SOFTWARE TESTING

Chapter 2

Basic Concepts of Software Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

- 2.1 软件缺陷
- 2.2 软件测试概述
- 2.3 软件测试的过程和方法
- 2.4 基于软件生命周期的软件测试方法
- 2.5 软件测试的分类与分级

软件缺陷

软件缺陷的概念

什么是软件缺陷

- A software defect is any problem/disfigurement/limitation in product design & development
- A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. (Wiki)
- 任何计算机程序或系统中的问题、与产品设计书的不一致性、不能满足用户的需求等都会形成软件缺陷。

软件缺陷的概念

- 软件缺陷的主要表现
  - 约定的功能、特性没有实现或只是部分实现。
  - 设计不合理。
  - 实际结果和预期结果不一致。
  - 数据结果不正确、精度不够。
  - 运行出错，包括运行中断、系统崩溃、界面混乱。
  - 用户界面不友好。
  - 性能低下。
  - 用户不能接受的其他问题。
    - 如存取时间过长、界面不美观。

软件缺陷的概念

软件缺陷的主要表现

- Feature or function can’t work
- Unreasonable design
- Partly realization in function
- Data error
- Run error
- Limitation in features
- Difference between actual results and expected results
- Unfriendly UI
- Low performance
- Others.

软件缺陷的概念

- *Ron Patton* (Software Testing, 2005)：
  - A software bug occurs when one or more of the following five rules is true:
    - The software doesn't do something that the product specification says it should do.
    - The software does something that the product specification says it shouldn't do.
    - The software does something that the product specification doesn't mention.
    - The software doesn't do something that the product specification doesn't mention but should.
    - The software is difficult to understand, hard to use, slow, or in the software tester's eyes will be viewed by the end user as just plain not right.

软件缺陷的概念

- *Ron Patton* (Software Testing, 2005)：
  - 至少满足以下5个规则之一，才称为发生了一个软件缺陷
    (1) 软件未实现产品说明书要求的功能。
    (2) 软件出现了产品说明书指明不应该出现的错误。
    (3) 软件实现了产品说明书未提到的功能。
    (4) 软件未实现产品说明书虽未明确提及但应该实现的目标。
    (5) 软件难以理解，不易使用，运行缓慢或者—从测试员的角度看—最终用户会认为不好。

软件缺陷的概念

- 软件缺陷将导致软件故障
  - 计算机系统的规模和复杂性急剧增加，计算机软件的数量快速膨胀。与此同时，计算机出现故障引起系统失效的可能性也逐渐增大。
  - 随着计算机硬件技术的进步，元器件可靠性的提高，硬件设计和验证技术的成熟，硬件质量冗余度的提高，软件故障正逐渐成为导致计算机系统失效和停机的主要因素。

软件缺陷

- Spectacular Software Failures
  - NASA's Mars lander
    - September 1999, crashed due to a units integration fault
  - THERAC-25 radiation machine
    - Poor testing of safety-critical software can cost lives : 3 patients were killed
  - Ariane 5 explosion
    - exception-handling bug : forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)
  - Intel's Pentium FDIV fault
    - Public relations nightmare
  - Northeast Blackout of 2003
    - The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system.
    - Financial losses of $6 Billion USD

软件缺陷的5级分级体系

- Fatal
  - 致命的。缺陷造成系统或应用程序崩溃、死机、系统挂起，或造成数据丢失、主要功能完全丧失等。
- Critical
  - 严重的。缺陷导致某些功能或特性没有实现，主要功能部分丧失，次要功能完全丧失，或发出致命的错误声明。
- Major
  - 不太严重的。缺陷虽然不影响系统的使用，但没有很好地实现系统功能，没有达到预期效果。例如次要功能丧失，提示信息不太准确，用户界面差，操作时间长等。

- 软件缺陷的5级分级体系
  - Minor
    - 小的问题。缺陷对系统功能几乎没有影响，产品及属性仍可使用。
  - Suggestion
    - 建议改进。

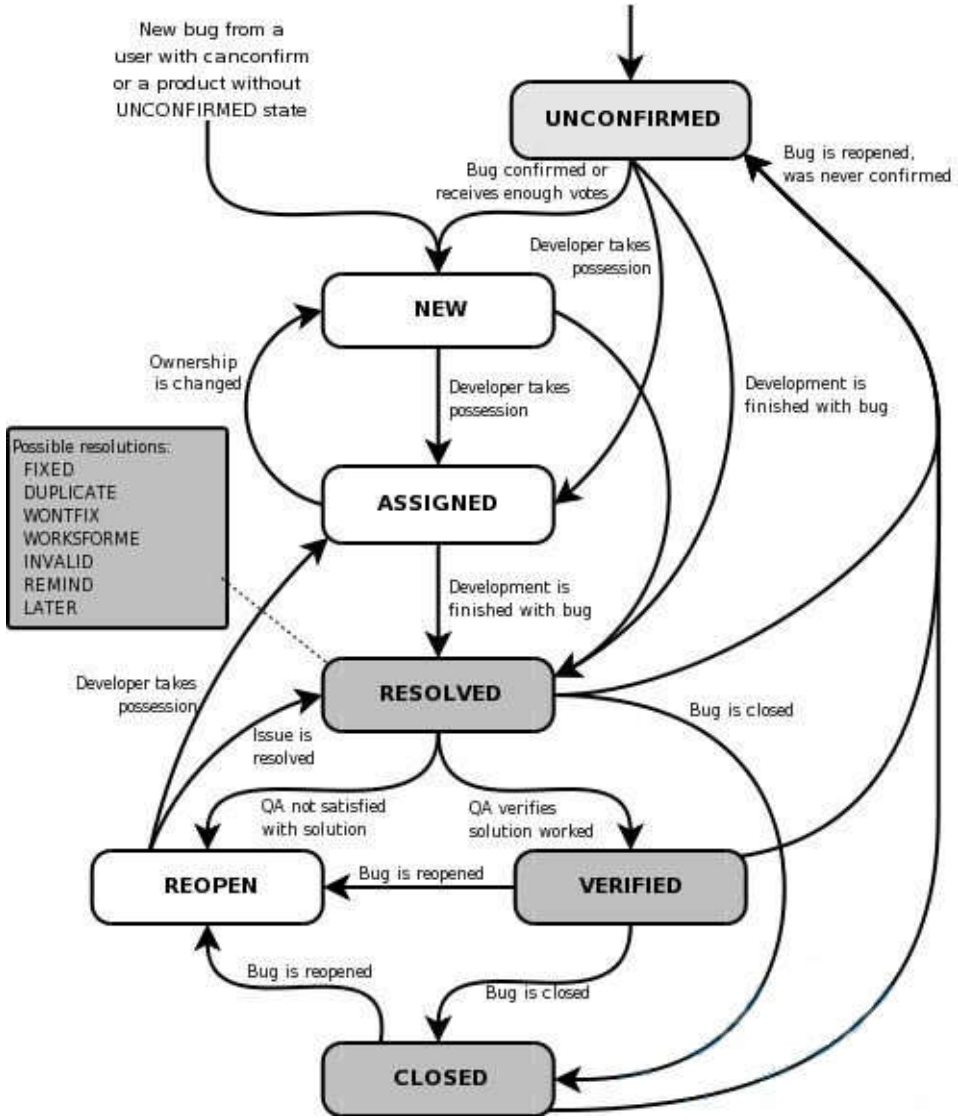
软件缺陷状态

缺陷状态	描　述
激活或打开 Active or Open	缺陷还没有解决，在等待处理中，如新报告的缺陷。
已修正或修复 Fixed or Resolved	缺陷已被开发人员检查、修复过，通过单元测试，但还没有被测试人员验证。
关闭或非激活 Closed or Inactive	缺陷经测试人员验证后，确认已经修复、或不存在之后的状态。
重新打开 Reopen	缺陷经测试人员验证后还依然存在，等待开发人员进一步修复。
推迟 Deferred	这个缺陷可以在下一个版本中解决。
保留 On hold	由于技术原因或第三方软件的缺陷，开发人员不能修复的缺陷。

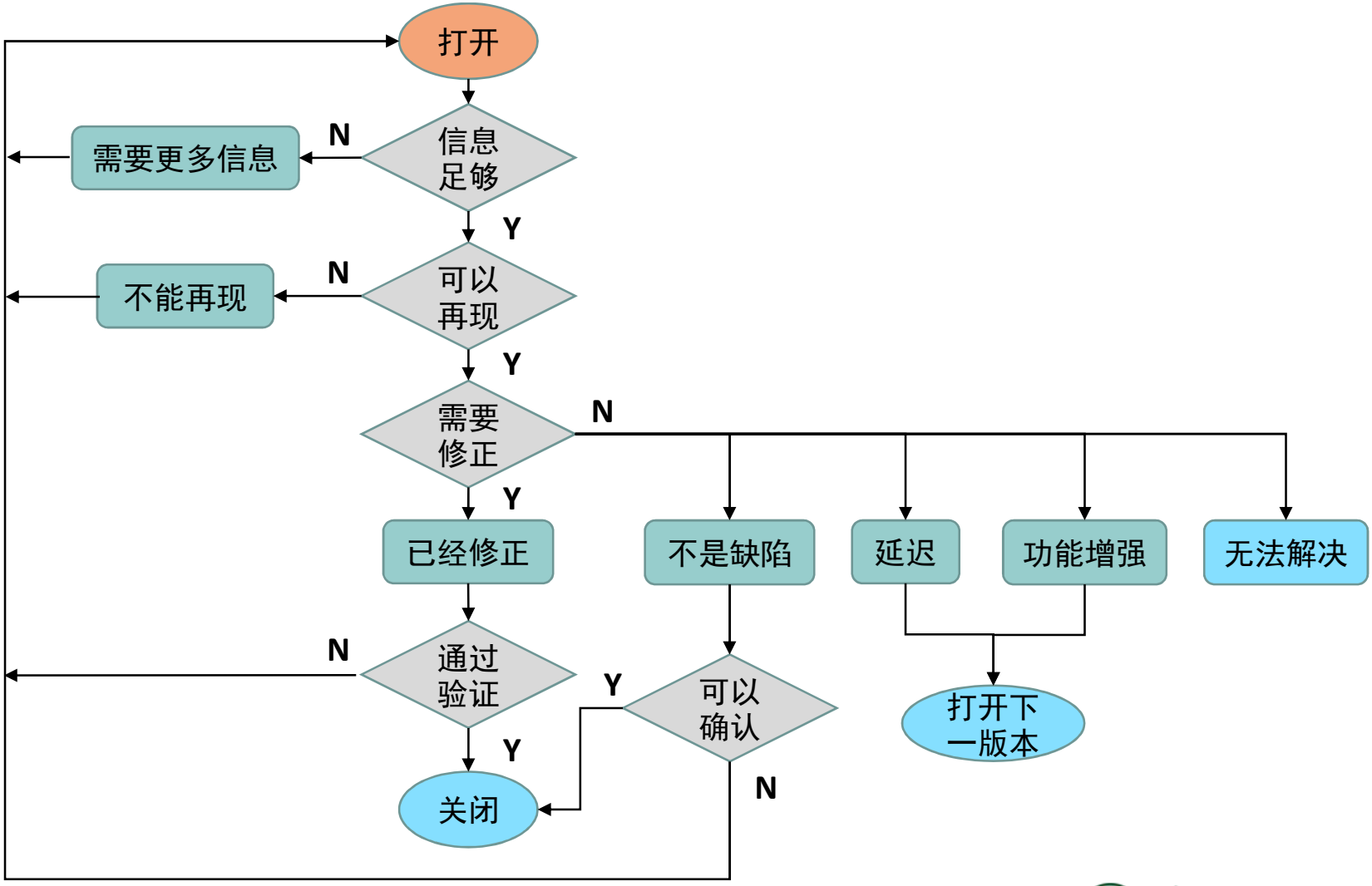
软件缺陷状态

缺陷状态	描　述
不能重现 Can not duplicate	开发者不能复现这个缺陷，需要测试人员检查缺陷复现的步骤。
需要更多信息 Need more info	开发能复现这个缺陷，但开发者需要一些信息，如：缺陷的日志文件，图片等。
重复 Duplicate	这个缺陷已经被其他的软件测试人员发现。
不是缺陷 Notabug/Reject	这个问题不是软件缺陷。
需要修改软件规格说明书 Spec modified	由于软件规格说明书对软件设计的要求，软件开发人员无法修复这个缺陷，必须修改软件规格说明书。

软件缺陷的生命周期


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
REMIND
LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

软件缺陷的生命周期


打开
信息足够
N
需要更多信息
Y
可以再现
N
不能再现
Y
需要修正
N
Y
已经修正
不是缺陷
延迟
功能增强
无法解决
通过验证
N
Y
可以确认
Y
打开下一版本
关闭
N

- 软件缺陷的产生原因
  - 团队协作问题
    - 沟通不准确或不充分产生的误解。
  - 技术问题
    - 算法错误，语法错误，计算和精度问题。
    - 接口参数匹配问题。
  - 软件本身问题
    - 文档错误或描述不足、用户场景 (User Scenario) 不清晰。
    - 时间不协调，或不一致。
    - 系统的自我恢复或数据的异地备份、灾难性恢复问题。
    - 软件可靠性缺少度量的标准。
    - 软件难以维护、不易升级。

软件缺陷

- 软件缺陷的术语
  - 常用术语
    - Error 错误
    - Defect/Bug 缺陷
    - Fault 故障
    - Failure 失效
    - Anomy 异常
    - Variance 偏差
    - Incident 事故
    - Inconsistency 矛盾/不一致
    - Problem 问题
  - 软件失效过程
    - Software Error → Software Fault → Software Failure
      - 需求、设计、编码、测试等阶段都可以是软件错误的来源并导致系统运行失效。

软件缺陷

- Software Faults, Errors & Failures
  - Error (错误：病因)
    - An incorrect internal state that is the manifestation of some fault
      - E.g., high blood pressure, irregular heartbeat, bacteria in the blood stream
  - Fault (故障：有病)
    - A static defect in the software (like design mistakes in hardware)
      - E.g., the ailment (疾病)
  - Failure (失效：故障的动态表现)
    - External, incorrect behavior with respect to the requirements or other description of the expected behavior
      - E.g., a list of symptoms (一系列症状)
  - The word “Bug” is used informally.
    - A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Software Faults, Errors & Failures

```
public static int numZero (int arr[])
{
      // Effects: If arr is null throw NullPointerEx
      // else return the number of occurrences of 0
    int count = 0;
    for (int i = 1; i < arr.
        if (arr[i] == 0) {
            count++;
        }
    }
    return count;
}
```

Fault: Should start searching at 0, not 1

Error: *i* is 1, not 0, on the first iteration
Failure: none

Error: *i* is 1, not 0, on the first iteration
Error propagates to the variable count
Failure: count is 0 at the return statement

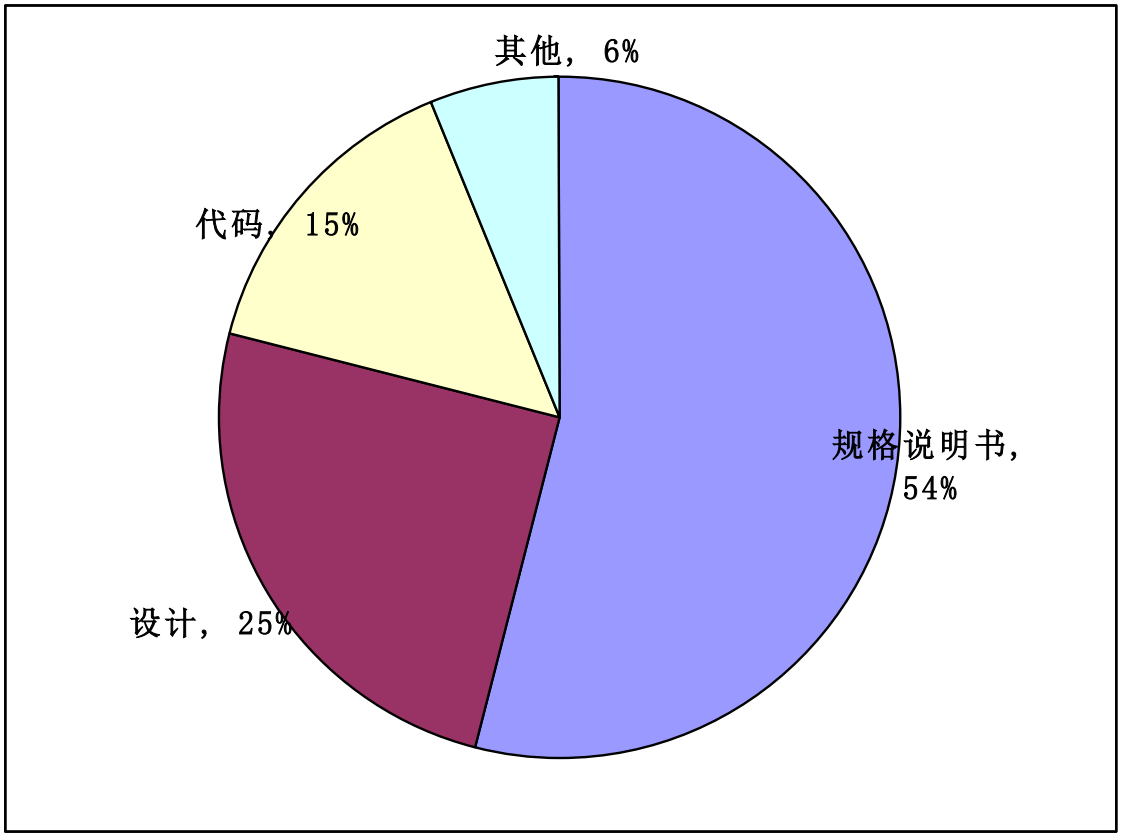
Test 1
{ 2, 7, 0 }
Expected: 1
Actual: 1

Test 2
{ 0, 2, 7 }
Expected: 1
Actual: 0

- An *error* may occur because of mistyping.

软件缺陷的构成

- 软件缺陷的来源 (Source)
  - 缺陷来源即缺陷的直接起因，是导致缺陷的错误所在的位置。

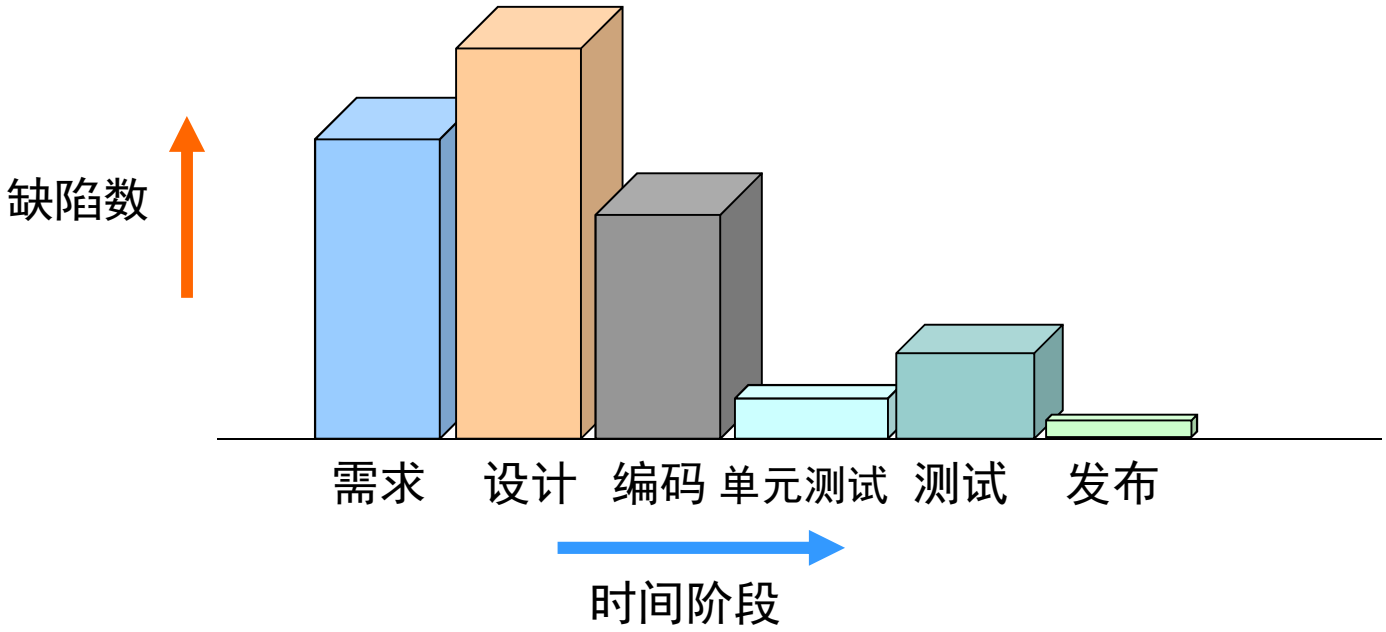


软件缺陷的构成

- 软件缺陷来源超过一半来自于规格说明书，其主要原因是：
  - 需求分析阶段的沟通障碍
    - 产品功能：软件开发人员和非计算机专业用户对要开发的产品的功能理解不一致。
    - 产品特性：由于软件产品还没有设计、开发，对于产品的表现只能凭经验去估计和想象，有些特性还不够清晰。
  - 用户的需求变化
    - 总在不断变化的用户需求如果没有在产品规格说明书中得到正确的最终描述，会导致说明书的上下文矛盾。
  - 主观因素的影响
    - 对规格说明书重视不够，在设计和写作上投入不足。
    - 经常只有设计师或项目经理才会得到比较全面的产品规格信息。

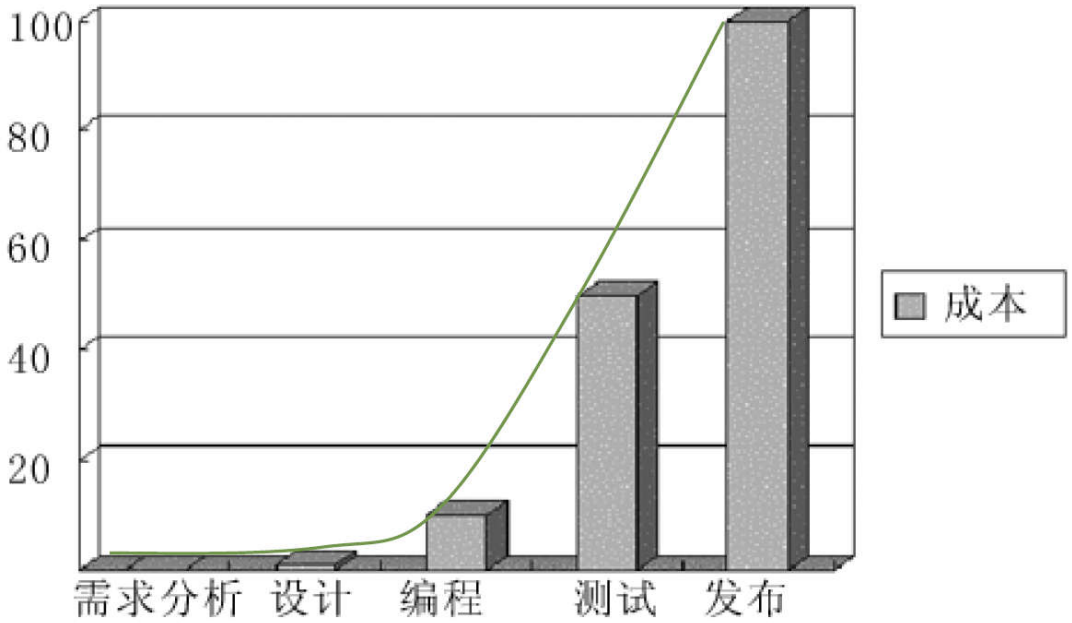
软件缺陷的构成

- 软件缺陷的起源 (Origin) 分布
  - 缺陷起源是缺陷引起的故障或失效事件被首次检测到的阶段。



  - 早期缺陷发现数：测试阶段之前发现的缺陷数占 70%-90%。
    - 在程序测试之前，通过审查和评审可以发现更多的缺陷。

软件缺陷修复成本



- 未能在早期发现并及时修复的软件缺陷，将在后期导致极大的修复成本。

- 软件测试的发展历程
  - 起步阶段
    - 20世纪50-60年代
      - 英国计算机科学家图灵给出了软件测试的原始定义：测试是程序正确性证明的一种极端实验形式。
      - 20世纪60年代 (软件工程建立前)：软件测试的目的是为了表明程序代码的正确性。
    - 20世纪70年代
      - 随着计算机硬件技术的进步与成熟，软件在整个计算机系统中的地位越来越重要，软件规模和复杂性大大增加，软件的可靠性面临前所未有的危机，给软件测试工作带来了挑战。
      - 软件测试的意义逐渐被人们认识，软件测试的研究开始受到重视，这是软件测试技术发展最活跃的时期。

- 软件测试的发展历程
  - 起步阶段
    - 20世纪70年代 (续)
      - 1972年首届软件测试正式会议在北卡罗来纳大学举行。
      - 1975年 *J. B. Goodenough* 和 *S. L. Gerhart* 发表研究论文 Towards a Theory of Test Data Selection (测试数据选择原理)，首次提出了软件测试理论，软件测试被确定为一种研究方向，软件测试这一实践性很强的学科被提高到理论的高度。
      - 1979年，*G. J. Myers* 在首版 The Art of Software Testing (软件测试艺术) 一书中指出：测试是为发现错误而执行的一个程序或者系统的过程。

- 软件测试的发展历程
  - 软件质量标准阶段
    - 20世纪80年代早期开始“软件质量管理”
      - 软件测试的定义发生了改变，测试不单纯是一个发现错误的过程，而且包含了软件质量评价的内容。
      - 各类软件质量评价标准被制定出来。
      - 1983年，*Bill Hetzel* 在 Complete Guide of Software Testing (软件测试完全指南) 一书中指出：测试是以评价一个程序或者系统属性为目标的任何一种活动，测试是对软件质量的度量。

- 软件测试的发展历程
  - 20世纪90年代，测试工具开始盛行
    - 1996年开始提出：
      - 测试能力成熟度 TCMM (Testing Capability Maturity Model)
      - 测试支持度 TSM (Testability Support Model)
      - 测试成熟度 TMM (Testing Maturity Model)
      - 测试过程改进 TPI (Testing Process Improvement)
  - 21世纪
    - 测试的生命周期理论被提出
      - 2002年，*Rick D. Craig* & *Stefan P. Jaskiel* 在 Systematic Software Testing (系统的软件测试) 一书中对软件测试做了进一步的定义：测试是为了度量和提高被测软件的质量而对被测软件进行工程设计、实施和维护的整个生命周期过程。

软件测试的发展历程

- Testing in the 21st Century
  - Software defines behavior
    - network routers, finance, switching networks, other infrastructure.
  - Today's software market
    - much bigger, more competitive, more users.
  - Embedded Control Applications
    - airplanes, air traffic control
    - spaceships
    - watches
    - ovens
    - remote controllers.
  - Agile processes put increased pressure on testers
    - Programmers must unit test
    - Tests are key to functional requirements.
  - Software is a skin that surrounds our civilization.

软件测试的发展历程

- Testing in the 21st Century
  - More safety critical, real-time software.
  - Embedded software is ubiquitous … check your pockets
    - 嵌入式软件无处不在
  - Enterprise applications means bigger programs, more users.
  - Paradoxically, free software increases our expectations !
    - 不可思议的是，自由软件反而提高了我们对软件的期望
  - Security is now all about software faults.
    - Secure software is reliable software.
  - The web offers a new deployment platform.
    - Very competitive and very available to more users
    - Web apps are distributed
    - Web apps must be highly reliable.

软件测试的概念

- IEEE Std 610.12-1990
  - An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
  - 软件测试是在特定条件下运行或测定某个系统或组件的活动，对结果进行观察或记录，并针对其某一方面给出评价。
  - 软件测试根据软件开发各阶段的规格说明和程序的内部结构设计测试用例，并利用这些测试用例去执行程序，以期发现软件故障。
- 软件测试的目的
  - 目的在于检验被测对象是否满足规定的需求，发现预期结果与实际结果之间的差别。寻找软件故障是软件测试的主要目的。
  - 软件测试是一种软件质量保证活动，其动机是通过一些经济有效的方法，发现软件中存在的缺陷，从而保证软件质量。

软件测试的概念

- *G. J. Myers*
  - 软件测试是为了发现错误而执行程序的过程。
  - 一个好的测试能够在第一时间发现程序中存在的错误。
  - 一个好的测试是发现了至今尚未发现的错误的测试。
- 软件测试检查由开发小组开发的软件，测定软件的状态，而不是决定该软件是否能成为产品。
  - 从用户的角度出发，希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品。
  - 从软件开发者的角度出发，希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。

软件测试的概念

- 综合
  - 软件的质量要求是软件测试的基础。
    - 软件质量水平应该具有明确定位；衡量软件质量的唯一标准是软件产品与需求的符合程度。以上两点使得测试人员在发现问题时有明确的处理原则，避免了测试的随意性。
  - 软件测试证明软件的功能和性能与需求说明相符合。
  - 软件测试以最少的时间和人力，系统地发现软件中潜在的各种错误和缺陷。
  - 软件测试收集到的测试数据为软件可靠性分析提供了依据。
  - 软件测试不能表明软件中不存在错误，它只能说明软件中存在错误。
  - 软件测试仍然是软件工程发展较为薄弱的一个方面。不仅测试理论不够完整，已有的测试方法也不能满足当前软件开发的实际需求。

软件测试的重要性

- 软件测试是在软件投入运行前，对软件需求、设计和编码的最终复审，是软件质量保证的关键步骤。
- 软件设计与编码过程是引入错误的过程，而软件测试是通过测试排除软件错误的过程。
- 测试在软件开发中占有重要地位。
  - 测试成本占开发成本的 30%-50%。
- 软件测试在软件开发过程中的具体作用：
  - 确认软件产品存在哪些属性和缺乏哪些属性；
  - 验证需求是否正确，确认最终产品符合用户真正的要求；
  - 对专门的过程进行分类测试，评价产品的合理性。

- Testing Goals Based on Test Process Maturity
  - Level 0
    - There's no difference between testing and debugging.
      - Testing is the same as debugging,
      - Does not distinguish between incorrect behavior and mistakes in the program,
      - Does not help develop software that is reliable or safe.

- Testing Goals Based on Test Process Maturity
  - Level 1
    - The purpose of testing is to show correctness.
      - Purpose is to show correctness.
        - Correctness is impossible to achieve.
      - What do we know if no failures?
        - Good software or bad tests?
      - Test engineers have no:
        - Strict goal,
        - Real stopping rule,
        - Formal test technique.
      - Test managers are powerless.

- Testing Goals Based on Test Process Maturity
  - Level 2
    - The purpose of testing is to show that the software doesn't work.
      - Purpose is to show failures.
        - Looking for failures is a negative activity (寻求软件失效是一种消极活动).
      - Puts testers and developers into an adversarial relationship.
      - What if there are no failures?

- Testing Goals Based on Test Process Maturity
  - Level 3
    - The purpose of testing is not to prove anything specific, but to reduce the risk of using the software.
      - Testing can only show the presence of failures,
      - Whenever we use software, we incur some risk,
      - Risk may be small and consequences unimportant,
      - Risk may be great and consequences catastrophic (灾难性的),
      - Testers and developers cooperate to reduce risk.

- Testing Goals Based on Test Process Maturity
  - Level 4
    - Testing is a mental discipline (智力训练) that helps all IT professionals develop higher quality software.
      - Testing is the only one way to increase quality,
      - Test engineers can become technical leaders of the project,
        - Primary responsibility to measure and improve software quality,
        - Their expertise should help the developers.

Seven Principles of Software Testing 软件测试的原则

- Software testing is an extremely creative and intellectually challenging task. When testing follows the principles given below, the creative element of test design and execution rivals any of the preceding software development steps.
- **1. Testing shows presence of defects** 测试显示软件存在缺陷
  - Testing is done to find as many defects as possible.
  - Testing is not done to prove that the software is bug-free.
  - Testing helps us to identify potential undiscovered defects in the software.
  - Even after testing is complete with no bugs found, we can't say that the product is 100% defect free.

- Seven Principles of Software Testing 软件测试的原则
  - **2. Exhaustive Testing is impossible** 穷尽测试是不可能的
    - Unless the application under test (AUT) has a very simple logical structure and limited input, it is impossible to test all possible combinations of data and scenarios.
    - Testing all combinations would cost more time and money.
    - Smarter ways of testing should be adopted in order to complete the project timelines.
    - Prioritize important tests based on risk.
  - **3. Early Testing** 测试的尽早介入
    - Software testing starts from requirement gathering.
    - Testing should be done parallelly along with other product development phases.
    - Finding defects early on will save a lot of money and rework time.

- Seven Principles of Software Testing 软件测试的原则
  - **4. Defect Clustering** 缺陷集群性
    - Defects are not necessarily spread even (缺陷并非必然地平坦分布).
      - They may be clustered in one or few modules.
      - Place where one bug found, likely to find some more
    - It's more like 80-20 Pareto principle that is 20% of the defects cause 80% of the problem
  - **5. Pesticide Paradox** 杀虫剂悖论
    - Executing same test cases again and again will not help us to identify new defects.
    - To overcome this, it is really important to review test cases regularly and modify them if required.
    - Different and new test cases should be added to find more defects in the software

- Seven Principles of Software Testing 软件测试的原则
  - **6. Testing is context dependent** 测试活动依赖于测试背景
    - Testing is always context dependent.
    - Software should be tested based on the purpose of the product that has been built.
    - Different methodologies, techniques and types of testing is related to the type and nature of the application.
    - Banking site cannot be tested same as the E-commerce website.
  - **7. Absence-of-errors fallacy** 不存在缺陷的谬误
    - Finding and fixing defects doesn't help if the built software fail to meet user's requirements.
    - Test designed should catch most defects and also should cover all client's requirements and specifications before shipping the product.

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - The 6TH Principle of Software Testing: Testing is context dependent.
  - *Paul Gerrard* introduced the idea of Test Axioms in posts on his blog in the spring of 2008. Over a few months their definitions evolved and in May 2008 he summarised the thinking behind them and tabulated 16 proposed axioms. Further evolution has occurred and with some changes. *Paul* believes that there are a set of rules or principles that provide a framework for all testing.
  - An axiom is something believed to be true, but cannot be proven in any practical way. It could be disproven by experiment or experience.
    - Some people object to the notion of Test Axioms. In their opinion, nothing in testing is certain. There are no axioms. All testing rules, principles, techniques, approaches etc. are heuristic. Heuristics have value in some contexts, but are limited in application, usefulness, accuracy etc. in other contexts. They are limited or fallible (不可靠) in known ways.

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Definition of test that is context-neutral
    - *Test: (noun) a procedure for critical evaluation; a means of determining the presence, quality, or truth of something; a trial.*
  - Definition of testing
    - *Test: (verb) to critically evaluate; to determine the presence, quality, or truth of something; to conduct a trial.*
  - Test Axioms (16)
    - Stakeholder Axioms (4)
    - Test Design Axioms (6)
    - Delivery Axioms (6)

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Stakeholder Axioms (4)
    - Testing is an information or intelligence-gathering activity performed on behalf of (people we will call) testing stakeholders. The manager who asked you to test could be your most important stakeholder. They think testing is important enough to get someone as important as you involved – but might not be able to articulate (清楚说明) why they see it as an important role.
    - By the way, if you are testing the products of your own efforts, you could be your own stakeholder. Your approach to testing your own products or systems will be focused on what you and others, as stakeholders, want to learn about those products or systems.

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Stakeholder Axioms (4)

    (1) The Stakeholder Axiom: Testing needs stakeholders
    - Identify and engage the people or organisations that will use and benefit from the test evidence we are to provide.
      - Who are they?
      - Whose interests do they represent?
      - What evidence do they want?
      - What do they need it for?
      - When do they want it?
      - In what format?
      - How often?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Stakeholder Axioms (4)

    (2) The Value Axiom: The value of evidence is for the stakeholder to decide

    - The outcome of a test and the way evidence is presented defines its value, regardless of its source.
      - What acceptance decisions must stakeholders make?
      - What evidence do stakeholders need to make these decisions with confidence?
      - When can the required evidence be gathered?
      - Who needs to provide subject-matter expertise to inform the testing (and make it valuable)?
      - Who are best placed to perform these tests?
      - Are the people or organisations nominated to perform the tests capable of doing so?
      - What environment and infrastructure is required to make the testing meaningful and valuable?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Stakeholder Axioms (4)

    (3) The Scope Management Axiom: If we don't manage scope, we may never meet stakeholder expectations
    - Testers need to identify and agree the items in and out of scope and manage change of scope over time.
      - How do stakeholders define the scope of the system and what needs testing?
      - What sources of knowledge are available to define and understand scope in detail?
      - What are the (likely) drivers for change?
      - How will testers accommodate changes of scope?
      - How will testers analyse the impact of change of scope?
      - Do testers have the authority to resist or challenge change?
      - How will defect fixes be assured (through re-testing)?
      - How will testers test changes?
      - How will testers communicate the status of changes?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Stakeholder Axioms (4)

    (4) The Good-Enough Axiom: The scope of testing and acceptance are always compromises

    - Stakeholders and testers must jointly appreciate that there is no limit to testing and that the acceptance decision will always be made on incomplete evidence. In fact, acceptance may occur in spite of evidence, based on information known only to stakeholders.
      - How much evidence from testing will be required to make the acceptance decision?
      - Who is authorised to make the acceptance decision?
      - What is the mechanism for assessing the value of evidence gathered during testing?
      - What coverage model(s) can be used to judge that enough evidence has been gathered?
      - What criteria will be used to judge that the system under test is acceptable or unacceptable?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)
    - Testing is a process in which we create mental models of the environment, the system, human nature, and the tests themselves. Test design is the process by which we select, from the infinite number possible, the tests that we believe will be most valuable to us and our stakeholders. Our test model helps us to select tests in a systematic way. The Test Basis is the source, or sources, of knowledge required to select what aspects of our system we might test and how we might test them.
    - A Test Oracle (测试预言) is the source, or sources, of knowledge that enable us to predict the outcome of any test. In effect, an oracle tells us what a system does in all circumstances and situations. Implicitly, if an oracle does this, it is perfect. Our oracle might be the same sources as the test basis. Whatever our sources of knowledge are, like test bases, they are fallible (不可靠的).

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)
    - **(5) Test Model Axiom: Test design is based on models**
      - Choose test models to derive tests that are meaningful to stakeholders. Recognise the models' limitations and the assumptions that the models make.
        - Are design models available to use as test models?
        - Are they mandatory?
        - What test models could be used to derive tests from the Test Basis?
        - Which test models will be used?
        - Are test models to be documented or are they purely mental models?
        - What are the benefits of using these models?
        - What simplifying assumptions do these models make?
        - How will these models contribute to the delivery of evidence useful to the acceptance decision makers?
        - How will these models combine to provide sufficient evidence without excessive duplication?
        - How will the number of tests derived from models be bounded?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)

    (6) Test Basis Axiom: Testers need sources of knowledge to select things to test

    - Identify and agree the sources of knowledge required to identify what to test. Use multiple sources; compare them and cross-check.
      - What sources will be used to describe the system and identify what is to be tested?
      - What is the derivation/heritage/provenance and reliability of these sources?
      - What are the levels of authority or precedence of these sources? How can test outcomes be related to the goals and concerns of stakeholders?
      - Who will authorise the use of these sources of knowledge?
      - Who will arbitrate/resolve conflicts between these sources of knowledge?
      - Who or what will provide the knowledge covering gaps in these sources?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)

    (7) The Oracle Axiom: Testers need sources of knowledge to evaluate actual outcomes or behaviours
    - Identify and agree the sources of knowledge required to determine expected outcomes. Use multiple sources; compare them and cross-check.
      - What sources of knowledge should be used to derive expected outcomes to validate observed behaviour?
      - Do stakeholders approve the use of these sources as a test oracle?
      - What confidence can we have in these sources?
      - Do expected outcomes need to be documented prior to test execution?
      - What is the derivation/heritage/provenance and reliability of these sources?
      - What are the levels of authority or precedence of these sources?
      - In case of dispute, who or what source will be the final arbiter?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)

    (8) The Coverage Axiom: Testing needs a test coverage model or models

    - Testers need a means of assessing the thoroughness or completeness of testing with respect to the chosen test models in ways that are meaningful to stakeholders.
      - How will coverage definitions that describe the thoroughness or adequacy of testing be articulated?
      - Can these coverage definitions be used to define a quantifiable coverage measure?
      - How can the coverage measures be related to the goals and concerns of stakeholders?
      - Could these measures support estimation, planning and progress reporting?
      - How will the thoroughness/adequacy of testing be articulated to stakeholders?
      - With respect to the acceptability of the system, what interpretation(s) of these coverage measures could be made?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)

    (9) The Prioritisation Axiom: Testing needs a mechanism for ordering tests by value
    - Testers need to be able to rank tests in order of value and identify which tests are the most valuable.
      - Who is authorised to define the priorities to be used to select things to test?
      - How will the priorities be articulated?
      - What are the constraints on testing in the current context?
      - Who is authorised to impose, change or remove these constraints?
      - Who will authorise the inclusion or exclusion of tests during scoping? Who will authorise the inclusion or exclusion of tests during test design?
      - Who will authorise the inclusion or exclusion of tests during test execution?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Test Design Axioms (6)

    (10) The Fallibility Axiom: Our sources of knowledge are fallible and incomplete
    - Test Bases, Models, Oracles and Prioritisation approach are fallible because the people who define and use them are prone to human error.
      - How have the sources of knowledge required for testing been identified?
      - Who is responsible for the content in these sources?
      - Has the content of the sources been agreed unanimously?
      - Have these sources stabilised?
      - Have these sources been verified against other references or validated against the experience of contributors and other stakeholders?
      - What measures can be taken to minimise the impact of error in our sources of knowledge?
      - If sources are fallible or conflict, who or what is the final arbiter?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)
    - The Good-Enough axiom summarised the need to provide enough evidence to support decision making with confidence. The value of testing evidence is determined by how confidently stakeholders can make the required judgements.

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (11) The Confidence Axiom: The value of testing is measured by the confidence of stakeholder decision making

    - Testers must understand the relationship between test evidence and the decisions that stakeholders must make. Testing should focus on providing the evidence that stakeholders require to make decisions with confidence.
      - What evidence do stakeholders need to make decisions with confidence?
      - How will stakeholders use that evidence?
      - How are the goals of testing articulated in plans, specifications, meetings and other communications?
      - How will we ensure that evidence is delivered as early as practical to stakeholders?
      - How will we assure the accuracy and currency of the evidence produced?
      - What preferences for format, detail, frequency, precision, accuracy exist for the evidence produced?
      - How will the evidence be transmitted, acknowledged and reviewed by stakeholders?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (12) The Repeat-Test Axiom: Some repeated tests are inevitable
    - Define and agree a policy for re-testing and regression testing; make an allowance for repeat-tests in estimates and plans.
      - Under what circumstances will failed tests be re-run?
      - Under what circumstances will passed tests be re-run?
      - What criteria will be applied for the retention of tests for reuse?
      - Under what circumstances will retained tests be discarded or amended?
      - For the purpose of planning, how will the following be estimated or defined: The proportion of tests that fail? The time required for defect fixing and re-testing? The proportion of tests to be used for regression purposes?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (13) The Execution Sequencing Axiom: Run our most valuable tests first – we may not have time to run them later
    - Sequence tests to ensure the most valuable tests are run if execution time is limited or testing is stopped.
      - What criteria for selection and/or prioritisation will be applied to sequence the planned tests?
      - What criteria for selection and/or prioritisation will be applied to sequence unplanned or ad-hoc tests?
      - What are the constraints to sequencing tests?
      - How will dependencies between tests be minimised and managed?
      - Under what circumstances can tests run out of sequence?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (14) The Environment Axiom: Test execution requires a known, controlled environment
    - Establish the need and requirements for an environment and test data to be used for testing, including a mechanism for managing changes to that environment – in good time.
      - Who is responsible for the acquisition, configuration and support of test environments?
      - What assumptions regarding test environments do our test models make?
      - How will requirements for test environments be articulated, negotiated?
      - How will the validity and usability of test environments be assured?
      - How will changes to environments be managed, consistent with changes in requirements and other deliverables under test?
      - How will the state of environments, including backed up and restored versions be managed?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (15) The Event Axiom: Testing never goes as planned; evidence arrives in discrete quanta

    - Acknowledge the testing uncertainty principle, and manage stakeholder expectations. Agree a mechanism for managing and communicating events that have a bearing on the successful delivery of test evidence.
      - How does the test approach accommodate unplanned events?
      - How do plans articulate the uncertainties and expectations of test execution?
      - How will changes to the test bases, system or environment be managed?
      - How will test failures be communicated, tracked and managed?
      - How will unplanned events having an adverse effect on testing be recorded, tracked, analysed and reported?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - Delivery Axioms (6)

    (16) The Never-Finished Axiom: Testing never finishes; it stops
    - Recognise that testing is usually time limited and may not complete. Manage the expectations of testers and stakeholders accordingly.
      - Are stakeholders aware that testing probably will not complete in planned timescales?
      - Do testers appreciate that all tests might not be run, that the acceptance decision may be made on incomplete evidence and that the accepted system could be imperfect?
      - Are testers prepared to inform, advise and facilitate the acceptance decision?
      - What role will the testers have in presenting test evidence and completion (or incompletion) status?
      - How will the testers support stakeholders in judging the relevance/significance of inconclusive evidence?

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - 利益相关者公理
    - 公理1：测试需要利益相关者。
    - 公理2：测试的价值是为利益相关者提供决策依据。
    - 公理3：如果我们不约束测试的范围，我们永远无法符合利益相关者的期望。
    - 公理4：测试和验收的范围始终是妥协的结果。

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - 测试设计公理
    - 公理5：测试设计基于模型
      - 注：这一模型可能是正式的，由形式化来描述；也可能是非正式的，在测试人员的脑中。
    - 公理6：测试人员需要知识来源以选择测试内容。
    - 公理7：测试人员需要知识来源以评价被测对象的实际产出或行为。
    - 公理8：测试需要一个或多个覆盖模型。
    - 公理9：测试需要一种机制来排序测试优先级。
    - 公理10：测试的知识来源是模糊且不完整的。

- Test Axioms: Context-free rules for software testing 软件测试的公理
  - 测试执行与交付公理
    - 公理11：通过利益相关者做决策时的信心来衡量测试的价值高低。
    - 公理12：一些重复测试是不可避免的。
    - 公理13：先执行最有价值的测试—否则可能没有时间执行它们
    - 公理14：测试执行需要明确、可控的测试环境。
    - 公理15：测试永远不会按计划进展，测试所得到的证据按照离散量子化的模式出现。
    - 公理16： 测试永远不会结束，只会停止。

一些讨论

- 软件测试不能表明被测软件中不存在错误，它只能证明被测软件中存在错误。
- 软件测试的两个作用是：确定被测程序中存在缺陷；判断该程序实际是否可用。
- 软件测试最困难的问题之一是：何时停止测试？
- 自己不能测试自己的程序。
- 当一个软件被测出的缺陷数目增加时，该软件中存在更多的未被发现的缺陷的概率也随之增加。
- 一个好的测试用例应当是一个对以前未被发现的缺陷有高发现率的用例，而不是一个表明被测程序工作正确的用例。
- 测试用例要兼顾有效和无效的输入。
- 每个测试用例的必备部分是描述预期的输出。
- 测试的目标从一开始就确定。

一些讨论

- 足够好原则
  - 足够好原则是一种权衡投入/产出比的原则。
  - 不充分的测试是不负责任的；过分的测试是一种资源的浪费，同样也是一种不负责任的表现。
    - 制定最低测试通过标准和测试内容
- 木桶原理
  - 把全面质量管理 (TQM) 的概念应用到软件产品的生产。
  - 产品质量的关键因素是分析、设计和实现，测试应该是融于其中的补充检查手段，其他管理、支持、甚至文化因素也会影响最终产品的质量。
  - 测试是提高产品质量的必要条件，也是提高产品质量最直接、最快捷的手段，但决不是一种根本手段。将提高产品质量的砝码全部押在测试上是完全不可取的。

一些讨论

- 软件缺陷发现的80-20原理
  - 一般情况下，在分析、设计、实现阶段的复审和测试工作能够发现和避免80%的缺陷，而系统测试又能找出其余缺陷中的80%，最后剩余的4%的缺陷可能只有在用户的大范围、长时间使用后才会曝露出来。
  - 测试只能够保证尽可能多地发现错误，无法保证能够发现所有的错误。

软件测试的质量度量

- 目的
  - 改进软件测试质量，提高测试效率，改进测试过程的有效性。
- 难度
  - 软件产品的质量不能直接反映到软件测试的效果，只能转移到对软件测试过程的度量以及对测试产出物的度量。
- 关键
  - 关键是对软件测试人员工作的质量度量。
  - 测试人员是测试过程的核心人物，测试人员的工作质量会极大地影响测试的质量，以及产品的质量评价。
  - 对测试人员的工作评价一般由测试经理或项目经理、质量保证人员以及开发人员这三类人员进行综合考核或评判。

软件测试的质量度量

软件测试的质量指标和度量范围

指标名称	定　义	度量范围
工作量偏差	((实际工作量-计划工作量)/计划工作量)*100%	进度
测试执行率	(实际执行的测试用例数/测试用例总数)*100%	进度
测试通过率	(执行通过的测试用例数/测试用例总数)*100%	开发质量
需求覆盖率	(已设计测试用例的需求数/需求总数)*100%	测试设计质量
需求通过率	(已测试通过的需求数/需求总数)*100%	进度
测试用例命中率	(缺陷总数/测试用例总数)*100%	测试用例质量
二次故障率	(Reopen 的缺陷数/缺陷总数)*100%	开发质量
NG 率	(不能通过验证的缺陷数/缺陷总数)*100%	开发质量
缺陷有效率	(无效的缺陷数/缺陷总数)*100%	测试质量
缺陷修复率	(已解决的缺陷数/缺陷总数)*100%	开发质量
缺陷探测率	(测试者发现的缺陷/(测试者发现的缺陷+客户发现的缺陷))*100%	测试质量

软件测试的质量度量

软件测试的质量指标和度量范围

指标名称	定　义	度量范围
缺陷生存周期	缺陷从提交到关闭的平均时间	开发、测试质量
缺陷修复平均时长	缺陷从提交到修复的平均时间	开发质量
缺陷关闭平均时长	缺陷从修复到关闭的平均时间	测试质量

软件测试的衡量标准

- 多
  - 能够找到尽可能多的、以至于所有的软件缺陷
- 快
  - 能够尽可能早地发现最严重的软件缺陷
- 好
  - 找到的软件缺陷是关键的、用户最关心的
  - 找到软件缺陷后能够重现找到的缺陷，并为修正缺陷提供尽可能多的信息
- 省
  - 能够用最少的时间、人力和资源发现软件缺陷
  - 测试的过程和数据可以重用

- 国外软件测试行业的情况
  - 在一些软件终端产业比较发达的国家，软件测试已经发展成为一个独立的产业，体现在：
    - 软件测试在软件生产过程中占有重要地位，软件测试占了软件企业的 40% 以上的工作量。
    - 软件测试人员和软件开发人员之比平均在 1:1 以上
      - 微软达到 1.5:1 至 2.5:1。
    - 软件测试费用占整体项目开发费用的 30%-50%。
    - 对于高可靠、高安全的软件，测试费用相当于软件开发费用的3至5倍。
    - 软件测试理论研究蓬勃发展。
    - 软件测试市场繁荣，MI、Compuware、Rational 等机构出品的测试工具在国际市场占有领导地位。

- 国内软件测试行业的情况
  - 目前国内软件业的软件产品测试一般有下列几种形式：
    - 软件企业内部进行的功能性测试
    - 用户进行的系统验收测试
    - 第三方测试：专业软件测试人员运用一定的测试工具对软件的质量进行检测

国内软件测试行业的情况

- 整体状况
  - 重开发、轻测试导致国内软件开发企业在软件测试上的投入远远低于软件开发上的投入，更远远低于软件产业发达国家在软件测试上的投入。
  - 软件测试人员和软件开发人员之比平均在 1:3 – 1:8 之间。
  - 软件测试人才的培养水平较发达国家以及我国软件产业提升状态严重滞后，这就形成了软件测试人才的供给严重不足。
  - 缺乏对软件产品化测试技术的研究。
  - 测试服务没有形成足够的规模 。
  - 。 。 。

- 国内软件测试行业的情况
  - 行业需求
    - 软件外包中对测试环节的强化，直接导致了软件外包企业对测试人才的大量需求。
    - 几乎所有的软件企业均存在不同程度的测试人才缺口，软件测试工程师已成为了亟待补充的关键技术工种之一。

软件测试职业

- **ISTQB** ®(International Software Testing Qualifications Board) 国际软件测试认证委员会
- 软件测试职业分类
  - 软件测试工具开发工程师
    - 负责编写测试工具代码，并利用测试工具对软件进行测试。
    - 开发测试工具为软件测试工程师服务。
  - 软件测试工程师
    - 负责理解产品的功能要求，然后对其进行测试，检查软件是否存在错误，决定软件是否具有稳定性，完成相应的测试计划和测试用例。

软件测试职业

- 一般软件测试团队人员分类
  - 测试经理：主要负责人员的招聘、培训和管理，以及资源调配、测试方法改进等。
  - 实验室管理人员：设置、配置和维护实验室的测试环境，如服务器和网络环境等。
  - 测试配置管理人员：审查流程，并提出改进流程的建议；建立测试文档所需的各种模板，进行测试的配置管理，检查软件缺陷描述及其他测试报告的质量等。
  - 测试组长：测试技术业务专家，负责项目管理、测试计划制订、项目文档审查、测试用例设计和审查、任务安排，以及和项目经理、开发组长沟通等。
  - 一般 (初级) 测试工程师：编写、执行测试用例，编制有关的测试文档或开展其它相关的测试任务。

- 软件测试职业
  - 软件测试工程师分类
    - 资格分级
      - 初级测试工程师
      - 测试工程师
      - 高级测试工程师
    - 业务归口
      - 自动化测试工程师
      - 系统测试工程师
      - 架构工程师

Thank you!