SOFTWARE TESTING

Chapter 2

Basic Concepts of Software Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

软件测试的对象

- 需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都是软件测试的对象。
  - 软件测试是用人工或自动的方法执行被测软件并把观察到的行为特性与所期望的行为特性进行比较的过程。
  - 随着对软件测试方法、测试工具和测试技术的研究，测试的概念已经从编程后的评估过程 (程序测试) 发展成软件开发生命周期中每个阶段的一个必须的活动。
  - 软件测试工作涉及多方面的内容和因素，但最为重要的是三项
    - 测试工作的流程
    - 测试工具的支持
    - 测试人员的素质

软件测试的过程和方法

- Software Verification & Validation (Software V&V)
  - According to the **lEEE Std 610.12-1990**
    - Software Verification: (1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness.
    - Software Validation: The process of evaluating software *during or at the end* of the development process to determine whether it satisfies specified requirements.
  - **VERIFICATION** and **VALIDATION** are hugely confused and debated terms in the software testing world. *B. Boehm* succinctly expressed the difference as
    - Verification: Are we building the product right? (过程正确性)
    - Validation: Are we building the right product? (结果正确性)

lEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology (Revision and Redesignation of IEEE Std 792-1983)
Also ref. to the Capability Maturity Model (CMMI-SW v1.1)

- Software Verification & Validation
  - Software Verification
    - Software verification is the process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase.
    - Software verification ensures that the output of each phase of the software development process effectively carry out what its corresponding input artifact specifies (requirement -> design -> software product).
    - Software verification ensures that "you built the thing right" and confirms that the product, as provided, fulfills the plans of the developers.
    - 软件验证试图证明软件开发周期的各个阶段，以及阶段间的逻辑协调性、完备性和正确性。

- Software Verification & Validation
  - Software Verification
    - Artifact verification
      - The output of each software development process stage can be subject to verification when checked against its input specification.
      - Examples of artifact verification:
        - Of the design specification against the requirement specification: Do the architectural design, detailed design and database logical model specifications correctly implement the functional and non-functional requirement specifications?
        - Of the construction artifacts against the design specification: Do the source code, user interfaces and database physical model correctly implement the design specification?

- Software Verification & Validation
  - Software Validation
    - Software validation is the process of evaluating software at the end of software development to ensure compliance with intended usage.
    - Software validation ensures that the software product meets the needs of all the stakeholders (such as users, operators, administrators, managers, investors, etc.).
      - Therefore, the requirement specification was correctly and accurately expressed in the first place.
    - Software validation ensures that "you built the right thing" and confirms that the product, as provided, fulfills the intended use and goals of the stakeholders.
    - Software validation during the software development process can be seen as a form of User Requirements Specification validation; and, that at the end of the development process is equivalent to Internal and/or External Software validation.

- Software Verification & Validation
  - Software Validation
    - 软件确认是一系列的活动和过程，目的是想证实在一个给定的外部环境中被测软件的逻辑正确性，包括需求规格说明确认和程序确认 (静态确认、动态确认)。
    - Internal validation and external validation
      - Internal validation assumed that the goals of the stakeholders were correctly understood and expressed in the requirement artifacts precise and comprehensively. If the software meets the requirement specification, it has been internally validated.
      - A final external validation confirms that all the stakeholders accept the software product and express that it satisfies their needs. Such final external validation requires the use of an acceptance test which is a dynamic test.

- Software Verification & Validation
  - Software Validation
    - Artifact Validation
      - Requirements should be validated before the software product as a whole is ready.
        - The waterfall development process requires them to be perfectly defined before design starts; but, iterative development processes do not require this to be so and allow their continual improvement.

- Software Verification & Validation
  - Software Validation
    - Examples of artifact validation:
      - User Requirements Specification validation: User requirements as stated in a document called User Requirements Specification are validated by checking if they indeed represent the will and goals of the stakeholders. This can be done by interviewing them and asking them directly (static testing) or even by releasing prototypes and having the users and stakeholders to assess them (dynamic testing).
      - User input validation: User input (gathered by any peripheral such as keyboard, bio-metric sensor, etc.) is validated by checking if the input provided by the software operators or users meet the domain rules and constraints (such as data type, range, and format).

- Software Verification & Validation
  - WiKi
    - Building the product right implies the use of the Requirements Specification as input for the next phase of the development process, the design process, the output of which is the Design Specification. Then, it also implies the use of the Design Specification to feed the construction process. Every time the output of a process correctly implements its input specification, the software product is one step closer to final verification. If the output of a process is incorrect, the developers are not building the product the stakeholders want correctly. This kind of verification is called "artifact or specification verification".
    - Building the right product implies creating a Requirements Specification that contains the needs and goals of the stakeholders of the software product. If such artifact is incomplete or wrong, the developers will not be able to build the product the stakeholders want. This is a form of "artifact or specification validation".

Software Verification & Validation

VERIFICATION vs. VALIDATION

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.	To ensure that the product actually meets the user's needs and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.

Software Verification & Validation

VERIFICATION vs. VALIDATION

Criteria	Verification	Validation
Question (*B. Boehm*)	Are we building the product *right*?	Are we building the *right* product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	Reviews Walkthroughs Inspections	Testing

- It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user's needs.

- Software Verification & Validation
  - **VERIFICATION vs. VALIDATION**

My Definition of Validation

From *Doug Amos*, 2018

VERIFICATION

- 2 sleeves?
- Is it size L?
- Is it blue?
- Are any buttons missing?

VALIDATION

- Does it fit?
- Is it comfortable to drive in?
- Does the colour match my eyes?
- Can I afford it?
- Is it good quality?
- Will my date like it?

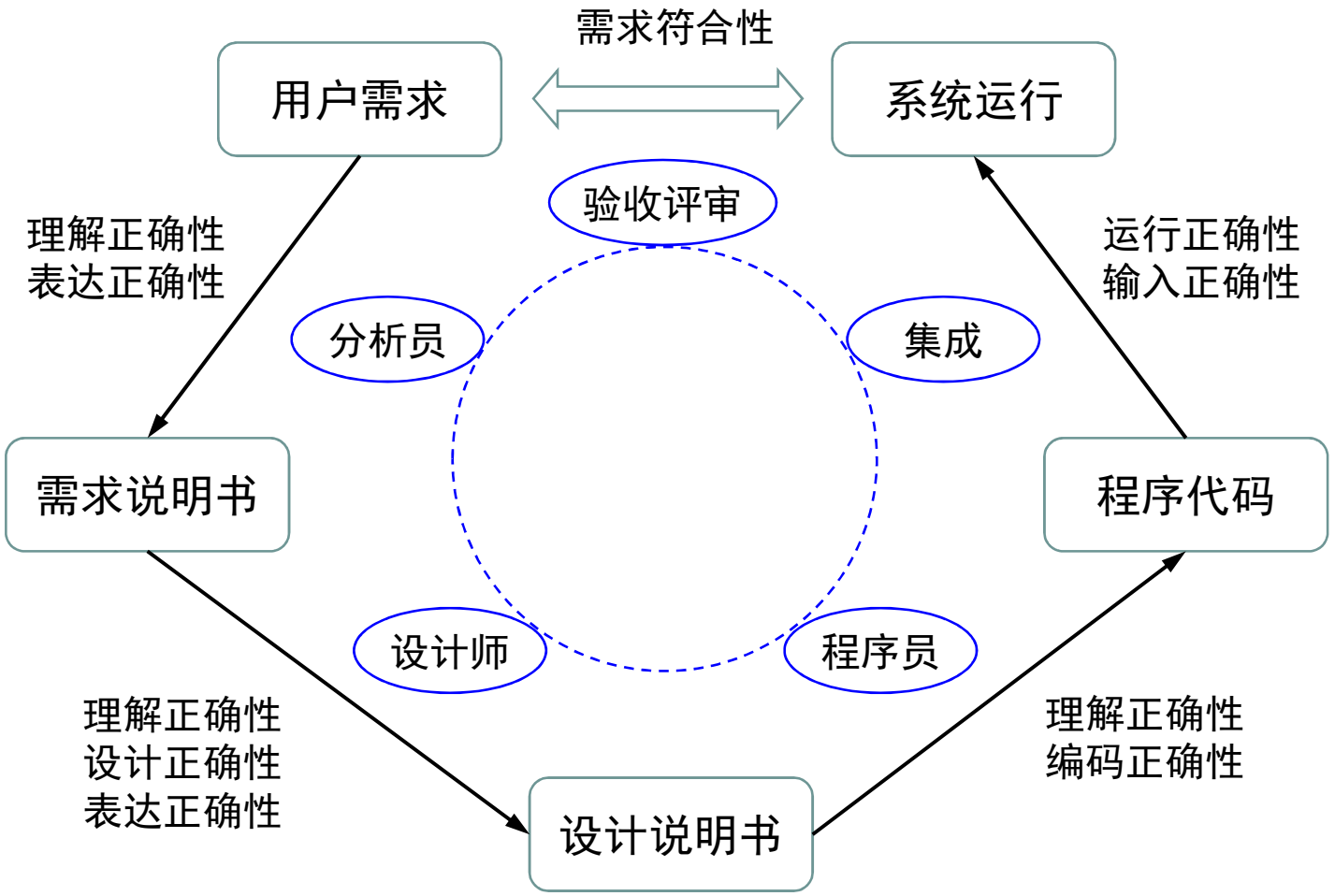
- Software Verification & Validation
  - **Independent Software Verification and Validation**
    - ISVV, deriving from the application of IV&V (Independent Verification and Validation, IEEE 1012) to the software, stands for Independent Software Verification and Validation. ISVV is targeted at safety-critical software systems and aims to increase the quality of software products, thereby reducing risks and costs through the operational life of the software. ISVV provides assurance that software performs to the specified level of confidence and within its designed parameters and defined requirements.
    - ISVV activities are performed by *independent* engineering teams, not involved in the software development process, to assess the processes and the resulting products. The ISVV team independency is performed at three different levels:
      - financial,
      - Managerial, and
      - technical.

- Software Verification & Validation
  - **Independent Software Verification and Validation**
    - ISVV goes far beyond “traditional” verification and validation techniques, applied by development teams. While the latter aim to ensure that the software performs well against the nominal (名义上的) requirements, ISVV is focused on non-functional (非功能性) requirements such as robustness and reliability, and on conditions that can lead the software to fail. ISVV results and findings are fed back to the development teams for correction and improvement.
    - ISVV 的目标：
      - 论证软件的技术正确性，包括安全性和保密性；
      - 评价软件产品的总体质量；
      - 确定是否符合开发过程标准；
      - 提高用户对软件信心；
      - 改进软件质量。

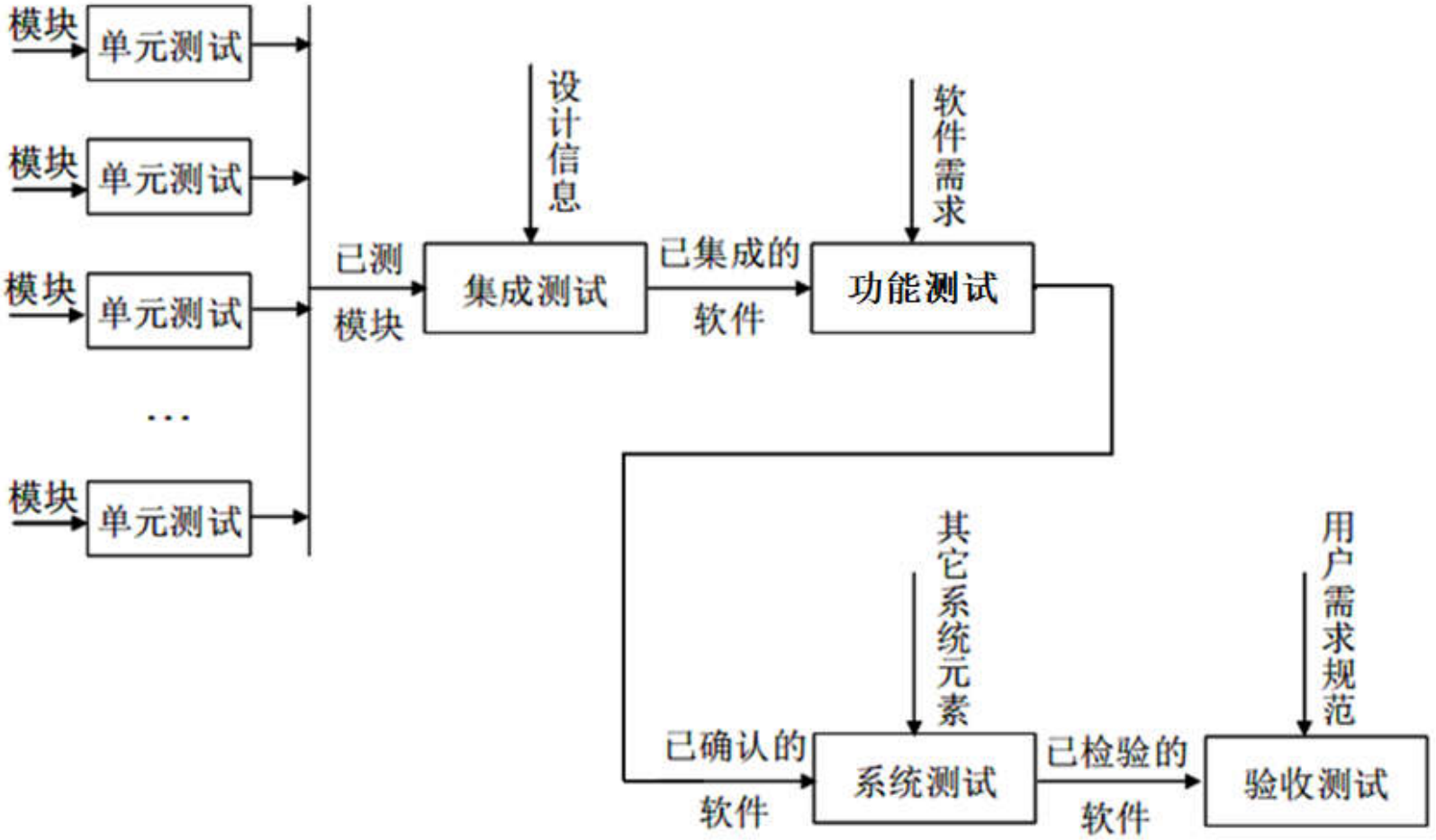
软件测试的生命周期

- 将软件测试阶段性地划分为：测试需求分析、测试计划、测试设计、测试开发、测试执行、测试评估，称为软件测试的生命周期。
- 每一个软件测试项目应该具备的基本流程：
  - 测试需求阅读与评审
  - 测试用例设计与评审
  - 测试环境搭建
  - 执行测试过程
  - 编写测试文档 (测试用例设计、测试报告、问题报告等)
  - 开发工程师与测试经理审核

软件开发与测试的协同作用



- SDLC 测试阶段的分级测试



- SDLC 测试阶段的分级测试
  - 单元测试
    - 目的：检测程序模块中有无故障存在。
    - 对象：软件设计的最小单位，与设计和实现有密切关系。
  - 集成测试
    - 目的：发现与接口有关的模块之间的问题。
    - 方法：非增量式集成测试法和增量式集成测试法。
    - 非增量式集成测试法
      - 对每一个模块进行单元测试；
      - 在此基础上按程序结构图将各模块连接起来，把连接后的程序当作一个整体进行测试。
    - 增量式集成测试法
      - 不断地把待测模块连接到已测模块集 (或其子集) 上，对待测模块进行测试，直到最后一个模块测试完毕。

- SDLC 测试阶段的分级测试
  - 功能测试/确认测试
    - 目的：对软件产品进行评估以确定其是否满足软件功能需求。
    - 方法：基于产品功能说明书，从用户角度进行功能验证，以确认每个功能是否都能正常使用。
  - 系统测试
    - 目的：针对系统中各个组成部分进行综合性检验，证明系统的性能。
    - 内容：包括恢复测试、安全测试、强度测试和性能测试等。
    - 约束：
      - 系统开发人员不能进行系统测试；
      - 系统开发组织者不能负责系统测试。

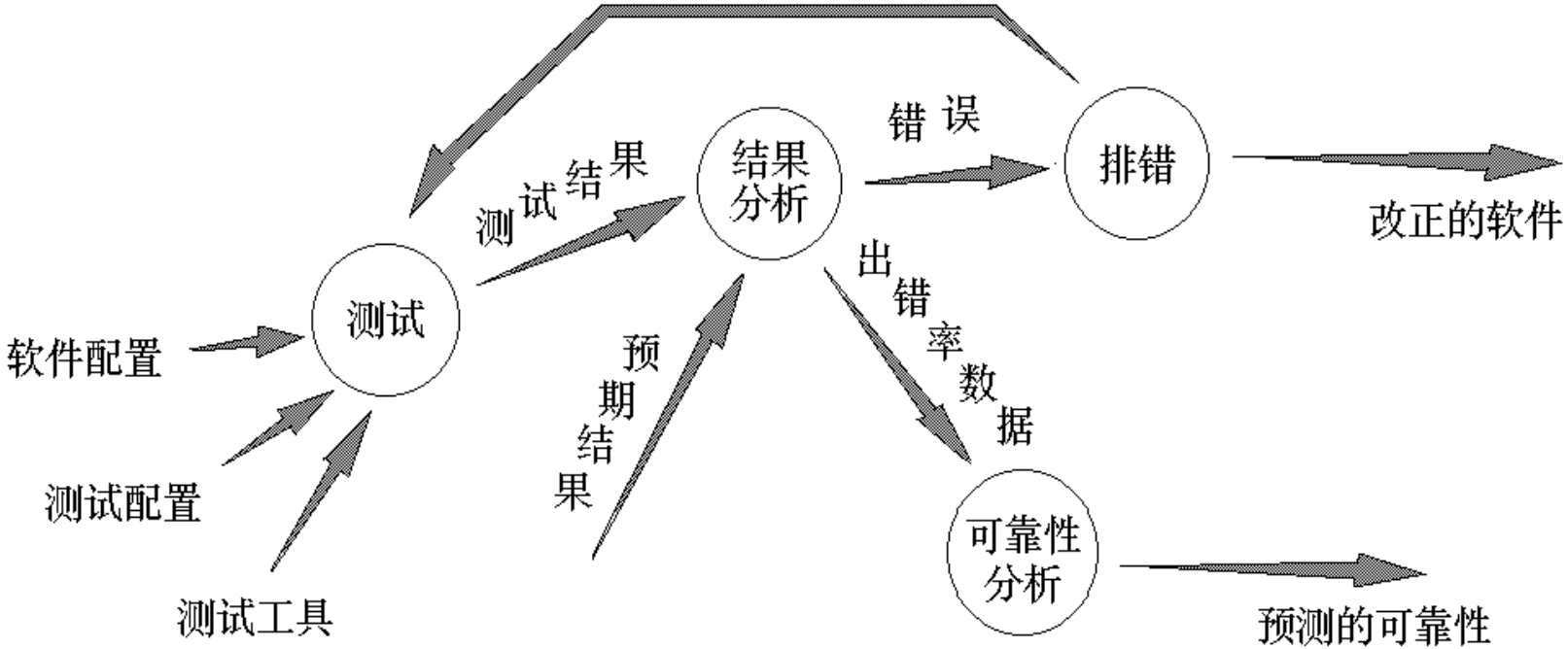
- SDLC 测试阶段的分级测试
  - 验收测试
    - 目的：向用户表明所开发的软件系统能够按照用户预期进行工作。
    - 主要流程：
      - 明确规定验收测试通过的标准
      - 确定验收测试方法
      - 确定验收测试的组织和可利用的资源
      - 确定测试结果的分析方法
      - 制定验收测试计划并进行评审
      - 设计验收测试的测试用例
      - 审查验收测试的准备工作
      - 执行验收测试
      - 分析测试结果，决定是否通过验收

主要软件测试阶段的输入和输出

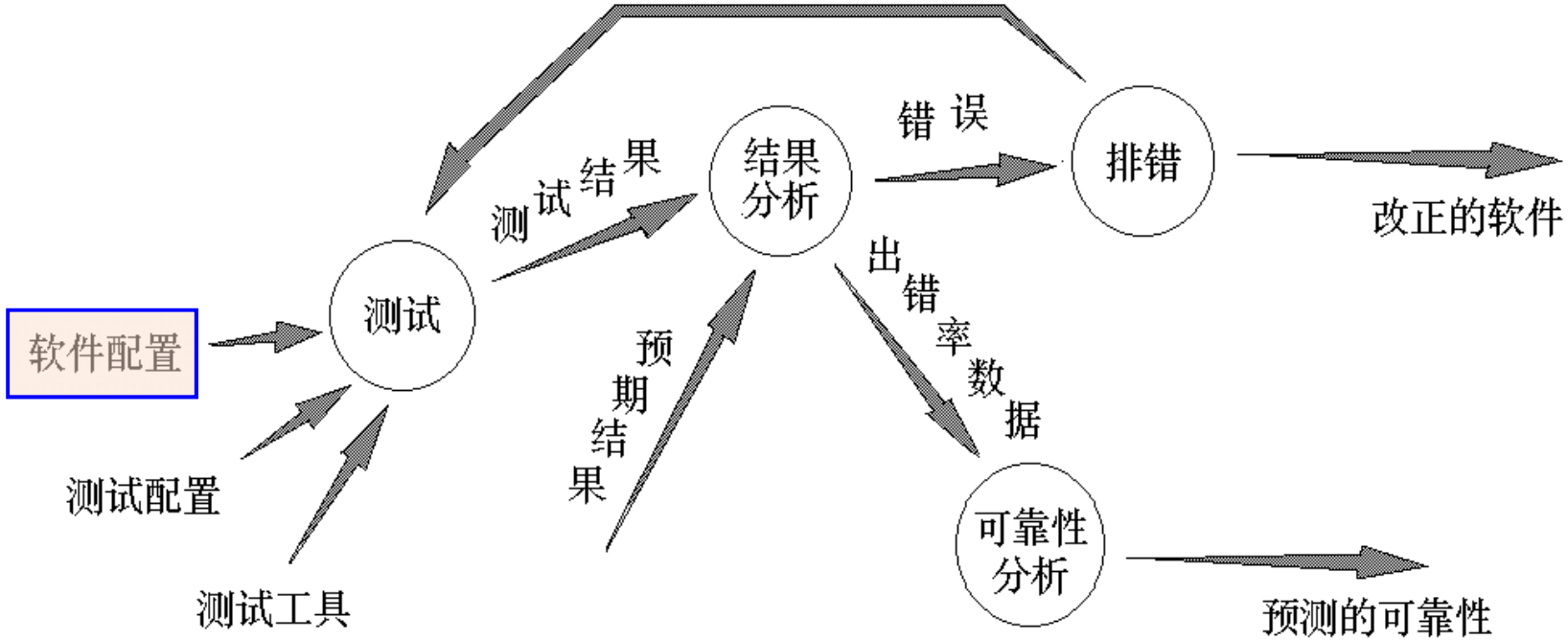
阶 段	输 入	输 出
需求审查	需求定义，市场分析文档，相关技术文档。	市场需求分析会议记要，功能设计，技术设计。
设计审查	市场需求文档，技术设计文档。	测试计划，测试用例。
功能验证	代码完成文件包，功能详细设计说明书，最终技术文档。	完整的测试用例，完备的测试计划，缺陷报告，功能验证测试报告。
系统测试	代码修改后的文件包，完整测试用例, 完备的测试计划。	缺陷报告，缺陷状态报告，项目阶段报告。
确认测试	代码冻结文件包，确认测试用例。	缺陷状态报告，缺陷报告审查，版本审查。
版本审查	代码发布文件包，测试计划检查清单。	当前版本已知问题的清单，版本发布报告。

软件测试的过程和方法

软件测试信息流

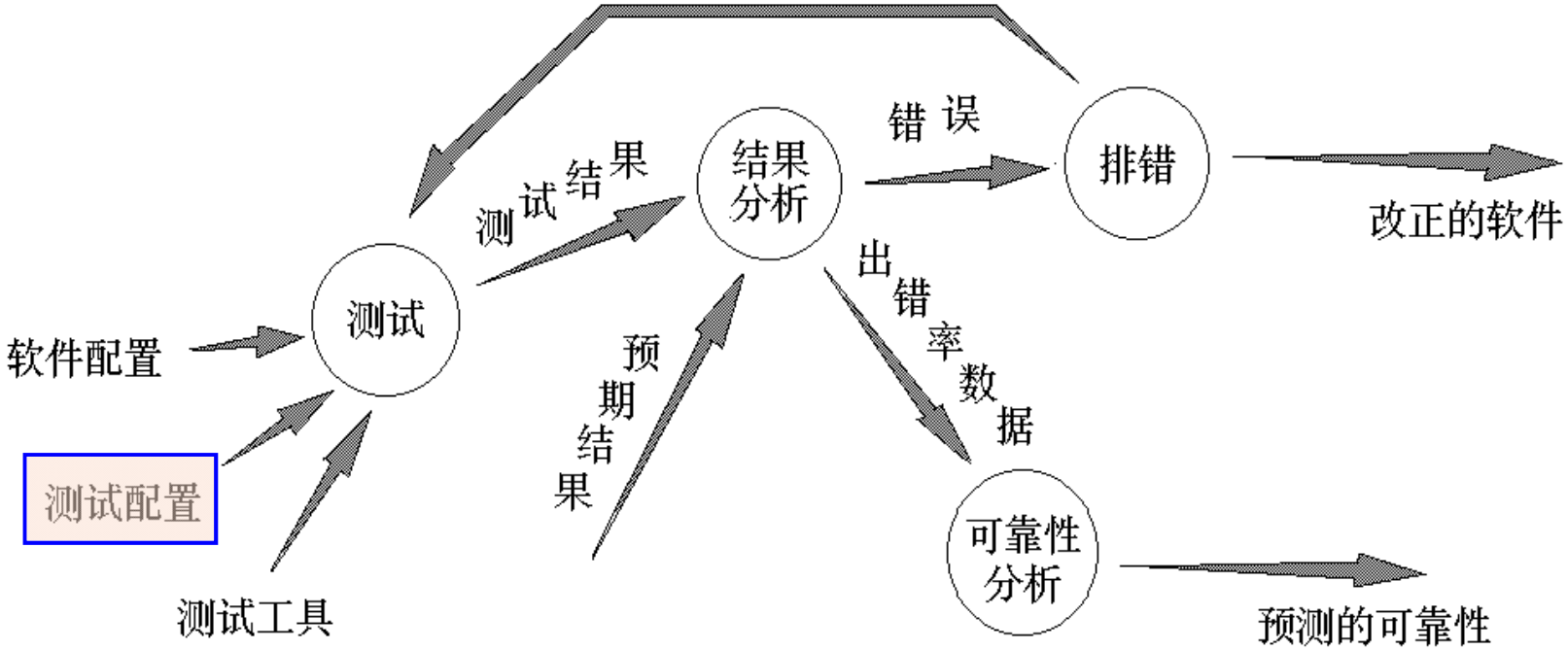


软件测试信息流



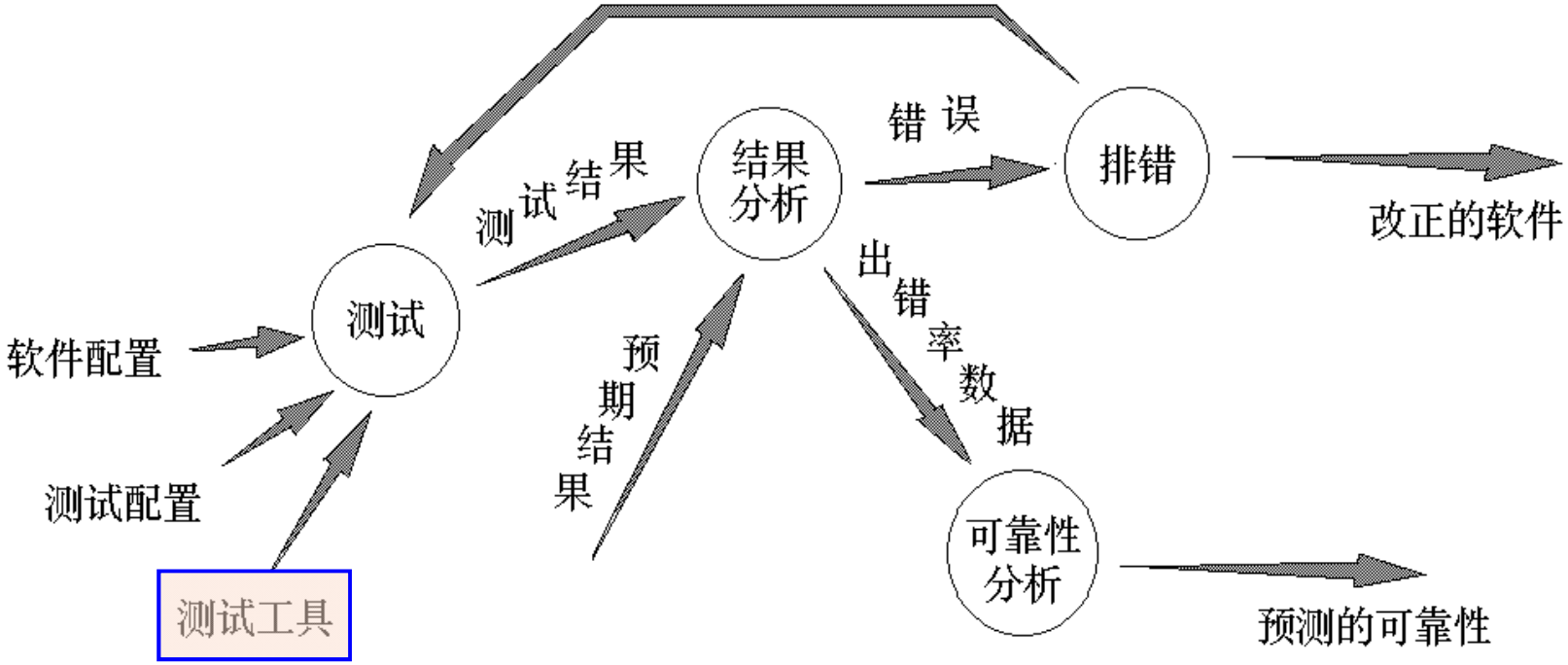
软件配置：被测软件的软件需求规格说明、软件设计规格说明、源代码等

软件测试信息流



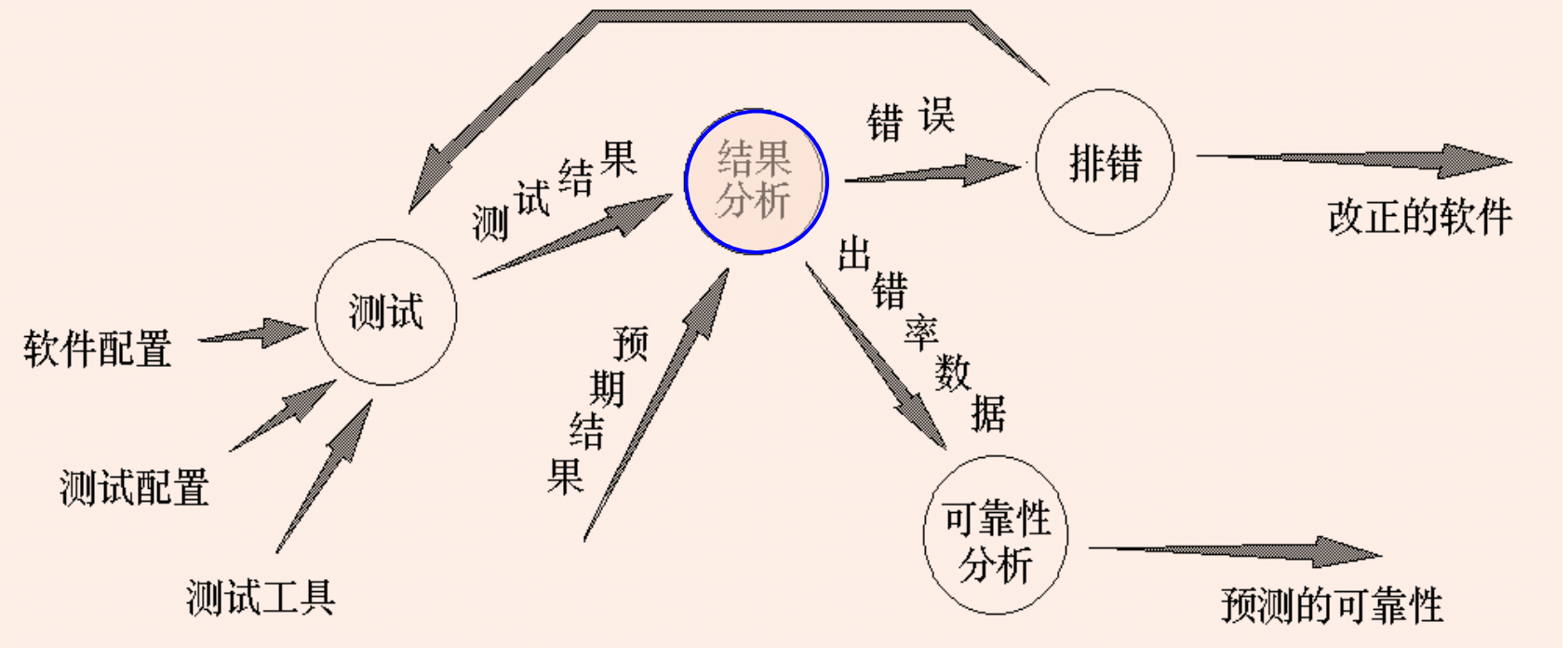
测试配置：测试计划、测试用例、测试程序等

软件测试信息流



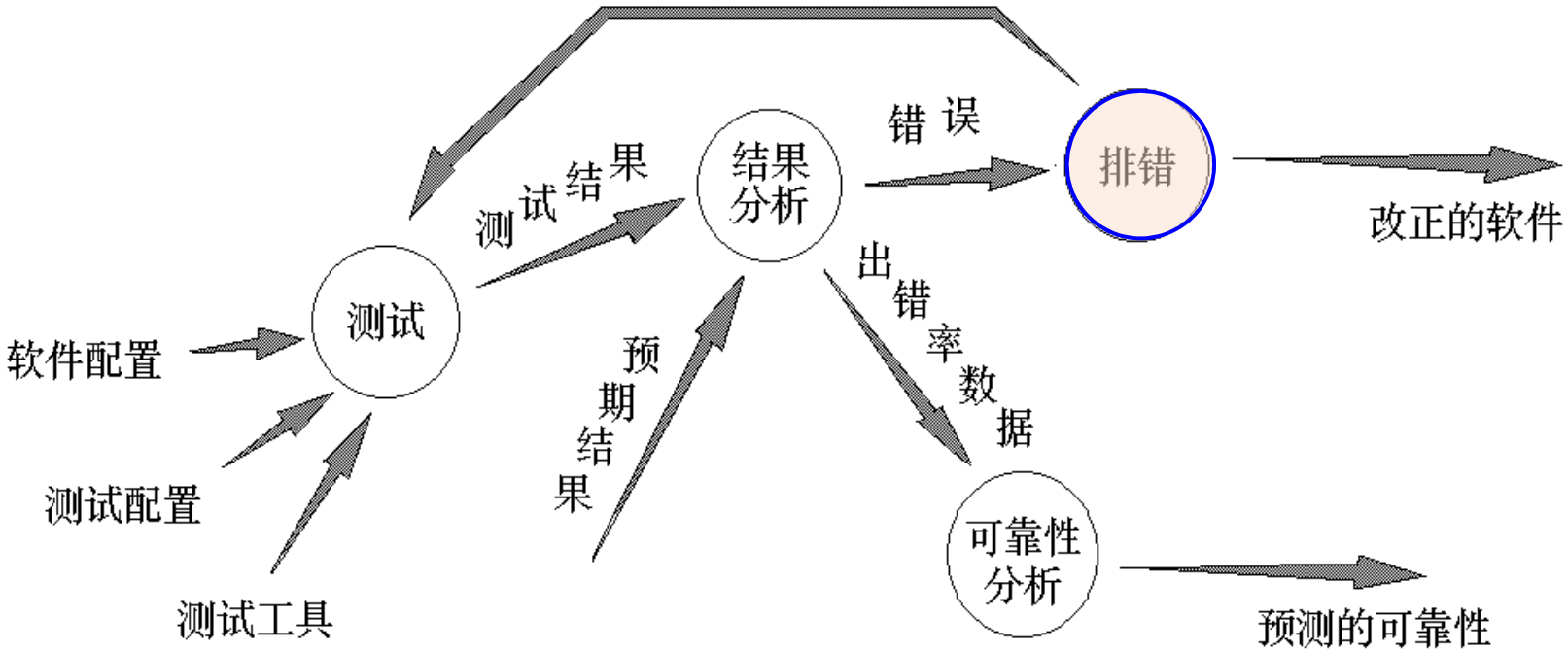
测试工具：测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序、驱动测试的测试数据库等

软件测试信息流



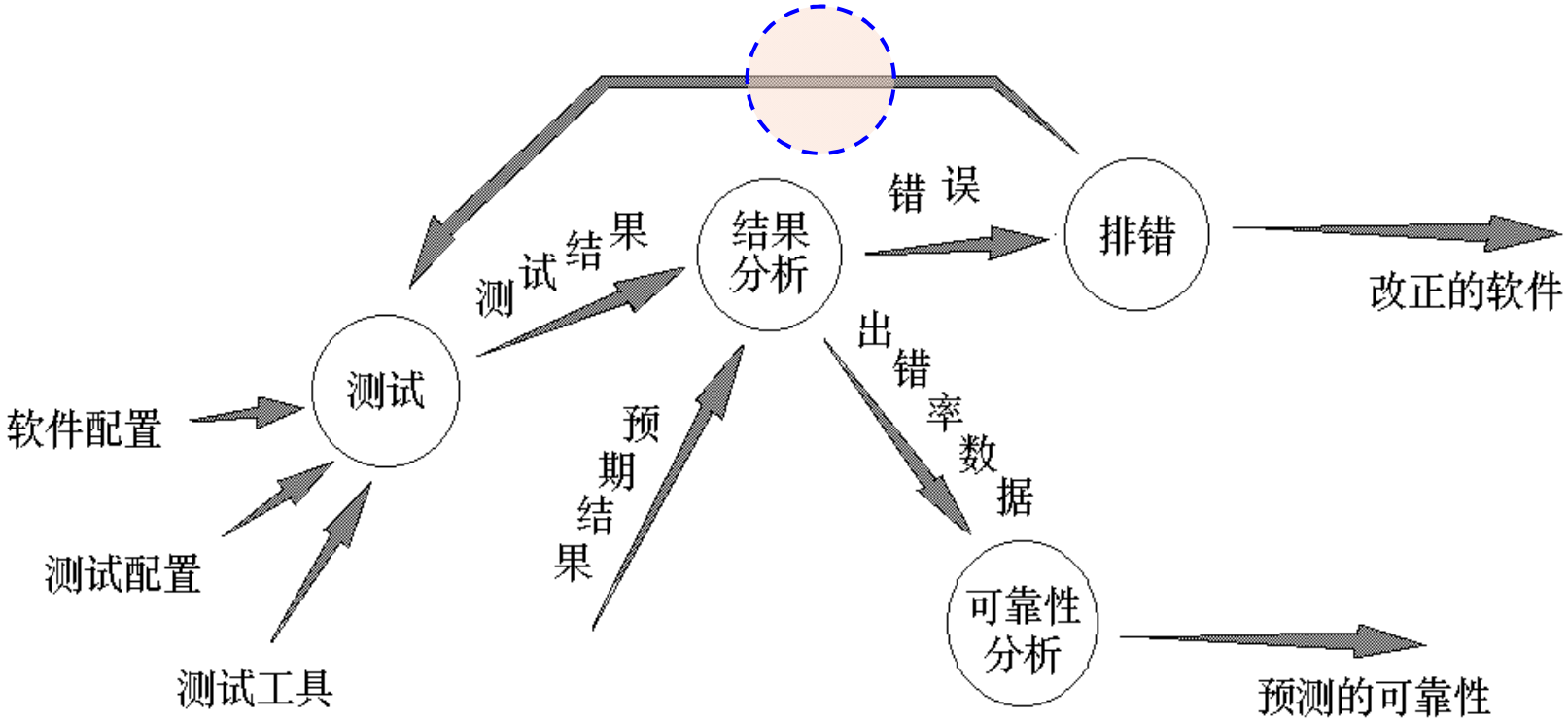
测试结果分析：比较实测结果与预期结果，评价是否发生软件失效。

软件测试信息流



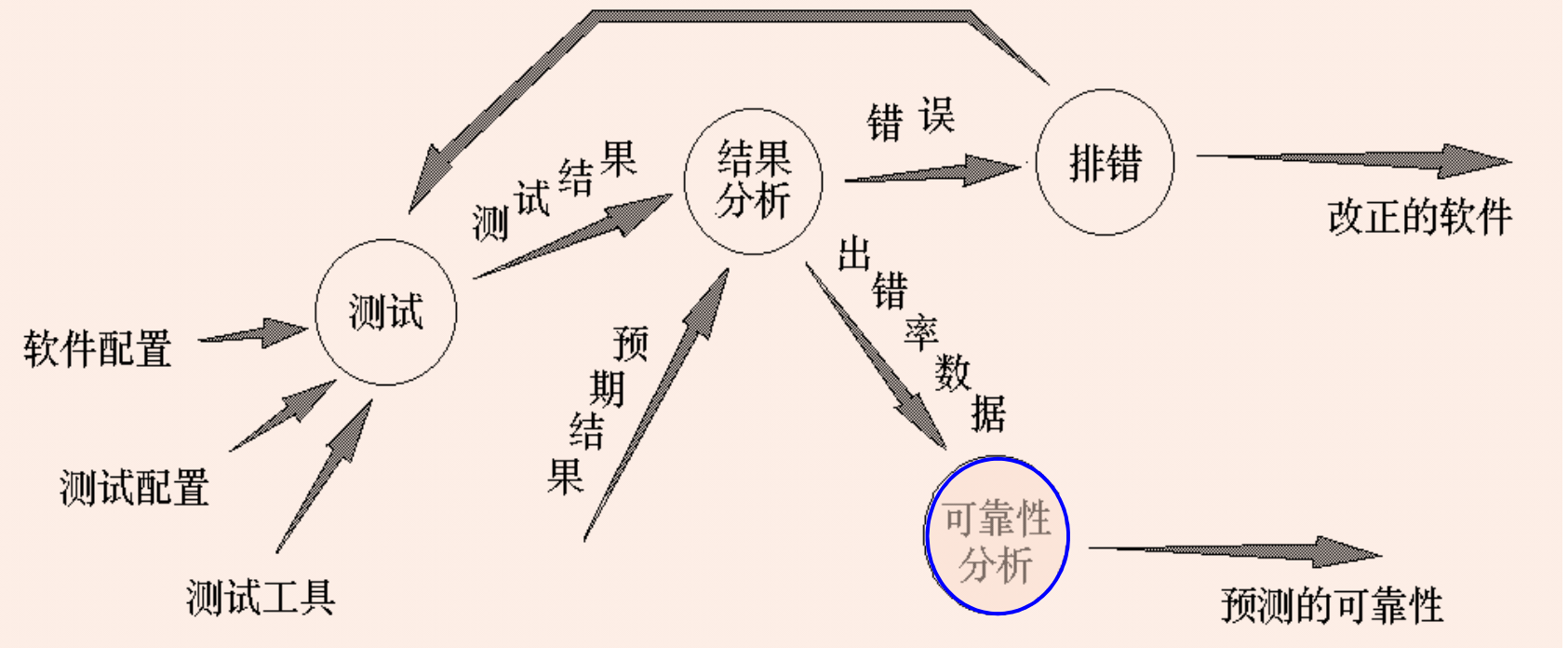
排错：对发现的软件故障进行错误定位，确定出错性质，改正错误，同时修改相关的文档

软件测试信息流



修正后的结果再测试，直到通过测试为止

软件测试信息流



可靠性分析：建立软件可靠性模型，评价软件质量

软件测试信息流

- 软件配置
  - 包括被测软件需求规格说明、软件设计规格说明、源代码等。
- 测试配置
  - 包括测试计划、测试用例、测试程序等。
- 测试工具
  - 包括测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序、以及驱动测试的测试数据库等等。
- 测试结果分析
  - 比较实测结果与预期结果，评价是否发生软件失效。
- 排错 (调试)
  - 对发现的软件故障进行错误定位，确定出错性质，改正错误，同时修改相关的文档。

- 软件测试信息流
  - 修正后的结果再测试
    - 进行回归测试，直到通过为止。
  - 可靠性分析
    - 收集和分析测试结果数据，建立软件可靠性模型。利用可靠性分析技术，评价软件质量：
      - 是否软件的质量和可靠性达到可以接受的程度；
      - 是否所做的测试不足以发现严重的错误。

- 软件测试的基本方法
  - 黑盒测试和白盒测试
  - 静态方法和动态方法
  - 文档、代码审查
  - 数据输入边界条件法
  - 等价划分、数据流程图
  - 状态变换图
  - 逻辑路径法

黑盒测试

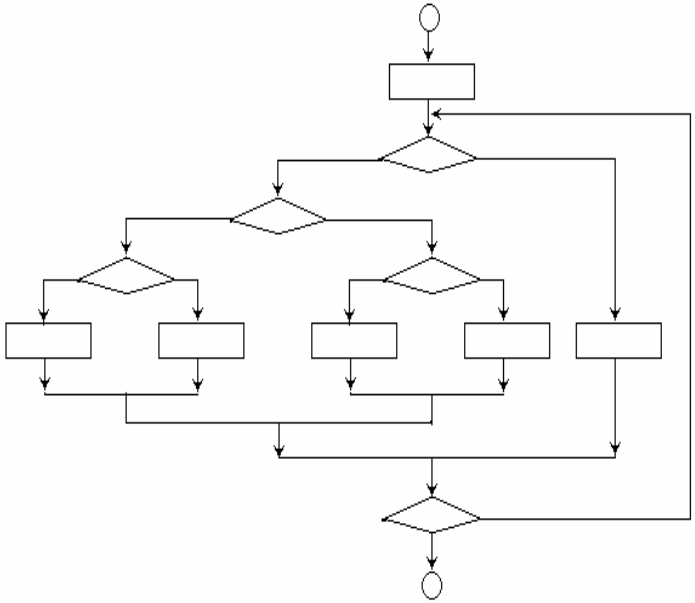
- 黑盒测试又称为功能测试或数据驱动测试。
  - 把测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合说明书的规定。
- 黑盒测试方法在程序接口上进行测试，主要目的是为了发现以下错误：
  - 是否有不正确或遗漏了的功能?
  - 在接口上，输入能否正确地被接受? 能否输出正确的结果?
  - 是否存在数据结构错误或外部信息 (例如数据文件) 访问错误?
  - 性能上是否能够满足要求?
  - 是否存在初始化或终止性错误?

- 黑盒测试
  - 局限性
    - 用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件中确定测试数据，检查程序是否都能产生正确的输出。这个要求在多数情况下难以做到。
    - 例：假设一个程序 P 有输入量 X, Y 及输出量 Z。在字长为32位的计算机上运行。若 X、Y 取整数，按黑盒方法进行穷举测试。可能采用的测试数据组：$232 \times 232 = 264$ 。如果测试一组数据需要1毫秒，一年工作365×24小时，完成所有测试需5亿年。

白盒测试

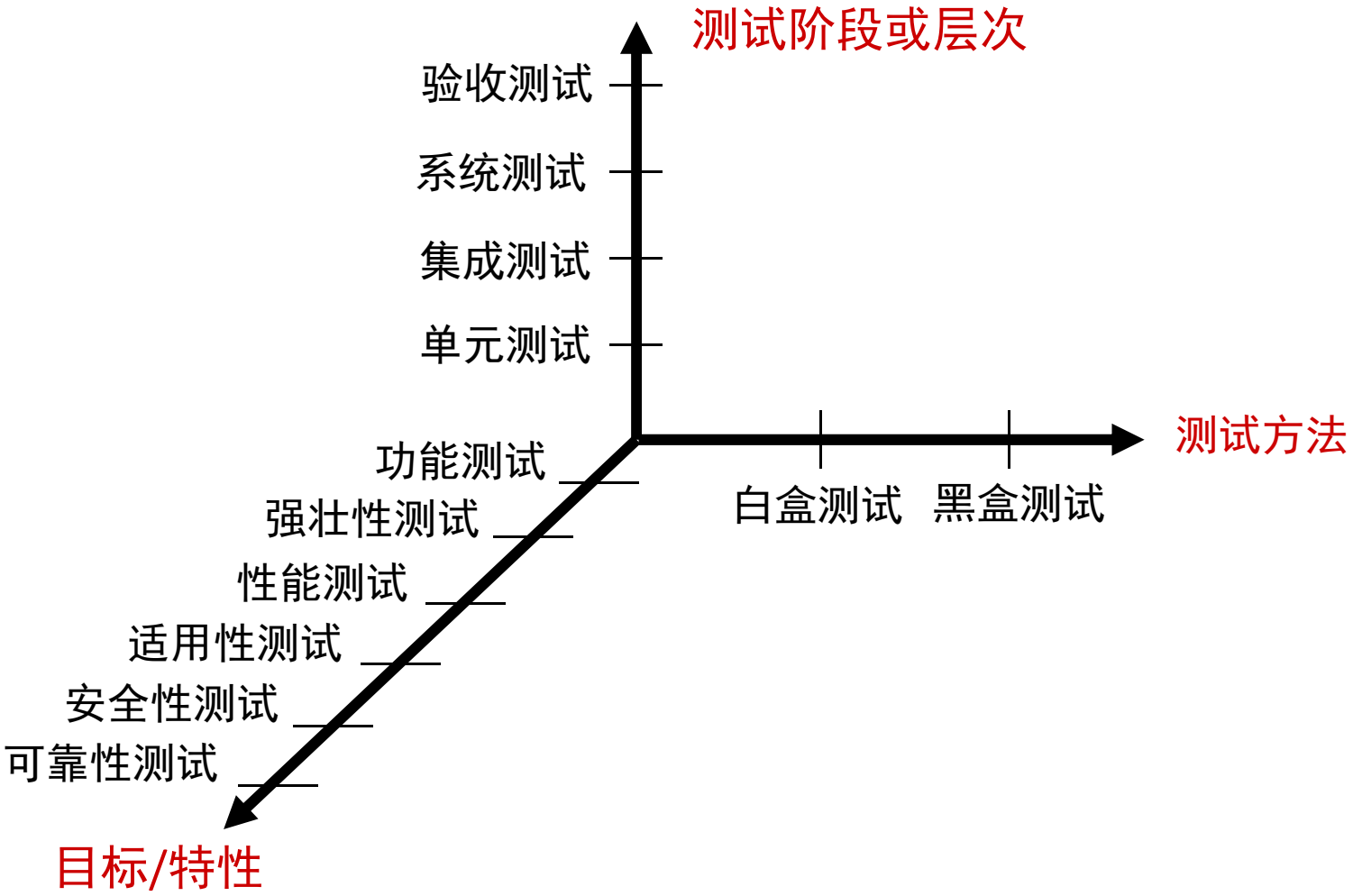
- 白盒测试又称为结构测试或逻辑驱动测试。
  - 白盒测试把测试对象看做一个透明的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序的所有逻辑路径进行测试。
  - 白盒测试在不同逻辑节点检查程序的状态，确定实际的状态是否与预期的状态一致。
- 测试原则：
  - 保证每个模块中所有独立路径至少被使用一次；
  - 对所有逻辑值均以真值和假值进行测试；
  - 在上下边界及可操作范围内运行所有循环；
  - 检查内部数据结构以确保其有效性。

- 白盒测试
  - 穷举路径测试的局限性
    - 对一个具有多重选择和循环嵌套的程序，不同的路径数目可能极其庞大。
    - 例：右面是一个程序的流程图，它包括了一个执行20次的外循环。包含的不同执行路径数达520条。假定对每一条路径进行测试需要1毫秒，一年工作365×24小时，测试完所有路径需要3170年。



- 白盒测试
  - 穷举路径测试的局限性 (续)
    - 不能查出程序违反设计规范，实现了一个非用户需要的功能。
    - 不可能查出程序中因遗漏路径导致的错误。
    - 可能发现不了一些与数据相关的错误。

软件测试阶段、目标和方法的相关性



- 软件测试技术的发展方向
  - 验证技术
    - 形式化数学方法，用于关键应用小程序。
  - 静态测试
    - WalkThrough/Inspection/Review (走查/审查/评审)。
  - 动态测试
    - 选择测试策略、设计测试方式和执行测试用例；
    - 分析和评估测试结果。
  - 自动测试
    - 测试管理和测试执行中都有可以自动测试的地方。
      - 自动测试点选择
      - 注重回归测试

- 软件测试工具
  - 软件测试工具的作用
    - 提高工作效率；
    - 保证测试的准确性、规范性和一致性；
    - 有些测试必须使用工具才能进行 (如性能测试)；
  - 测试工具体现了先进的测试思想、方法和技术，能够快速地提升软件测试的专业化水平。
  - 软件测试工具
    - 目前市场上的测试工具很多，可分为静态测试工具、动态测试工具、黑盒测试工具、白盒测试工具、测试执行评估工具、测试管理工具等。
      - 根据测试需求和实际条件选择购买测试工具，或自行开发相应的测试工具。

- 软件测试工具
  - 软件测试工具平台
    - 软件测试商业化平台包括捕获/回放工具、Web 测试工具、性能测试工具、测试管理工具、代码测试工具等等。
    - 正规的软件测试工具产品版权限制严格，价格昂贵。
      - 可以有限制申请试用版本
    - 开源社区也有一些有效的软件测试工具。

- 软件测试工具
  - 白盒测试工具
    - 静态测试工具
      - 用途：主要集中在需求文档、设计文档以及程序结构的静态测试，可以进行类型分析、接口分析、输入输出规格说明分析等。
      - McCabe & Associates: McCabe Visual Quality ToolSet
      - ViewLog: LogiScope
      - Software Research: TestWork/Advisor
      - Software Emancipation: Discover
      - 北京邮电大学：DTS 缺陷测试工具

- 软件测试工具
  - 白盒测试工具 (续)
    - 动态测试工具
      - 用途：功能确认与接口测试、覆盖率分析、性能分析、内存分析等。
      - Rational: Purify
      - Compuware: DevPartner

- 软件测试工具
  - 黑盒测试工具
    - Rational: TeamTest
    - Compuware: QACenter

- 软件测试工具
  - 测试设计和开发工具
    - 测试设计
      - 说明测试被测软件特征或特征组合的方法，确定并选择相关测试用例的过程。
    - 测试开发
      - 将测试设计转换成具体的测试用例的过程。
    - 测试数据生成器
      - Bender & Associates: SoftTest 功能测试数据生成器
      - Parasoft: Parasoft C++test C/C++ 单元测试工具
    - 基于需求的测试设计工具
    - 捕获/回放工具和覆盖分析工具
    - ……

- 软件测试工具
  - 测试执行和评估工具
    - 测试执行和评估
      - 执行测试用例并对结果进行评估的过程，包括选择用于执行的测试用例、设置测试环境、运行所选择的测试、记录测试执行活动、分析潜在的软件故障并测量测试工作的有效性。
    - 工具类型
      - 捕获/回放
      - 覆盖分析
      - 存储器测试

- 软件测试工具
  - 测试管理工具
    - 测试管理
      - 帮助完成测试计划，跟踪测试运行结果等。
    - 用途：
      - 测试用例管理
      - 缺陷跟踪管理
      - 配置管理
    - Rational: Test Manager
    - Compuware: TrackRecord

- 软件测试工具
  - 市场主流测试工具

    (以 MI, Rational 和 Compuware 的产品为主导)
    - MI (HP-Mercury Interactive)
      - LoadRunner (性能测试)
      - WinRunner (功能回归测试)
      - TestDirector (测试管理工具)
      - QTP (Quicktest Pro. 简单的功能回归测试)
    - IBM-Rational
      - Rational Robot (功能/性能测试工具)
      - Rational Purify (白盒测试工具)
      - Rational Testmanager (测试管理工具)
      - Rational ClearQuest (缺陷/变更管理工具)

- 软件测试工具
  - 市场主流测试工具 (续)
    - Compuware
      - QACenter (自动黑盒测试工具)
      - DevPartner (自动白盒测试工具)
      - Vantage (应用级网络性能监控管理软件)

- 软件测试和软件开发过程的关系
  - 软件的生存周期
    - 软件开发过程一般包括六个阶段，构成了软件的生存周期：
      - 第1阶段 计划
      - 第2阶段 需求分析
      - 第3阶段 设计
      - 第4阶段 编码
      - 第5阶段 测试
      - 第6阶段 运行和维护

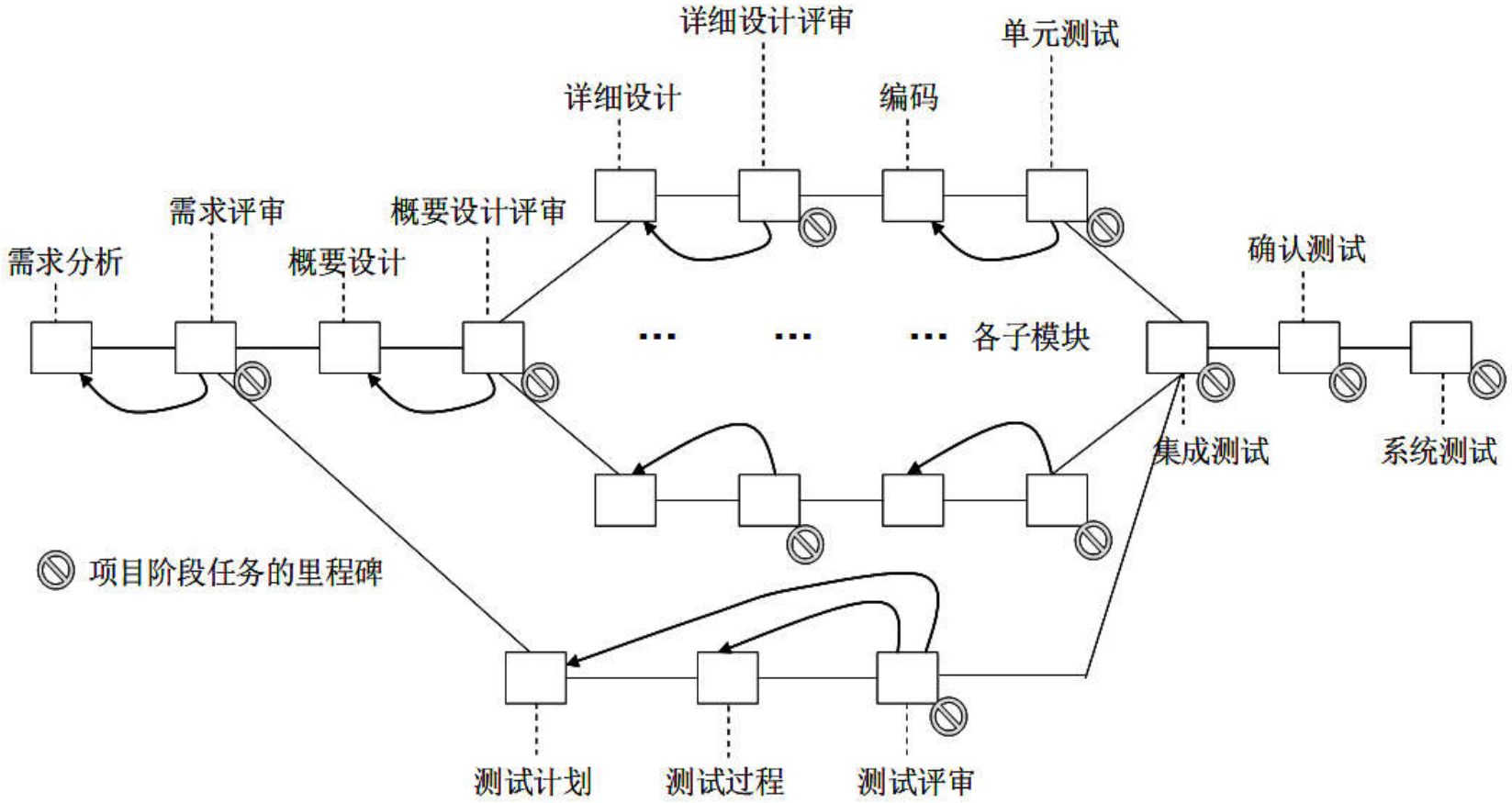
- 软件测试和软件开发过程的关系
  - 软件开发过程和软件测试过程的比较
    - 软件开发过程是一个自顶向下，逐步细化的过程。
      - 计划阶段定义软件作用域；
      - 需求分析建立软件信息域、功能和性能需求、约束等；
      - 设计阶段把需求分析结果用系统设计语言进行形式描述；
      - 编码阶段把详细设计用程序设计语言转换成程序代码。
    - 软件测试过程是依相反顺序自底向上，逐步集成的过程。
      - 对每个程序代码模块进行单元测试，消除程序模块内部逻辑和功能上的错误和缺陷；
      - 对照软件设计说明进行集成测试，检测和排除子系统或系统结构上的错误；
      - 对照需求说明进行确认测试；
      - 最后配置系统环境，实地运行系统，进行系统测试。

- 软件测试和软件开发过程的关系
  - 软件开发过程和软件测试过程的比较
    - 软件开发工作的根本是让软件产品尽量实现用户的需求。
    - 软件测试工作的根本是检验软件产品是否满足用户的需求。
      - 软件测试的主要内容是验证和确认软件的设计、开发是否符合软件用户的需求。
        - 验证：保证软件正确地实现了用户要求的业务功能，即保证软件实现了需求所规定的内容。
        - 确认：使用开发完成的软件体验各项业务流程，证实在一个给定的外部环境中软件逻辑的正确性。
      - 不论是软件模块还是整个系统，它们有着共同的测试内容，如正确性测试、容错性测试、性能与效率测试、易用性测试、文档测试等。

- 软件测试和软件开发过程的关系
  - 软件开发过程和软件测试的协同作用
    - 项目规划阶段
      - 制定整个测试阶段的监控策略。
    - 需求分析阶段
      - 确定测试需求分析，制定系统测试计划；
      - 测试需求分析指产品生存周期中测试所需的资源、配置、各阶段评审通过的标准等。
    - 概要设计和详细设计阶段
      - 制定集成测试计划和单元测试计划。
    - 编码阶段
      - 开发相应的测试代码或测试脚本。
    - 测试阶段
      - 实施测试，并提交相应的测试报告。

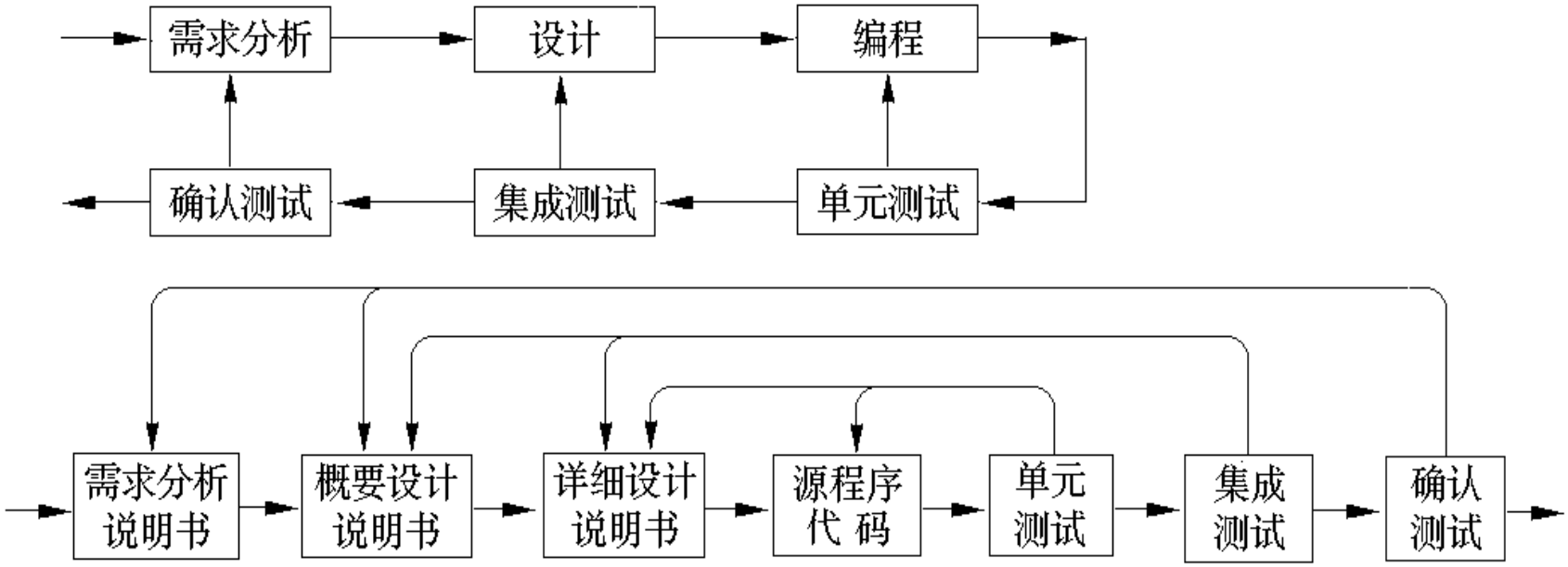
软件测试和软件开发过程的关系

- 软件开发过程和软件测试的协同作用



软件测试和软件开发过程的关系

软件开发过程和软件测试的协同作用



软件开发过程对应的软件测试

软件测试的误区

- 在软件测试工作中存在的一些误区
  - 整体认识上重开发而轻测试；
  - 软件开发完成后再进行软件测试；
  - 软件测试是为了证明软件的正确性；
  - 软件发布后如果发现质量问题，那是软件测试人员的过错；
  - 软件测试的技术要求不高，测试人员可以临时随意指派；
  - 软件测试员是软件程序员的对头；
  - 软件测试是测试人员的事情，与软件程序员无关；
  - 项目进度紧张时少做些测试，时间富裕时多做测试；
  - 软件测试是没有前途的工作，软件开发程序员才是高手；
  - 软件测试就是程序调试，测试发现了错误说明程序员编写的程序有问题；

软件测试的误区

- 在软件测试工作中存在的一些误区 (续)
  - 期望用自动化测试工具代替大部分的人工劳动；
  - 所有软件缺陷都可以修复；
  - 软件测试文档只是用于备案，并不重要；
  - 期望短期通过增加软件测试投入，迅速达到软件零缺陷率；
  - 规范化的软件测试不必要地增加了项目开发总成本。

Thank you!