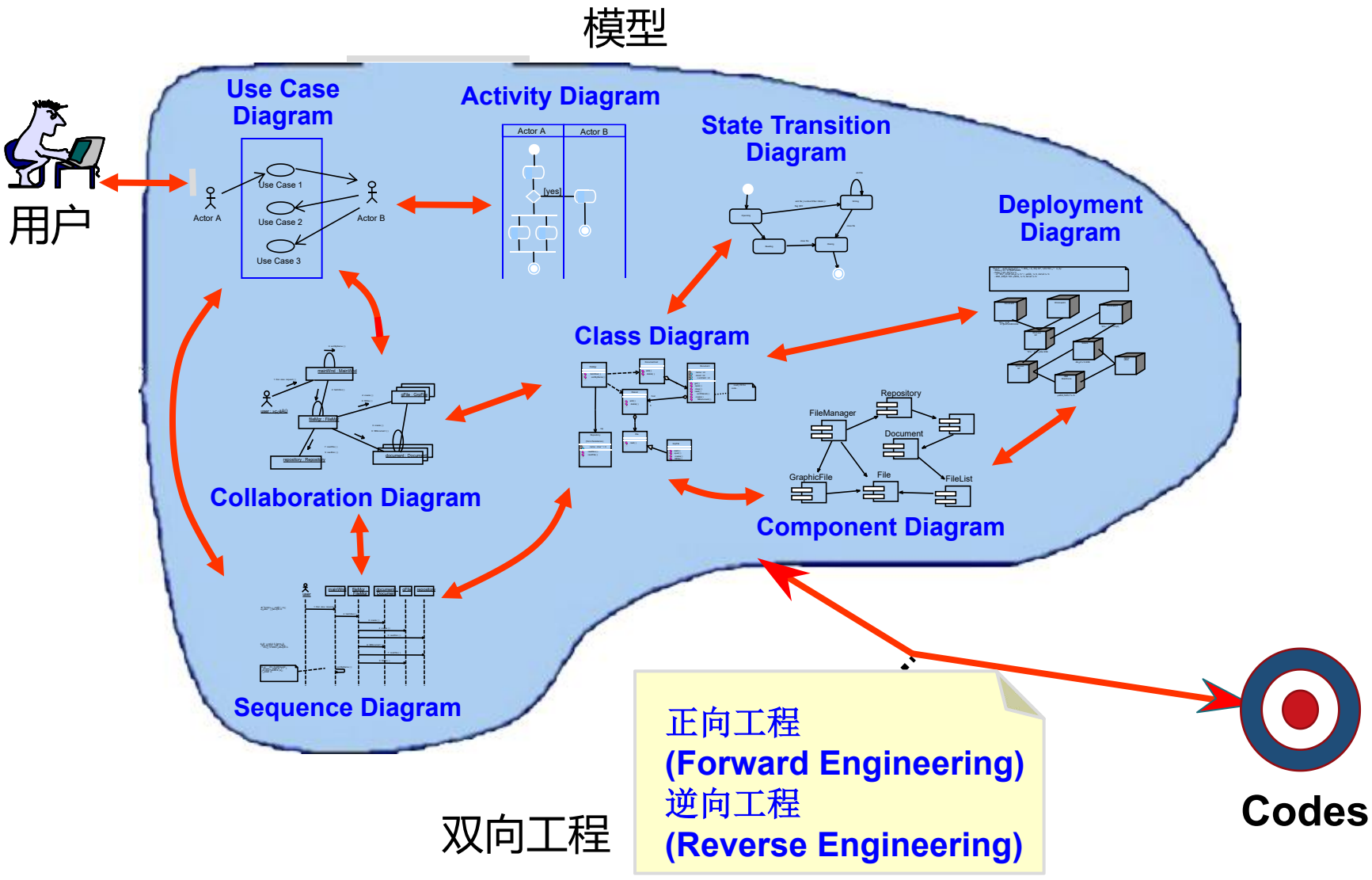


- 通过持续集成工具进行工具链集成

代码自动生成的技术

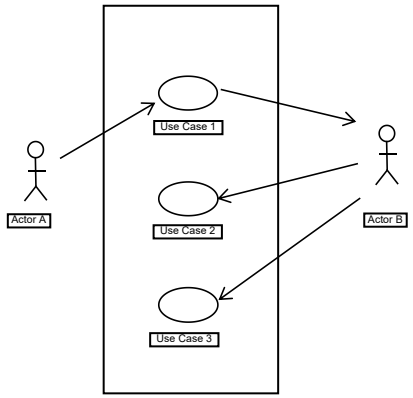
- Model2Code （从模型生成代码）
  - 根据软件分析模型或设计模型生成代码
- NL2Code （从需求生成代码）
  - 根据自然语言描述生成代码
- Code2Code （代码补全）
  - 从前续代码生成后续代码
- Sample2Code (示例编程)
  - 从例子生成代码
- UI2Code （从界面图像生成代码）
  - 基于CNN和RNN的神经网络机器翻译

从模型中生成代码


模型
用户
Use Case Diagram
Activity Diagram
State Transition Diagram
Deployment Diagram
Class Diagram
Collaboration Diagram
Component Diagram
Sequence Diagram
Use Case 1
Use Case 2
Use Case 3
Actor A
Actor B
[yes]
Repository
FileManager
Document
GraphicFile
File
FileList
正向工程
(Forward Engineering)
逆向工程
(Reverse Engineering)
双向工程
Codes

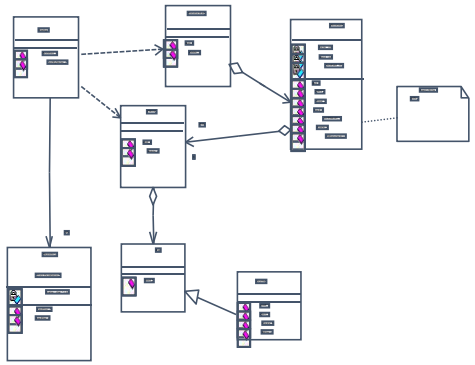
确保模型和代码的一致

需求



一致

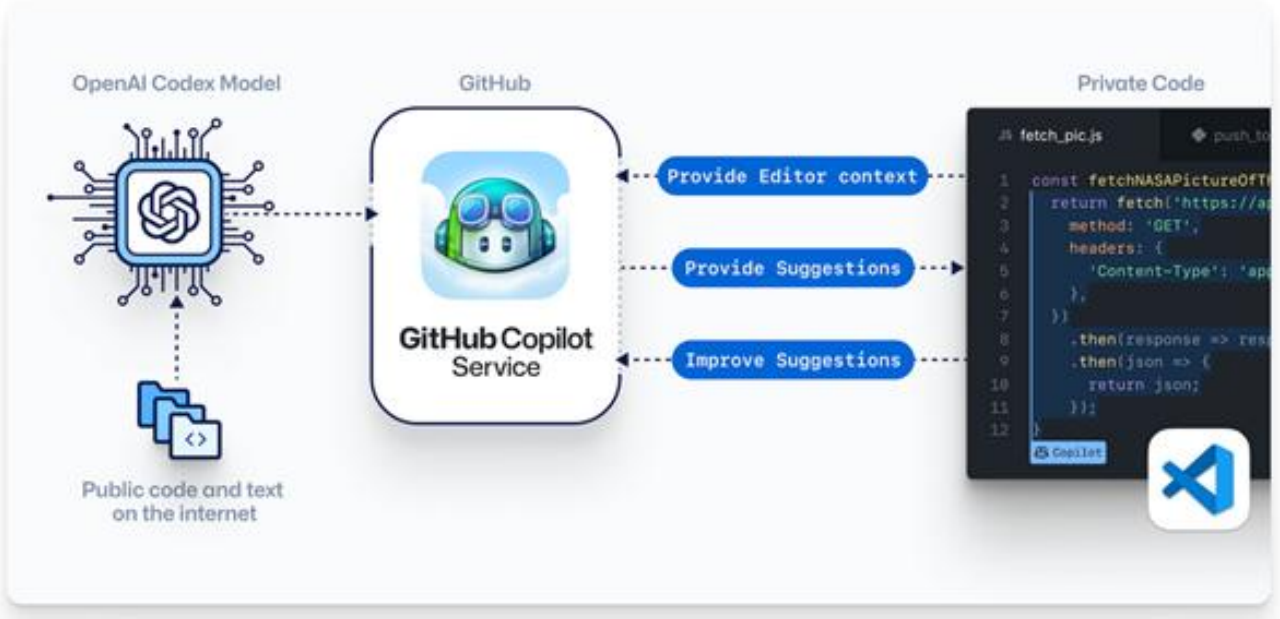
设计



一致

代码

微软的AI结对编程助手GitHub Copilot



- 工具： https://github.com/features/copilot/
- 特性：根据上下文自动生成代码，包括文档字符串、注释、函数名称、代码。
- 模型：OpenAI 的Codex，GPT-3的120亿个参数的一个版本，在GitHub的159GB代码样本上进行了微调。

大 纲

第八章 编码

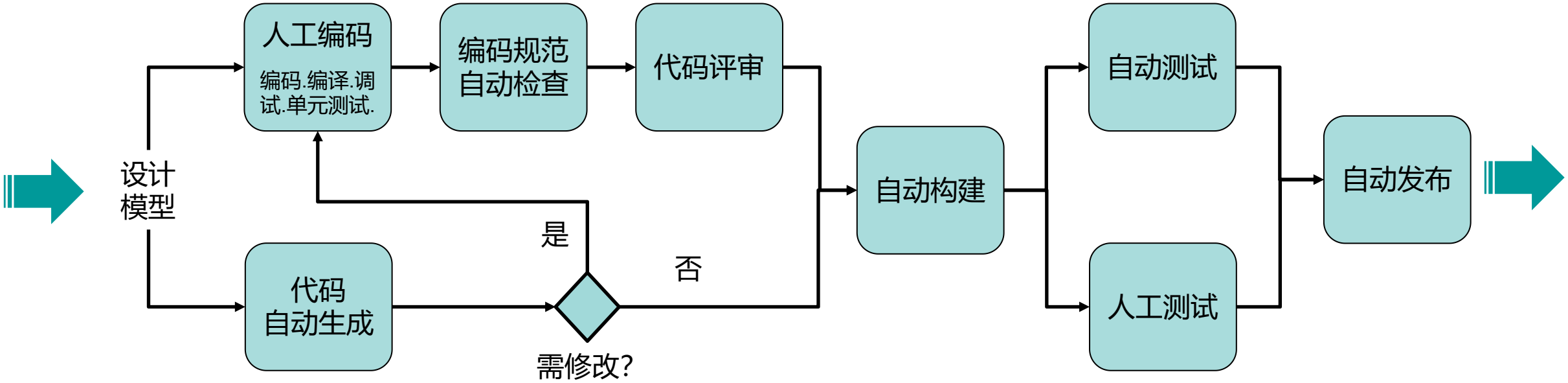
01-编码

02-代码静态检查和评审

03-代码自动生成

04-软件持续集成

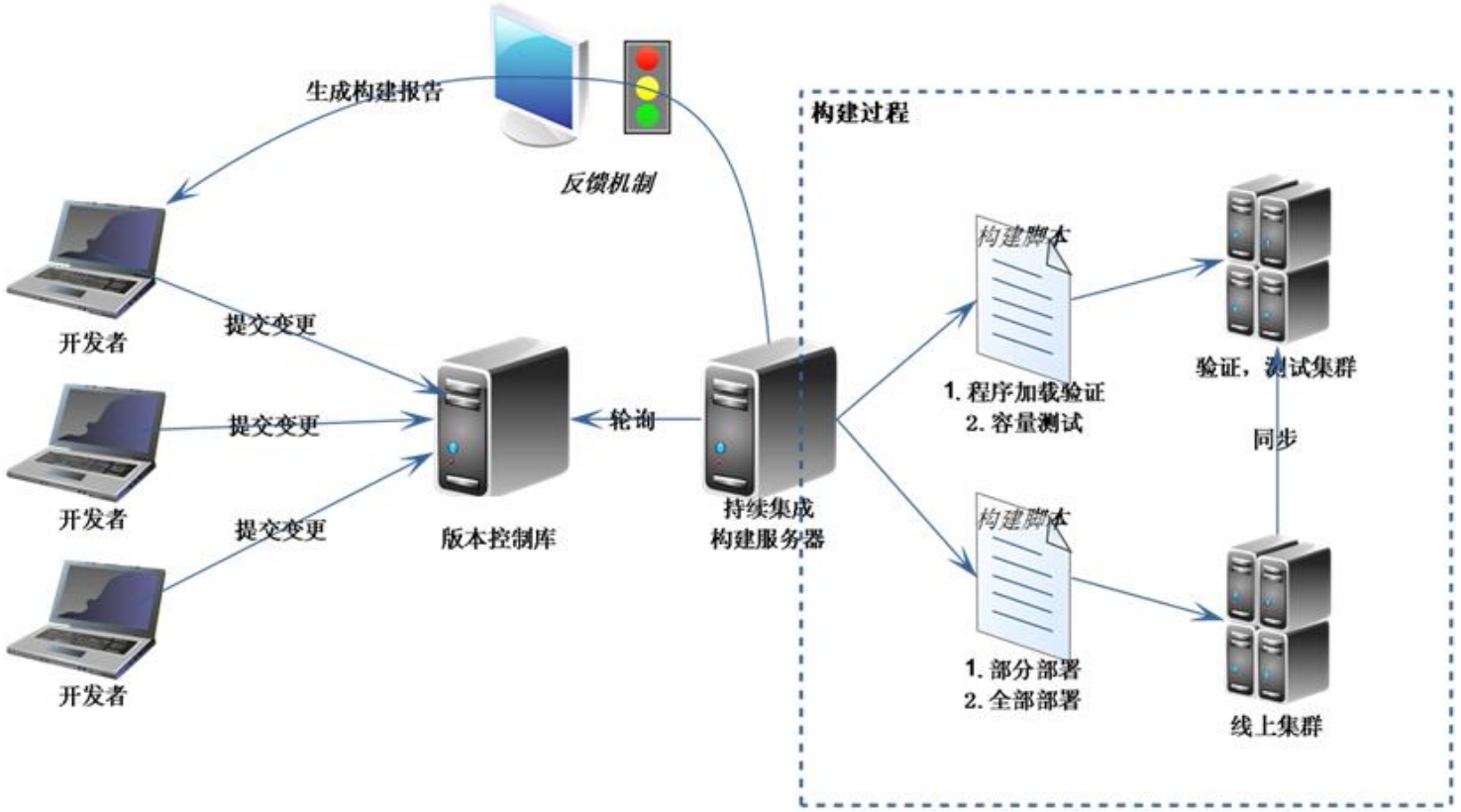
软件持续集成



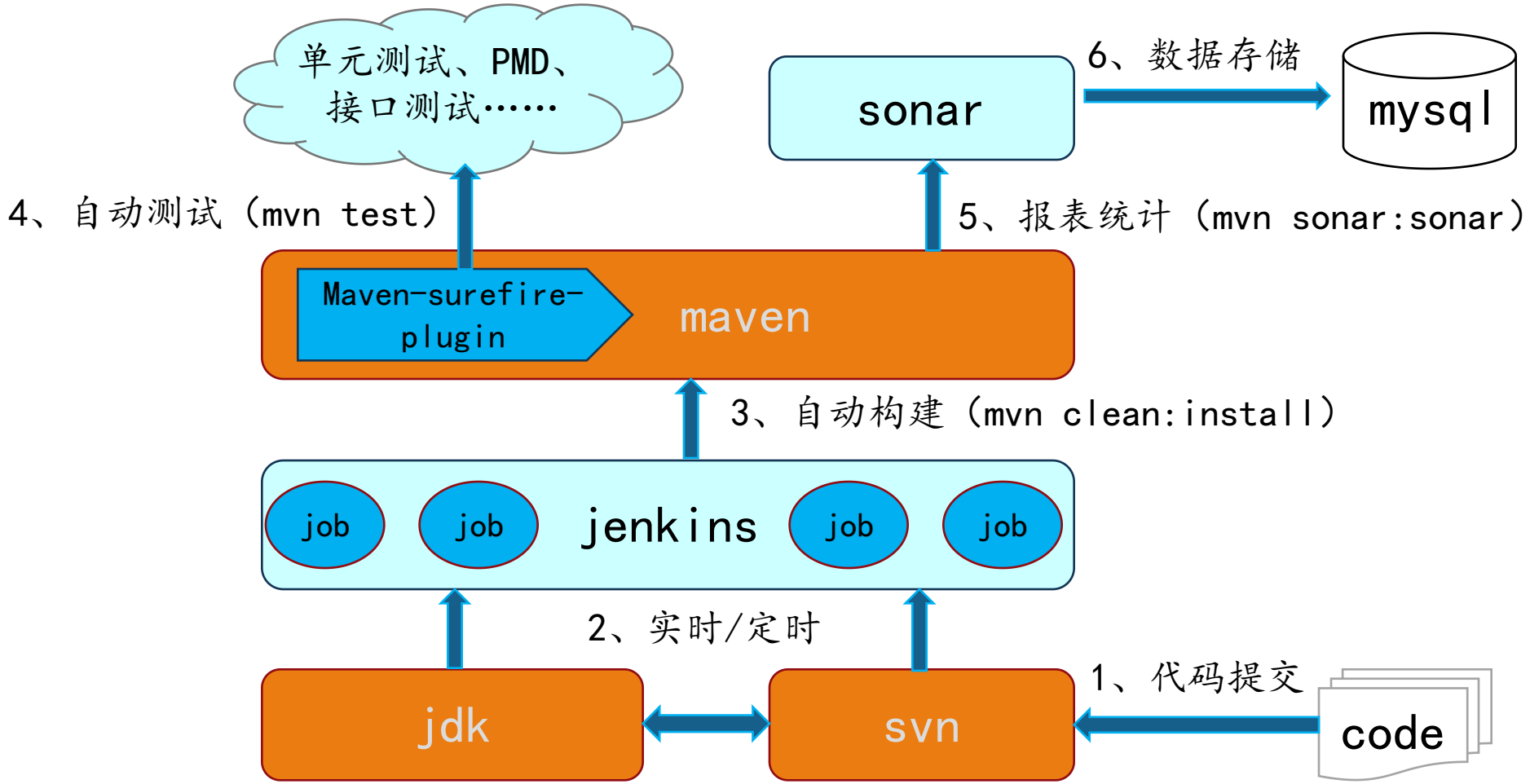
通过持续集成工具进行工具链集成

软件持续集成

- 持续集成是一种软件开发实践，即团队开发成员经常集成它们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。
- 每次集成都通过自动化的构建（包括编译，发布，自动化测试)来验证，从而尽快地发现集成错误。



持续集成示例


单元测试、PMD、
接口测试……
sonar
6、数据存储
mysql
4、自动测试（mvn test）
5、报表统计（mvn sonar:sonar）
Maven-surefire-plugin
maven
3、自动构建（mvn clean:install）
job
job
jenkins
job
job
2、实时/定时
1、代码提交
jdk
svn
code

软件编码的技术问答社区

- 常见社区
  - Stack Overflow 、CSDN
- 交流形式
  - 提出问题
  - 回答问题
  - 参加评论