第六讲 可信软件开发方法

高可信软件工程

内容安排

6.1 软件开发的历史

6.2 软件危机

6.3 可信软件开发方法

6.4 软件可靠性

6.1 软件开发的历史

- 软件
- 软件开发

  把现实世界的需求反映成软件的模型化并予以实现的过程
- 软件开发的三个阶段

  程序设计阶段（1946年-1956年）
  - 科学计算、机器语言及汇编语言、个体编程
  - 编程技巧、程序效率
  - 没有文档、软件一词尚未出现

  程序系统阶段（1956年-1968年）

  1956年，J.Backus(77) Fortran语言诞生

  大量数据处理、小组开发

  “软件”一词出现：程序及其说明

  60年代中期，软件危机。IBM OS/360 P.Brooks

  软件工程阶段（1968年以来）

  1968年NATO会议，提出“软件工程”术语

  工程化方法、描述语言、团队开发

6.1 软件开发的历史

软件的定义

①当它被执行时能够提供所要求的功能和性能的指令或计算机程序

②使得该程序能够满意地处理信息的数据结构

③描述程序的功能需求以及程序的操作和使用文档

6.2 软件危机

- 1968年，NATO会议提出了“软件危机”一词
- 软件危机包含两方面问题
  ① 如何开发软件，以满足不断增长、日趋复杂的需求；
  ② 如何维护数量不断膨胀的软件产品。
- 软件危机主要表现如下几个方面
  ① 开发成本昂贵
  ② 项目进度难控
  ③ 质量无法保证
  ④ 修改维护困难

6.2 软件危机

开发成本昂贵

- 1968年，美国花费于软件的投资高达60亿美元，有些系统，特别是军用系统，软费用要高出硬件费用好几倍，例如美国全球军事指挥控制系统的计算机硬件费用为1亿美元，而软件费用高达7.2亿美元。
- 1980年美国政府的财政年度当中，计算机系统方面(软，硬件与服务)共耗资达570亿美元，其中320亿美元(占总数的56%)用于计算机软件方面(与同年的美国汽车行业进行简单的比较，美国是当时的世界第一汽车生产大国，汽车的年销售量为900万辆，总的销售额仅为720亿美元〕。
- 技术的进步使得计算机硬件的成本持续降低，而软件成本则不断增长，软件成本在计算机系统总成本中所占的比例呈现日益扩大的趋势．来自美国空军计算机系统的数据表明，1970年，软件费用约占总费用的60%，1975年达到72%，1980年达到80%，1985年计达到85%。这种增长的速度是惊人的。（1979年，美国的国防预算为1258亿美元，其中9%用于计算机领域，约113亿美元。在这113亿美元当中，91亿美元（约占80%）用于软件投资。仅有22亿美元用于硬件设备）。

6.2 软件危机

项目进度难控

在研究大型系统时，遇到越来越多的困难。有的系绞干脆失败了，损失了大量金钱和人力；有的系统虽然完成了，但性能不理想，或推迟了许多年，经费大大超过预算。如一个大项目负责人所说：“软件人员太像皇帝新衣故事中的裁缝了、当我来检查软件开发工作时；所得到的回答好像对我说我们正忙于编织这件带有魔法的织物。只要等一会儿，你就会看到这件织物是极其美丽的。但是我什么也看不到，什么也摸不到，也说不出任何一个有关的数字；没有任何办法得到一些信息说明事情确实进行的非常顺利，而且我已经知道许多人最终已经编织了一大堆昂贵的废物而离去，还有下少人最终什么也没有作出来。”

为软件开发制定进度是根困难的事情：通常我们对一个任务根据其复杂性、工作量及进度要求安排人力。如有10人月的工作量，则由一个人完成需要10个月，由10个入完成则需要一个月。但这种工作量估计方式仅对各部分工作互下干扰的情况下才适用，例如当各部分工作尚能很好地划分时，安排由不同人完成不同部分的工作。但作为整体，尚需讨论合作，这种讨论交流活动就增加了工作量。软件系统的结构很复杂，各部分附加联系极大。增加更多人工作，往往不是缩短时间进度，而是会延缓进度。

6.2 软件危机

项目进度难控

- 对于一项复杂的任务，通常难于通过增加人力来缩短开发时间。Brook提出的法则“在已拖延的软件项目上增加入力只会使其更难按期完成”。这对于一般的工业产品来说是难于想象的!
- 1995年，美国共取消810亿美元的软件项目，其中31%未完取消，53%的项目延长一半时间，9%按期完成且不超期。1998年，美国企业应用项目不成功比率75%,其中28%的项目取消，40%无限拖长且资金超出预赛
- 对于一项复杂的任务，通常难于通过增加人力来缩短开发时间。Brook提出的法则“在已拖延的软件项目上增加入力只会使其更难按期完成”。这对于一般的工业产品来说是难于想象的!

6.2 软件危机

质量无法保证

- 1985年11月21日,由于计算机软件的错误,造成纽约银行与美联储电子结算系统收支失衡,发生了超额支付,而这个问题一直到晚上才被发现,纽约银行当日帐务出现了230亿的短款。
- Therac-25是加拿大原子能公司AECL和一家法国公司CGR联合开发的一种医疗设备，它产生的高能光束或电子流能够杀死人体毒瘤而不会伤害毒瘤附近的人体健康组织。在1985年6月到1987年1月，因为软件缺陷引发了6起由于电子流或X-光束的过量使用的医疗事故，造成4人死亡、2人重伤的严重后果。
- 美国Florida州的福利救济系统用于处理数百万受抚养儿童、食品券、医疗援助等受资助家庭接受者的资格认证，其基于巨型机系统，支持84个数据库、1390个程序、12000多个终端和个人计算机。1992年，该系统的错误使得成千上万的人收到了他们无权收到的救济，而其他成千上万急需食品券的人却排着长队等待了好几天。该错误导致了多支付2.6亿美元以及少支付5800万美元医疗补助的后果。
- 1996年，欧洲航天局阿丽亚娜5型（Ariane5）火箭在发射后40秒钟后发生爆炸，2名法国士兵当场死亡，损耗资产达10亿美元之巨，历时9年的航天计划因此严重受挫。爆炸原因在于惯性导航系统软件技术要求和设计的错误。
- 2002年，美国商务部的国立标准技术研究所（NIST）的调查报告：“据推测，由于软件缺陷而引起的损失额每年高达595亿美元。这一数字相当于美国国内生产值的0.6%”

6.2 软件危机

软件维护困难

- 软件的维护任务特别重。事实上，正式投入使用的商用软件，总是存在着一定数量的错误。随着时间的延伸，在不同的运行条件下，软件就会出现故障，就需要维护。这种维护与通常意义下的设备（硬件）维护是完全不同的。因为软件是逻辑元件，不是一种实物。软件故障是软件中的逻辑故障所造成的，不是硬件的“用旧”、“磨损”之类问题。软件维护不是更换某种备件，而是要纠正逻辑缺陷。当软件系统变得庞大，问题变得复杂时，常常会发生“纠正一个错误带来更多新的错误！”的问题。
- 新的错误发现、运行环境的改变、用户提出新要求，软件需不断修改
- 没有遵循标准、没有准确的文档，维护困难巨大。

6.2 软件危机

软件危机的原因：复杂性

软件危机的原因

- 缺乏指导或实施软件设计、开发、维护的有效理论、方法及技术 或者缺乏有效解决软件设计、开发、维护中相关实际问题的理论、方法及技术

□ 复杂性

① **规模的复杂性**
② **结构的复杂性**
③ **环境的复杂性**
④ **领域的复杂性**
⑤ **交流的复杂性**

6.2 软件危机

软件规模的复杂性

- 随着计算机应用的日益广泛，需要开发的软件规模越来越庞大。以美国宇航局的软件系统为例：1963年，水星计划的软件系统约有200万条指令；1967年，双子星座计划系统约为400万条指令；1973年，阿波罗计划系统达到1000万条指令；1979年，哥伦比亚航天飞机系统更是达到了4000万条指令。
- 软件庞大的规模是引起技术上和心理上挫折的一个重要因素；此外，规模的复杂性引起了大量学习和理解上的负担。由于在需求分析及生成规格的阶段需要搜集和分析的信息数量非常巨大，从而可能会使得信息不正确或不完整，并且在审查阶段也未能检查出来。
- 正如Leveson 所认为的：几乎所有与计算机过程控制系统有关的事故都是源于这类由软件规模因素所引起的错误。

6.2 软件危机

结构的复杂性

结构复杂性体现在管理和技术两个方面。

- 在管理方面，开发小组用来组织和管理开发活动时所采用的层次的宽度和深度，决定了用来管理系统的结构的复杂性；此外，软件开发机构内部的惯例和制度可能会改变各小组之间的信息流动，从而增加了结构复杂性。
- 在技术方面，软件系统的模块结构愈加复杂，模块之间复杂的调用关系以及接口信息往往超过了人们所能接受的程度。这种结构的复杂性可以用模块之间的耦合度来衡量，耦合度反映了在需求变化的情况下，相应所需修改的模块的数量。

6.2 软件危机

环境的复杂性

- 首先，运行中的软件总是受其所处环境的影响，在接收到外界环境的触发事件时，软件应该做出正确的响应。为了保证软件的可靠性，原则上必须对其所处环境有很好的理解，对外界环境可能产生的所有事件进行考虑，但这往往是难以办到的。
- 其次，对于许多软件系统来说，人们往往缺乏对其所运行的环境特性的认识。许多系统只有当成功地运行于其环境时，才能对其环境进行很好的理解。
- 再次，软件运行环境的多样性和异构性给软件开发者带来了更大的挑战。

6.2 软件危机

领域的复杂性

- 软件中所操作的对象仅仅是对应用领域真实对象的模拟，因而软件开发者需要从现实世界中抽象出软件模型所需的部分，并以其为基础构建软件。但是对于有的应用领域来说，这些模拟只能是近似的。其原因可能是由于对应用领域对象的认识不完全，或者是由于该模型所具有的苛刻条件限制，或者两者兼而有之。
- 对于一个应用工程来说，其中所使用的模型应该是具有合理的科学理论支持，并且经过良好测试的。然而在某些软件应用领域中，并不存在可以认知的模型，或者没有准确的模型来描绘相应的物理对象的几何、拓扑以及其它特征。在这种缺少准确认识的情况下，应用领域的某些方面很可能在软件中不能体现出来。同时，由于有的软件是根据近似的模型来构建的，因而这些模型不一定能反映事实情况。

6.2 软件危机

交流的复杂性

- 由于软件庞大的规模、大量的内部结构、以及应用领域的不同属性等因素，在开发一个大型软件系统时所参与的不仅仅是单个的人，而是一个团队。该团队里的每个人参与开发过程中的一个或多个活动。此时，对于参与不同开发阶段的人来说，彼此之间明确的交流非常重要。
- 一方面，由于结构的复杂性以及开发团队的庞大等原因，团队成员之间的交流非常困难；
- 另一方面，成员之间在进行交流时使用的媒介往往是自然语言、图、表等非形式化的方式，这些媒介很难准确地描述出所开发的产品的基本属性，并且，由于这些媒介本身所具有的二义性，往往会使开发人员产生错误的理解，这种错误将会随着开发过程的进行而逐渐蔓延开来，最终导致灾难性的后果。

6.3 可信软件开发方法

6.3.1 面向方面的软件开发

6.3.2 形式化的软件开发

6.3.1 面向方面的软件开发

可信软件是军用信息系统的重要组成部分，而软件开发框架对于软件系统的保密安全性等重要质量属性具有至关重 要的决定作用。因此，可信软件开发框架是规范可信软件开发，提高软件质量的基础。

面向方面的可信软件开发框架以面向方面技术为基础，主要解决如何规范和指导可信软件开发流程的问题，从而支持高质高效地开发可信软件。

6.3.1 面向方面的软件开发

面向方面的软件开发(AOSD，Aspect—Oriented Software Development)技术起源于面向方面编程(AOP，Aspect-Oriented Programming)技术。
AOP是一种编程方法，最早由Gregor Kiczales于1997年正式提出，并逐步应用于软件开发的多个环节，从而逐渐形成了面向方面软件开发方法。

- 提高开发效率理念
- 将某个普遍存在的功能，当作一个“方面”来统一开发，从而进行统一的管理和重用
- 语言与工具：AspectJ、Spring
- 方法：方面分析、方面设计、方面实现、方面代码织入（将“方面”代码跟普通代码整合）

6.3.1 面向方面的软件开发

利用AOP开发步骤

- 核心模块分解
- 方面分解
- 关注点的实现
- 方面重组

6.3.1 面向方面的软件开发

AOP重要概念

横切关注点

AOP把整个系统看作是不同关注点的组合,系统实现也是各个关注点的叠加过程。

“关注”是一个比“需求”涵盖范围更广的概念,它不但包括客户关注,即客户所提出的系统功能,还包括项目负责人,开发人员等所有相关者的关注,如系统性能,约束,即指对系统需求功能所提供的支持,例如系统日志事务处理等。

这些关注点表现为一种“贯穿特性”,一般是横切整个系统,或者系统中的几个模块,就是所称的横切关注点。

6.3.1 面向方面的软件开发

AOP重要概念

方面

方面是AOP所独有的数据结构,是AOP实现横切行为模块化机制的基础。

从编程的角度看,方面是AOP中用来描述和实现横切关注点的模块单元,是横切行为的封装体,定义了横切行为发生的主体、横切行为发生的条件以及横切行为本身的执行内容。

从代码实现的角度来看,方面是一个面向方面编程语言的语言结构,它将AOP中实现横切行为所需要的语法元素,如切点、通知等,都封装在同一个逻辑单元里,并且可以拥有自定义的结构成员:数据属性和方法，方面与类也有不同之处,就是方面可以有通知。

6.3.1 面向方面的软件开发

AOP重要概念

联接点和切点

联接点是AOP的胶合剂，是指在程序中能够被标识的位置,它提供一个框架能让面向方面的代码与普通代码结合起来。它是程序执行中一个精确执行点,关注点将在这里横切应用程序。从技术的角度讲,可以将程序执行中的任何可以识别的位置定义为联接点。

切点是一个或多个联接点的组合,用以捕捉程序执行中的特定联接点,并搜集该联接点上下文的程序结构。它通过与、非、或等逻辑操作符将多个联接点组织起来,并用唯一的名称进行标识。

6.3.1 面向方面的软件开发

AOP重要概念

动态横切

相对于静态横切,动态横切使用更为广泛“通知(adviCe)是AOP实现动态横切的方式,动态横切也叫做行为横切,是实现日志记录、上下文敏感的错误处理、性能的优化、应用系统的异常捕捉及处理等系统级横切关注点的常用方式。

静态横切

静态横切是提供了增加数据域和方法到一个存在的类中的途径,不仅允许在方面中定义插入到已有类中新的类成员(包括数据成员和方法),添加类的接口和构造器定义,而且还可以定义类的层次关系。

6.3.1 面向方面的软件开发

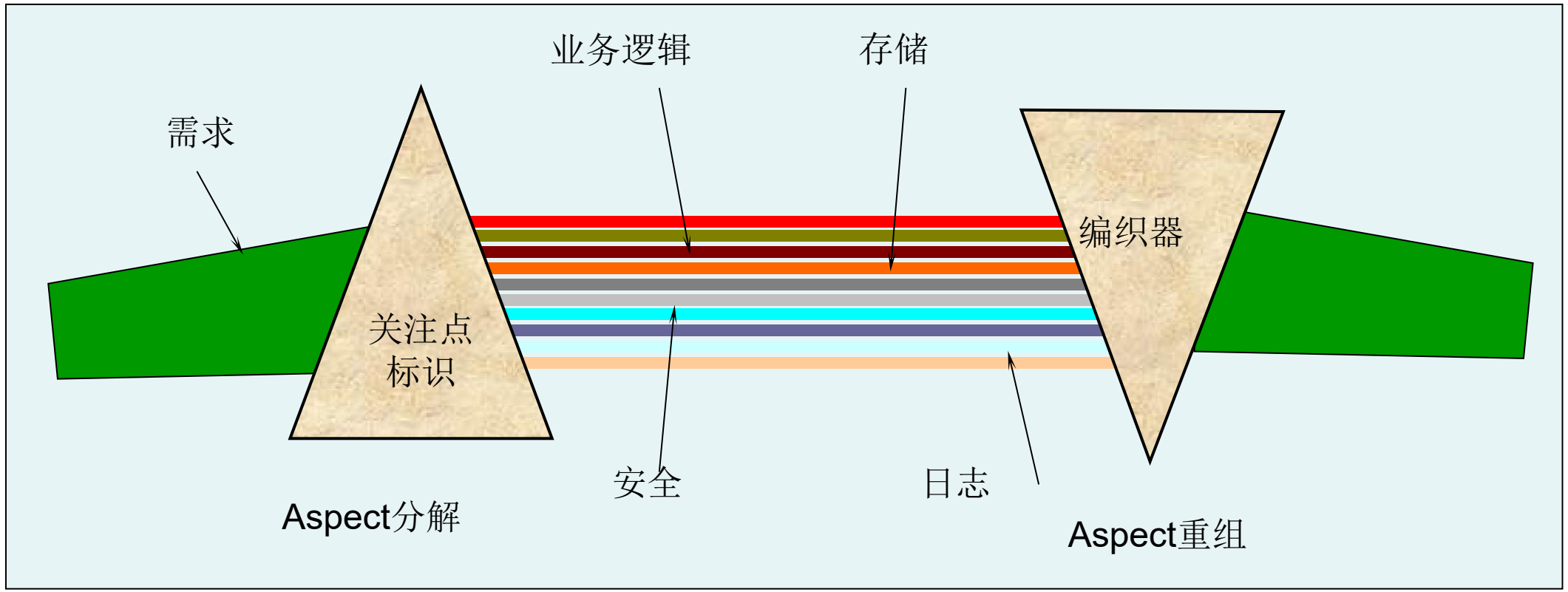
AOP重要概念

AspectJ

AspeCtJ是对现有Java语言的无缝扩展,它引入新的语言元素来支持面向方面软件开发,因此所有合法的Java程序也是合法的AspeetJ程序。AspeCtJ引入了方面(Aspect)连接点,切点,通知等基本元素。

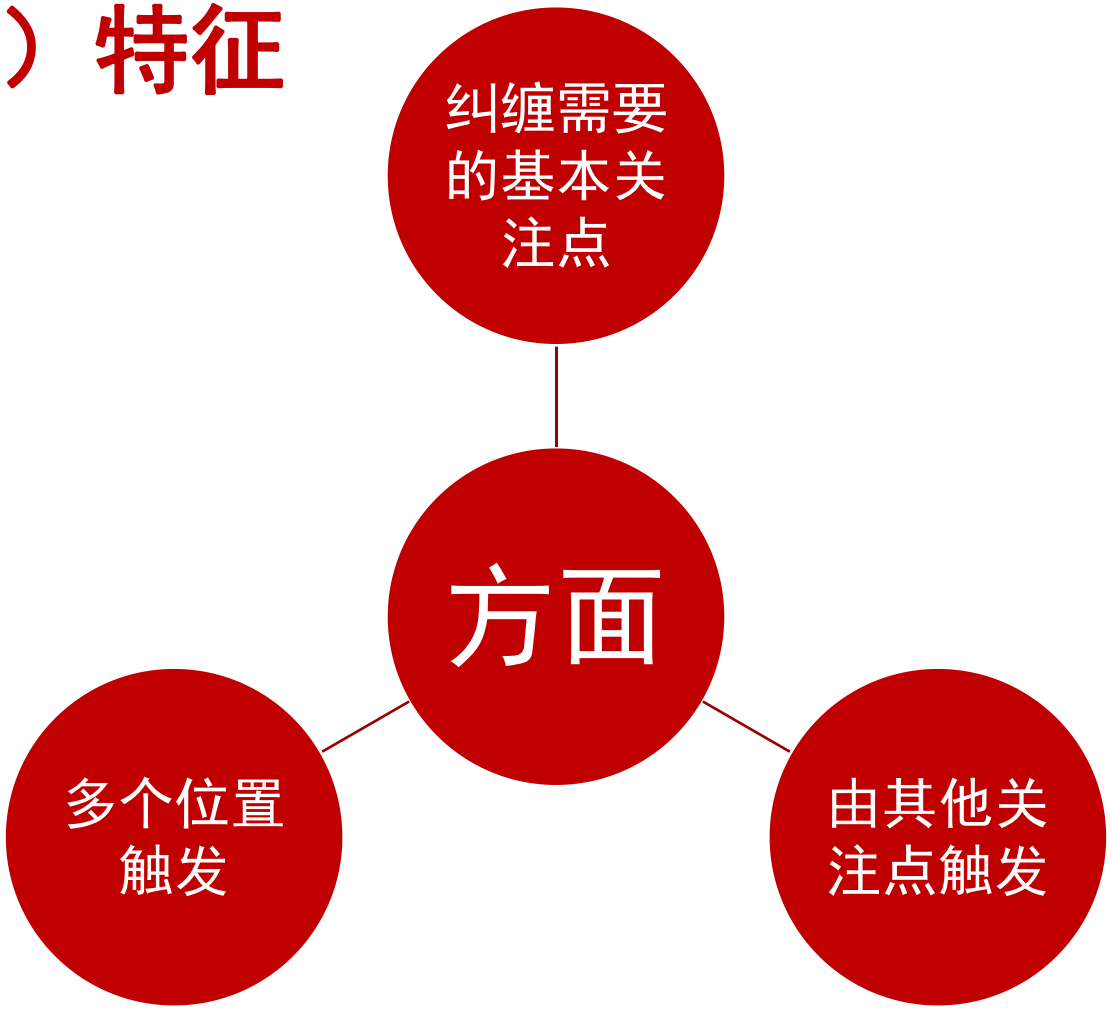
6.3.1 面向方面的软件开发

AOP系统的软件开发过程



6.3.1 面向方面的软件开发

方面（aspect）特征



方面的核心思想：
关注点的模块化

6.3.1 面向方面的软件开发

AOP系统的软件开发过程

要充分发挥AOP的优势和促进其发展，就需要一个贯穿 从需求分析、设计、实现和测试全过程的面向方面的软件开 发方法。这就是面向方面软件开发AOSD。

AOSD针对的是难于实现软件非功能性需求(包括保密安全性等)与功能性需求的相互独立，而导致软件实现中非功能性代码“散射” 或“横切”于功能性代码中。

6.3.1 面向方面的软件开发

面向方面的软件开发

AOSD的目标主要围绕着如何使整个软件系统更好地模 块化，包括功能性需求、非功能性需求等众多不同关注点的 模块化，从而保证所有关注点的相互独立，使得软件系统更 易于理解，并实现可配置、可扩展、可维护和可重用等软件工程目标。

可信软件的可信性质均为非功能性需求，因而采用AOSD技术开发可信软件具有传统的面向对象编程(OOP．Object．Oriented Programming)技术所不具备的优势。当然，AOSD并不是与现有OOP等竞争的技术，而是基予这些技术的改进和提高。

6.3.2 形式化的软件开发

形式化方法是关于在计算系统的开发中进行严格推理的理论、技术和工具,它主要包括形式化规约技术和形式化验证技术。

统计表明,传统的非形式化的软件工程技术对软件质量的保证具有一个难以逾越的顶点,而形式化方法的实践证明形式化方法是提高软件质量的重要途径。

在从高层规范至最终实现的过程中,选用适当的、以形式化方法为基础的工具进行辅助设计和验证,对提高安全攸关系统的可信度有很大帮助。

6.3.2 形式化的软件开发

- 形式化方法概念
- 发展历史
- 主要目标
- 形式化方法软件开发
  - 形式化规格
  - 形式化验证
  - 程序求精
- 形式化方法的应用与作用

6.3.2 形式化的软件开发

形式化方法概念

形式化方法是渗透在软件生命期中各个环节（如：需求、设计、实现、测试等）的数学方法或者具有严格数学基础的软件开发方法。

形式化方法的基本含义是借助数学的方法来研究计算机科学中的有关问题。

《Encyclopedia of Software Engineering》对形式化方法定义为：“用于开发计算机系统的形式化方法是基于数学的用于描述系统性质的技术。这样的形式化方法提供了一个框架，人们可以在该框架中以系统的方式刻画、开发和验证系统”。

6.3.2 形式化的软件开发

形式化方法概念

- 在软件开发的全过程中，凡是采用严格的数学语言，具有精确的数学语义的方法，都称为形式化方法。
  - 从广义角度，形式化方法是软件开发过程中分析、设计及实现的系统工程方法。
  - 狭义地，形式化方法是软件规格（specification）和验证（verification）的方法。
- 形式化方法又分为形式化规格方法和形式化验证方法。
  - 形式化规格是通过具有明确数学定义的文法和语义的方法或语言对软件的期望特性或者行为进行的精确、简洁描述。
  - 形式化验证是基于已建立的形式化规格，对软件的相关特性进行评价的数学分析和证明。

6.3.2 形式化的软件开发

发展历史

- 20世纪50年代后期，Backus（77）提出了巴克斯范式（Backus normal formula，简称BNF），作为描述程序设计语言语法的元语言；
- 20世纪60年代，Floyd（78）、Hoare（80）和Manna等开展的程序正确性证明研究推动了形式化方法的发展，他们试图用数学方法来证明程序的正确性并发展成为了各种程序验证方法，但是受程序规模限制这些方法并未达到预期的应用效果；
- 20世纪80年代，在硬件设计领域形式化方法的工业应用结果，掀起了软件形式化开发方法的学术研究和工业应用的热潮，Pnueli(96)提出了反应式系统规格和验证的时态逻辑(temporal logic，简称TL)方法，Clarke、Emerson、Sifakis(07)提出了有穷状态并发系统的模型检验(model checking)方法；
- 近十年来，建立了易于理解的规格概念和术语、形式化规格方法及语言、形式化验证技术:模型检测和定理证明，开发了支撑工具和环境

6.3.2 形式化的软件开发

形式化方法主要目标

- 形式化方法用于软件开发的主要目的是保证软件的正确性。
- 形式化方法基于严格的数学，而在软件开发过程中使用数学具有如下优点：数学是准确的建模媒体，能够对现象、对象、动作等进行简洁、准确的描述；数学支持抽象，它使得规格的本质可以被展示出来，并且还可以以一种有组织的方式来表示系统规格中的抽象层次；数学提供了高层确认的手段，可以使用数学证明来揭示规格中的矛盾性和不完整性、以及用来展示设计和规格之间的一致情况等。
- （从软件工程知识体角度）2004年5月，IEEE-CS和ACM联合任务组提交了CCSE（Computing Curriculum-Software Engineering）最终报告，在该报告给出的SEEK（Software Engineering Education Knowledge）中，“软件的形式化方法（Formal Methods in Software Engineering）”被单列为一门必修课程（序列号为SE313）。

6.3.2 形式化的软件开发

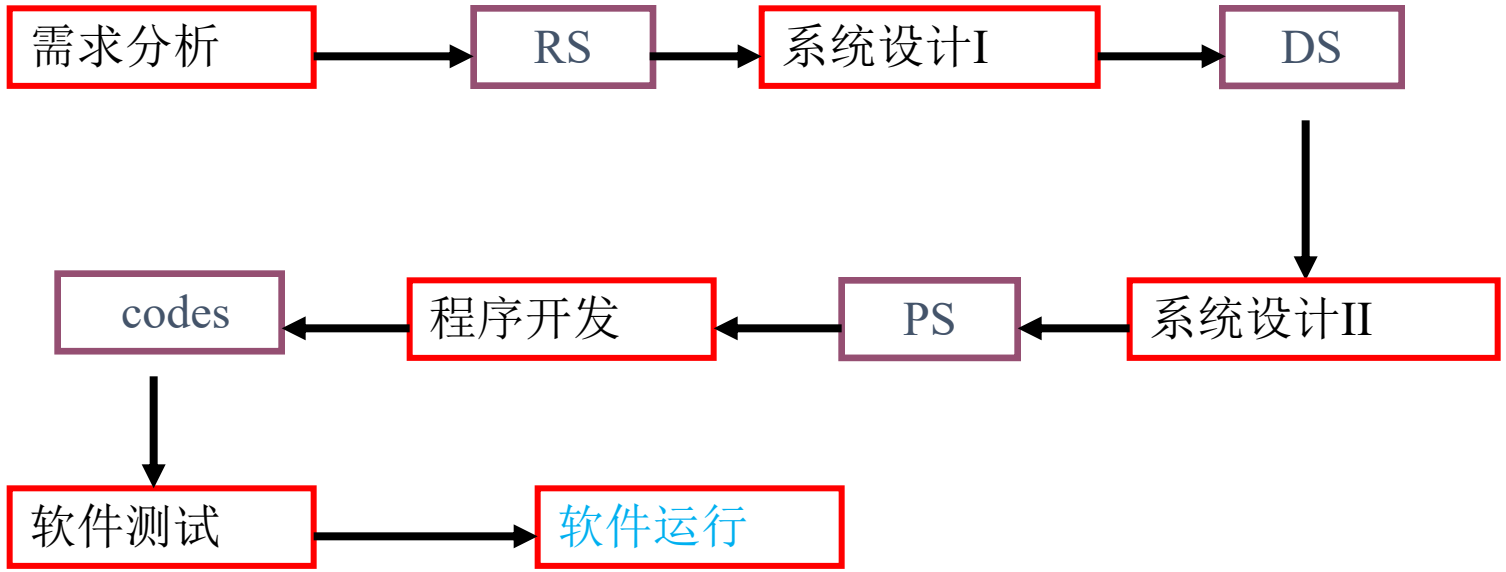
形式化方法软件开发

- 变化模型：把现实世界的需求反映成软件的模型化过程
- 模型化涉及：现实世界、模型表示、计算机系统
- 开发过程的任务分为：模型获取、模型验证、模型变换
  - 模型获取：从现实世界向模型表示转换，对应软件生命周期中的需求分析、规格、设计活动
  - 模型验证：是否满足需求及具有所期望的特性
  - 模型变换：从模型表示向计算机系统变换的过程。关键任务是一致性测试，对应软件生命周期中的实现和测试
- 这些任务对应三方面的活动：形式化规格、形式化验证、程序求精

6.3.2 形式化的软件开发

（1）形式化规格

软件规格是指对软件系统对象及用来对系统对象进行操作的方法集合，以及对所开发系统中的各对象在其生存期间里的集体行为的描述。软件生命周期模型将整个软件开发过程分解为一系列的阶段，并为每个阶段赋予明确的任务。“规格”应当理解为一个多阶段的、而不是仅仅某一个阶段的行为。



6.3.2 形式化的软件开发

（1）形式化规格

规格可以采用非形式化的方式来描述，包括自然语言、图、表等，也可以采用形式化方式来描述。

由于**非形式化方法**本身所存在的矛盾、二义性、含糊性，以及描述规格时的不完整性、抽象层次混杂等情况，使得所得到的规格不能准确地刻画系统模型，甚至会为后来的软件开发埋下出错的隐患。

而对于**形式化方法**来说，由于其基于严格的数学，具有严格的语法和语义定义，从而可以准确地描述系统模型，排除了矛盾、二义性、含糊性等情况；同时，在对系统进行严格地描述的过程中，将会帮助用户明确其原本模糊的需求，并发现用户所陈述的需求中存在的矛盾等情况，从而相对完整、正确地理解用户需求，最终得到一个完整、正确的系统模型。

6.3.2 形式化的软件开发

（1）形式化规格

形式规格精确地描述了用户的需求、软件系统的功能以及各种性质，其描述的是“做什么”，而不考虑“怎么做”。故在书写规格时应该注意的一个问题是如何描述得恰如其分，既不过多也不过少。在规格中描述过多会导致“实现偏向”，给实现施加了不必要的限制，从而排除了一些原本是合理的实现；描述得过少又有容纳不合理实现的危险。为了开发出良好的规格，除了应透彻理解、熟练掌握所使用的形式规格语言和方法外，更重要的是对所要描述的系统有全面深入的了解。

6.3.2 形式化的软件开发

（1）形式化规格

形式化规约的分类:

- 操作类方法：基于状态和转移，通过可执行模型来描述系统，模型本身能够采用静态分析和模型执行而得到验证，这类方法包括有限状态机、Statecharts、Petri网等
- 描述类方法：基于数学公理和概念，通过逻辑或代数给出系统的状态空间，是高度抽象的，便于通过自动工具进行验证。包括基于代数的Z、VDM、Larch等方法，和基于时态逻辑的PLTL、FOLTL、CTTL方法
- 双重类方法：兼有前面二者的特点，既能够通过数学公理和概念来高度抽象地描述系统，又具有状态和转移的可执行特征，这类方法包括扩展状态机/实时时态逻辑(ESM/RTTL)、TRIO+、TROL等

6.3.2 形式化的软件开发

（1）形式化规格

形式化规格实例

- 20世纪80年代，牛津大学和Hursley实验室于合作将Z用于IBM商用信息控制系统。IBM对整个开发所进行的测试表明：明显地改善了产品质量、大量地减少了错误和早期诊断错误。IBM估计使总体开发成本降低9%。这一成果获皇家技术成就奖；
- 1992年，Praxis公司交付给英国民航局的信息显示系统是伦敦机场新空中交通管理系统的部分。在系统规格阶段，采用了抽象的VDM模型。在设计阶段，抽象VDM细化为更为具体的模块化规格。项目开发的生产效率和采用非形式化技术相当、甚至更好。同时，软件质量得到了很大的提高，软件的故障率仅为0.75每千行代码，大大低于采用非形式化技术所提供的软件的故障率（约为2~20每千行代码）；
- 美国加州大学的安全关键系统研究组所开发的空中交通防碰撞系统的形式化需求规格TCASII，采用了基于Statecharts的需求状态机语言RSML，解决了开发过程中遇到的许多问题。
- 其它方面的应用：数据库：用于存储病人监护信息的HP医用仪器实时数据库系统；电子仪器：Tektronix系列谐波发生器、Schlumberger家用电度计；硬件：INMOS浮点处理器、INMOS中T9000系列的虚拟信道处理器；医疗设备：核磁共振理疗系统；核反应堆系统：核反应器安全系统、核发电系统的切换装置；保密系统：NATO控制指挥和控制系统中的保密策略模型、Multinet网关系统的数据安全传输、美国国家标准和技术院的令牌访问控制系统；电信系统：AT&T的5ESS电话交换系统、德国电信的电话业务系统；运输系统：巴黎地铁的自动火车保护系统、英国铁路信号控制、以色列机载航空电子软件。

6.3.2 形式化的软件开发

（2）形式化验证

主要技术：模型检测和定理证明

模型检测

- 模型检验是一种基于有限状态模型并检验该模型的期望特性的技术。粗略地讲，模型检验就是对模型的状态空间进行蛮力搜索，以确认该系统模型是否具有某些性质。搜索的可终止性依赖于模型的有限性。
- 模型检验主要适用于有穷状态系统，其优点是完全自动化并且验证速度快；并且，模型检验可用于系统部分规格，即对于只给出了部分规格的系统，通过搜索也可以提供关于已知部分正确性的有用信息；此外，当所检验的性质未被满足时，将终止搜索过程并给出反例，这种信息常常反映了系统设计中的细微失误，因而对于用户排错有极大的帮助。
- 模型检验方法的一个严重缺陷是“状态爆炸问题”，即随着所要检验的系统的规模增大，模型检验算法所需的时间/空间开销往往呈指数增长，因而极大限制了其实际使用范围。

6.3.2 形式化的软件开发

（2）形式化验证

模型检测工具

- 时态逻辑模型检验工具有EMC、CESAR、SMV、Mur、SPIN、UV、SVE、HyTech、Kronos等；
- 行为一致检验工具有FDR、Cospan/Formal Check等；
- 复合检验工具有HSIS、VIS、STeP、METAFrame等。

模型检测实例

- 贝尔实验室对其高级数据链路控制器在FormalCheck下进行了模型检验功能验证，6个性能进行了规格，其中5个验证无误、另外一个失败，从而进一步发现了一个影响信道流量的Bug；
- 对某楼宇抗震分布式主动结构控制系统设计进行了规格，所生成的系统模型有2.12×1019数目的状态。经过模型检验分析发现了影响主动控制效果的计时器设置错误；

6.3.2 形式化软件开发方法

（2）形式化验证

模型检测实例

- 基于SMV输入语言建立了IEEE Futurebus+896.1-1991标准下cache一致协议的精确模型，通过SMV验证了模型满足cache一致性的规格。并发现了先前并未找到的潜在协议设计错误。该工作是第一次从IEEE标准中发现错误；
- Philips公司音响设备的控制协议通过HyTech得到了完全自动验证，这是一个具有离散和连续特征的混杂系统验证问题；
- AT&T公司的7500条通信软件的SDL源代码进行了验证，从中发现112个错误，约55%的初始设计需求在逻辑上不一致。

6.3.2 形式化的软件开发

（2）形式化验证

定理证明

- 采用逻辑公式来规格系统及其性质，其中的逻辑由一个具有公理和推理规则的形式化系统给出，进行定理证明的过程就是应用这些公理或推理规则来证明系统具有某些性质。
- 不同于模型检验，定理证明可以处理无限状态空间问题。定理证明系统又可以粗略地分为自动和交互式两种类型。自动定理证明系统是通用搜索过程，在解决各种组合问题中比较成功；交互式定理证明系统需要与用户进行交互，要求用户能提供验证中创造性最强部分（建立断言等）的工作，因而其效率较低，较难用于大系统的验证。

6.3.2 形式化的软件开发

（2）形式化验证

定理证明工具

- 用户导引自动推演工具有ACL2、Eves、LP、Nqthm、Reve和RRL；
- 证明检验器有Coq、HOL、LEGO、LCF和Nuprl；复合证明器Analytica、PVS和Step；

定理证明实例

- 基于符号代数运算的自动定理证明用于证明Pentium中SRT算法的正确性，检查出了一个由故障商数字选择表引起的错误；

6.3.2 形式化的软件开发

（2）形式化验证

定理证明实例

- PowerPC和System/390中寄存器传输级、门级、晶体管级的硬件设计模拟为布尔状态转移函数，基于OBDD的算法用来检验不同设计级上状态转移函数的等价性；
- Nqthm用于Motorola 68020微处理器的规格，进而用来证明不同来源的二进制机器码的正确性；
- ACL2用于AMD5K86的浮点除微代码的规格和机械证明，ACL2还用来检验浮点方根微的正确性，发现了其中的Bug，并对修改后的微代码进行了正确性机械证明；
- ACL2用于Motorola复数算术处理器CSP的完全规格，同时对CSP的几个算法进行了验证；
- PVS用于航空电子微处理器AAMP5的规格和验证，对209条AAMP5指令中的108条进行了规格，验证了11个有代表性的微代码。

6.3.2 形式化的软件开发

（3）程序求精

程序求精，又称为程序变换，是将自动推理和形式化方法相结合而形成的一门新技术，它研究从抽象的形式规格推演出具体的面向计算机的程序代码的全过程。

程序求精的基本思想是用一个抽象程度低、过程性强的程序去代替一个抽象程度高、过程性弱的程序，并保持它们之间功能的一致。这里所说的“程序”与传统观点中“可以由计算机直接执行”的“程序”不同，**这里的“程序”是对规格、设计文档以及程序代码的统称。**在这种理解下，程序可以划分为若干层次：最高层是不能直接执行的程序，即规格，它由抽象的描述语句构成；最低层是可以直接执行的程序，称为程序代码，它由可执行的命令语句构成；最高层和最低层之间为一系列混合程序，其中既含有抽象的描述语句，又含有可执行的命令语句。

程序求精技术是形式化方法研究的一个热点，在已出现的许多相关技术中，真正能够应用到实际软件开发过程中的并不多。目前比较典型的是IBM Hursly公司以及牛津大学PRG程序设计研究组提出的针对Z规格的求精方法、以及Carroll Morgan的规则求精方案。

6.3.2 形式化的软件开发

形式化方法的应用与作用

安全关键系统的定义

安全关键系统(Safety Critical System----SCS)指对软件、硬件安全性级别要求极高的系统,这种系统的运行直接关系着人员生命和财产安全。

SCS所涉及的领域很广,目前对 & 的研究主要集中于航空、航天、军事、水利、铁路信号、交通运输、核电站、化工系统以及医疗系统等领域。

近年来随着计算机在安全关键领域的广泛使用,系统内及系统与环境间的复杂交互作用所引起的故障较其它类型的故障而言,对系统安全性的影响最大,尤其是软件系统引起的故障,称之为设计型故障。设计型故障更具隐蔽性,更不易被检测和排除。

6.3.2 形式化的软件开发

关键系统的形式化保障

目前的安全关键软件系统开发模型存在的问题和不足主要是系统验证和系统的需求规范。

目前安全关键系统的开发方法中系统验证指对系统的安全性和可信性进行确认，确认的依据源自如下两种事实

- 在相关标准中规定的开发者在系统开发和安全性评估过程中需要遵循的原则
- 工程系统自身的完善性需求。目前的安全关键系统开发模型中系统验证主要是在系统设计制造的后期，或者系统设计完成后进行的。所采用的方法主要是系统模拟、仿真和系统测试。系统模拟和仿真只能用典型的情况对系统进行考察。对系统进行穷尽的模拟是不可能的。系统的需求和规范阶段，没有很好的措施，可避免矛盾、二义性、含糊性和不完整陈述以及抽象层次的混杂等问题。

6.3.2 形式化的软件开发

形式化方法在安全关键系统中的作用

形式化方法是保证设计正确性的一条重要途径。它用数学方法表达系统的规范或性质，并且根据数学理论证明所设计的系统是否满足系统的规范或具有所期望的性质。在不能证明所期望的性质时，则可以发现设计错误。

实践证明，形式化方法确实通过形式规范和证明而增强了对系统的理解而发现了设计错误，或者通过形式化的自动验证发现了用其它方法不能发现的设计错误，因此有充分的理由说明**形式化方法是计算机系统设计验证的一条有效途径**。

尽管目前未能在软件生产实践中全面推广，但是对软件工程毕竟产生了方法学方面的重大影响。后来这些成果又在硬件设计中应用并发展，目前已取得了突破性进展。

可以认为，形式化方法的研究路线不是传统的科学发现的模式，即“实验一归纳一理论一验证一实验”的模式。而是“理论一演绎一验证一实验一佯谬一创造性思维—理论”的模式，

所以，从科学方法论的角度来说，形式化方法是以演绎法为主的科学研究方法的运用。

6.3.2 形式化的软件开发

形式化方法在软件开发中的作用

- 对软件要求的描述。软件要求的描述是软件开发的基础。
- 其次是对软件设计的描述。软件设计的描述和软件要求的描述一样重要。形式化方法的优点对于软件要求的描述同样适用于软件设计的描述。另外由于有了软件要求的形式化描述，可以检验软件的设计是否满足软件的要求。
- 形式化方法可以用于程序的验证，以保证程序的正确性。对于测试来讲，形式化方法可用于测试用例的自动生成，这可以节约许多时间和在一定程度上保证测试用例的覆盖率

随着软件的广泛应用，特别是软件在尖端领域的应用，软件可靠性成为一个非常重要的问题。**软件的可靠取决于两个方面，一个是软件产品的测试与验证，另一个是软件开发的方法与过程。**对简单的软件开发，应该是先有对软件的需求，然后对软件进行设计，然后是编写程序，最后是对程序进行测试。

6.4 软件可靠性

6.4.1 软件可靠性

6.4.2 软件可靠性三要素

6.4.3 可靠性分析方法

6.4.1 软件可靠性

软件可靠性是软件系统固有特性之一，它表明了一个软件系统按照用户的要求和设计的目标，执行其功能的正确程度。软件可靠性与软件缺陷有关，也与系统输入和系统使用有关。理论上说，可靠的软件系统应该是正确、完整、一致和健壮的。

但是实际上任何软件都不可能达到百分之百的正确，而且也无法精确度量。

一般情况下，只能通过对软件系统进行测试来度量其可靠性。软件可靠性给出如下定义：

“软件可靠性是软件系统在规定的时间内及规定的环境条件下，完成规定功能的能力”。

6.4.2 软件可靠性三要素

- 规定的时间

  软件可靠性只是体现在其运行阶段，所以将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起的累计时间

- 规定的环境条件

  环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素

- 规定的功能

  软件可靠性还与规定的任务和功能有关。由于要完成的任务不同，软件的运行剖面会有所区别，则调用的子模块就不同，其可靠性也就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能

6.4.3 可靠性分析方法

1. 故障模式影响与危害性分析法（FMECA）
2. 故障树分析法（FTA）

6.4.3 可靠性分析方法

FMECA

- 概述
- FMECA的定义、目的和作用
- FMECA的方法
- FMECA的步骤
- 系统定义
- 故障模式影响分析
- 危害性分析
- FMECA结果输出

概述

元部件的故障对系统可造成重大影响

- 灾难性的影响

  挑战者升空爆炸——发动机液体燃料管垫圈不密封

- 致命性的影响

  起落架上位锁打不开

以往设计师依靠经验判断元部件故障对系统的影响

- 依赖于人的知识和工作经验

系统的、全面的和标准化的方法—— FMECA

- FMECA的发展
- 设计阶段发现对系统造成重大影响的元部件故障
- 设计更改、可靠性补偿

是可靠性、维修性、保障性和安全性设计分析的基础

FMECA的概念

FMECA的定义

故障模式影响及危害性分析(Failure Mode ,Effects and Criticality Analysis , 记为FMECA)是分析系统中每一产品所有可能产生的故障模式及其对系统造成的所有可能影响，并按每一个故障模式的严重程度及其发生概率予以分类的一种归纳分析方法。

- FMECA是一种自下而上的归纳分析方法；
- FMEA和CA。

FMECA的概念

FMECA的目的

从产品设计（功能设计、硬件设计、软件设计）、生产（生产可行性分析、工艺设计、生产设备设计与使用）和产品使用角度发现各种影响产品可靠性的缺陷和薄弱环节，为提高产品的质量和可靠性水平提供改进依据。

FMECA的概念

FMECA的作用

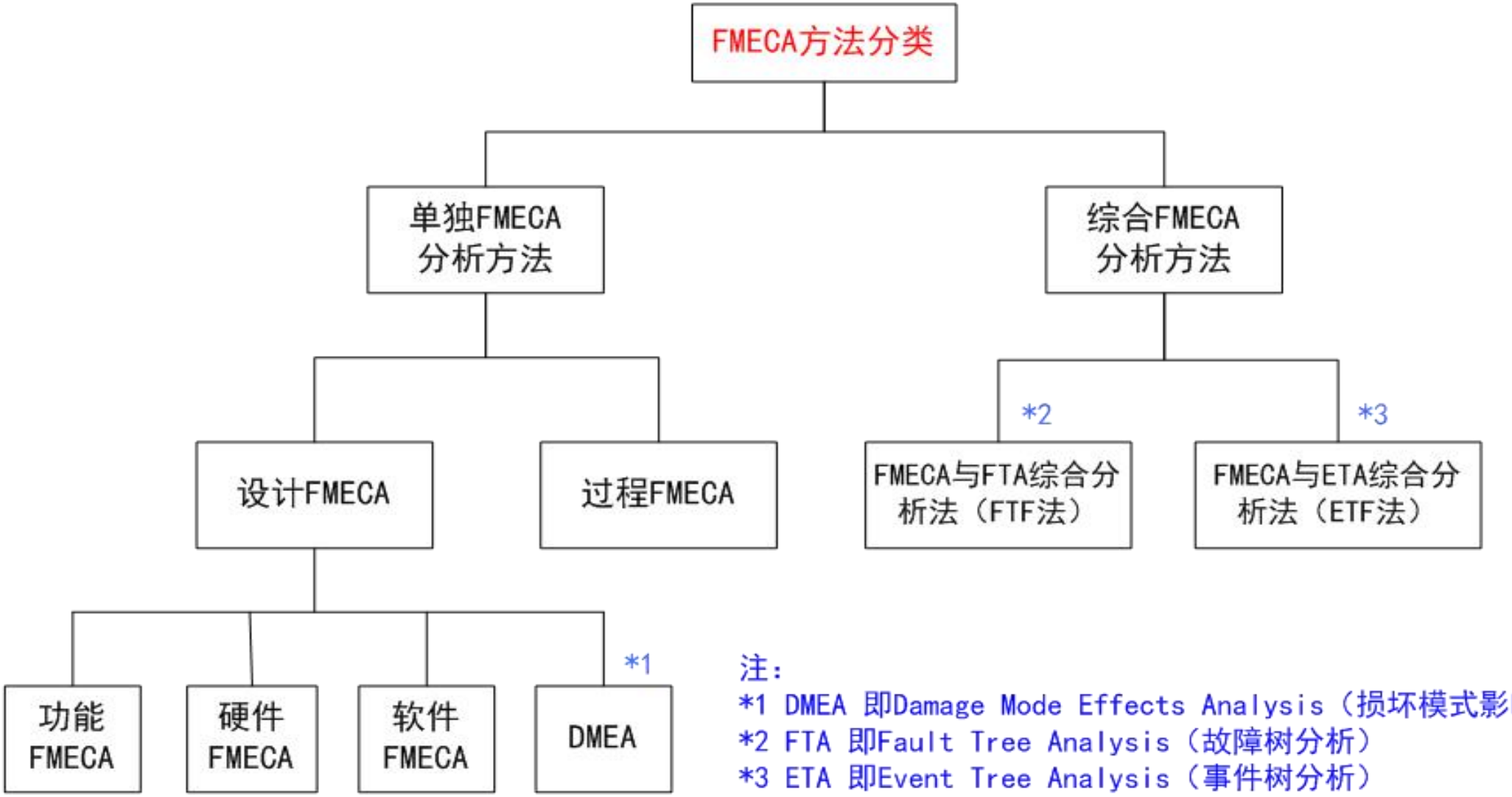
保证有组织地定性找出系统的所有可能的故障模式及其影响，进而采取相应的措施。

为制定关键项目和单点故障等清单或可靠性控制计划提供定性依据；为制定试验大纲提供定性信息；为确定更换有寿件、元器件清单提供使用可靠性设计的定性信息；为确定需要重点控制质量及工艺的薄弱环节清单提供定性信息。

可及早发现设计、工艺中的各种缺陷。

为可靠性（R）、维修性（M）、安全性（S）、测试性（T）和保障性（S）工作提供一种定性依据。

FMECA方法分类

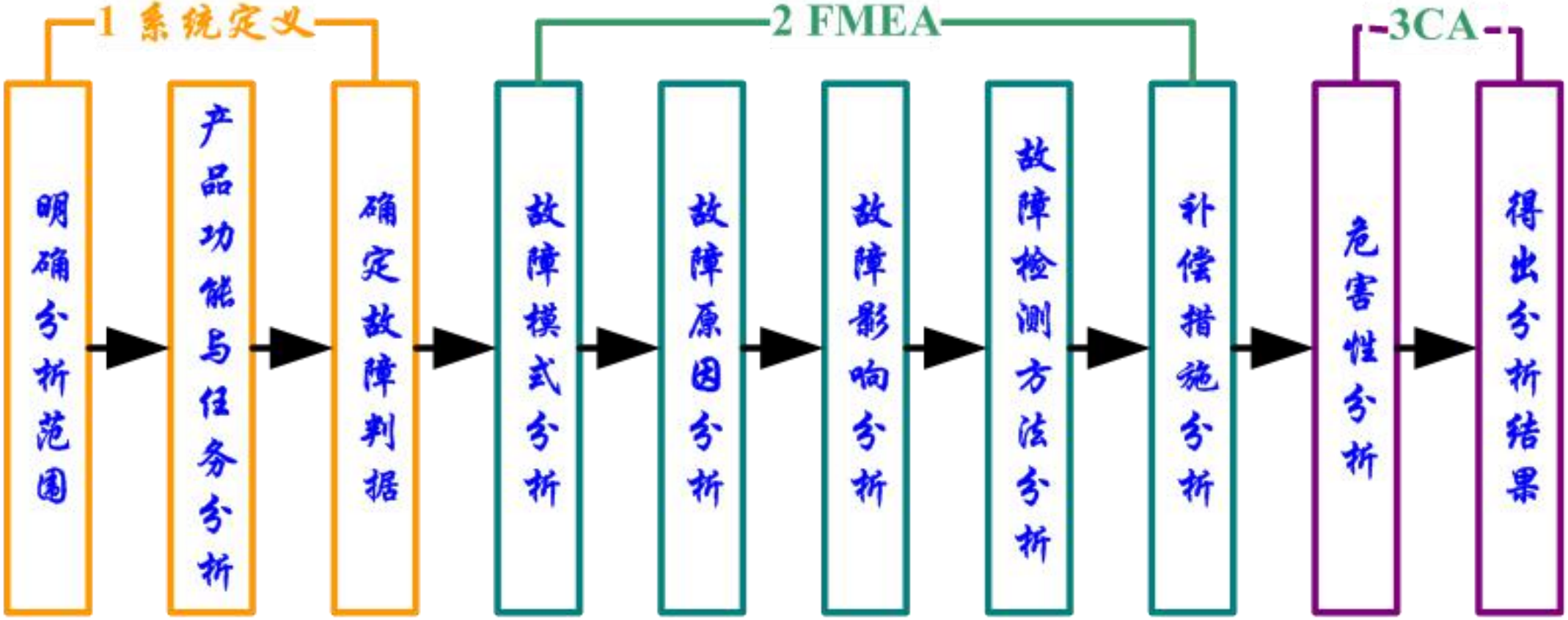


注：
*1 DMEA 即Damage Mode Effects Analysis（损坏模式影响分析）
*2 FTA 即Fault Tree Analysis（故障树分析）
*3 ETA 即Event Tree Analysis（事件树分析）

产品寿命周期各阶段的FMECA方法

	论证与方案阶段	工程研制阶段	生产阶段	使用阶段
方法	功能FMECA	·硬件FMECA ·软件FMECA ·损坏模式影响分析	过程FMECA	统计FMECA
目的	分析研究系统功能设计的缺陷与薄弱环节，为系统功能设计的改进和方案的权衡提供依据。	分析研究系统硬件、软件设计的缺陷与薄弱环节，为系统的硬件、软件设计改进和保障性分析提供依据。	分析研究所设计的生产工艺过程的缺陷和薄弱环节及其对产品的影响，为生产工艺的设计改进提供依据。	分析研究产品使用过程中实际发生的故障、原因及其影响，为提供产品使用可靠性和进行产品的改进、改型或新产品的研制提供依据。

FMECA的步骤


1 系统定义
2 FMEA
3CA
明确分析范围
产品功能与任务分析
确定故障判据
故障模式分析
故障原因分析
故障影响分析
故障检测方法分析
补偿措施分析
危害性分析
得出分析结果

系统定义

- 明确分析范围
- 根据系统的复杂度、重要程度、技术成熟性、分析工作的进度和费用约束等，确定系统中进行FMECA的产品范围
  - 产品层次示例
  - 约定层次——规定的FMECA的产品层次
  - 初始约定层次——系统最顶层
  - 最低约定层次——系统最底层

系统定义

- 系统任务分析和功能分析
  - 描述系统的任务要求及系统在完成各种功能任务时所处的环境条件
    - 任务剖面、任务阶段
  - 分析明确系统中的产品在完成不同的任务时所应具备的功能、工作方式及工作时间等
    - 功能描述
- 确定故障判据
  - 制定系统及产品的故障判据。选择FMECA方法等
    - 故障判据
    - 分析方法

故障模式影响分析FMEA

FMEA的工作内容

- 故障模式分析
  找出系统中每一产品所有可能出现的故障模式。
- 故障原因分析
  找出每一个故障模式产生的原因。
- 故障影响分析
  找出系统中每一产品的每一个可能的故障模式所产生的影响，并按这些影响的严重程度进行分类。
- 故障检测方法分析
  分析每一种故障模式是否存在特定的发现该故障模式的检测方法，从而为系统的故障检测与隔离设计提供依据。
- 补偿措施分析
  针对故障影响严重的故障模式，提出设计改进和使用补偿的措施。

危害性分析(CA)

目的是按每一故障模式的严重程度及该故障模式发生的概率所产生的综合影响对系统中的产品划等分类，以便全面评价系统中可能出现的产品故障的影响。CA是FMEA的补充或扩展，只有在进行FMEA的基础上才能进行CA。

常用方法

- 风险优先数（Risk Priority Number, RPN）法
  - 主要用于汽车等民用工业领域
- 危害性矩阵法
  - 主要用于航空、航天等军用领域

危害性分析(CA)

风险优先数法

$$RPN=OPR\times ESR\times DDR$$

OPR（Occurrence Probability Ranking）—— 故障模式发生概率等级
ESR（Effect Severity Ranking）—— 影响严酷度等级
DDR（Detection Diffculty Ranking）—— 检测难度等级

上述三项因素通过评分获得。因此，首先应给出各项因素的评分准则。

危害性分析(CA)

发生概率等级OPR

用于评定某一特定的故障原因导致的某故障模式实际发生的可能性。

等级	故障发生的可能性		参考值
1	稀少	故障模式发生的可能性极低	$1/10^6$
2 3	低	故障模式发生的可能性相对较低	1/20000 1/4000
4 5 6	中等	故障模式发生的可能性中等	1/1000 1/400 1/80
7 8	高	故障模式发生的可能性高	1/40 1/20
9 10	非常高	故障模式发生的可能性非常高	1/8 1/2

危害性分析(CA)

严重度等级ESR

用于评定所分析的故障模式的最终影响。

等级	故障影响的严重程度	
1	轻微	对系统的性能不会产生影响，用户注意不到的轻微故障
2,3	低	对系统性能有轻微影响的故障，用户可能会注意到并引起轻微抱怨
4,5,6	中等	引起系统性能下降的故障，用户感觉不舒适和不满意
7,8	高	中断操作的重大故障或提供舒适性的子系统不能工作的故障，用户感到强烈不满意。但此类故障不会引起安全性后果也不违反政府法规
9,10	非常高	引起生命、财产损失的致命故障或不符合政府法规的故障

危害性分析(CA)

检测难度等级DDR

用于评定通过企业内部预定的检验程序查出引起所分析的故障模式的各种原因的可能性。

等级	检验程序查出故障的难度	
1,2	非常低	检验程序可以检出的潜在设计缺陷
3,4	低	检验程序有较大机会检出的潜在设计缺陷
5,6	中等	检验程序可能检出的潜在设计缺陷
7,8	高	检验程序不大可能检出的潜在设计缺陷
9	非常高	检验程序不可能检出的潜在设计缺陷
10	无法检出	检验程序绝不可能检出的潜在设计缺陷

危害性分析(CA)

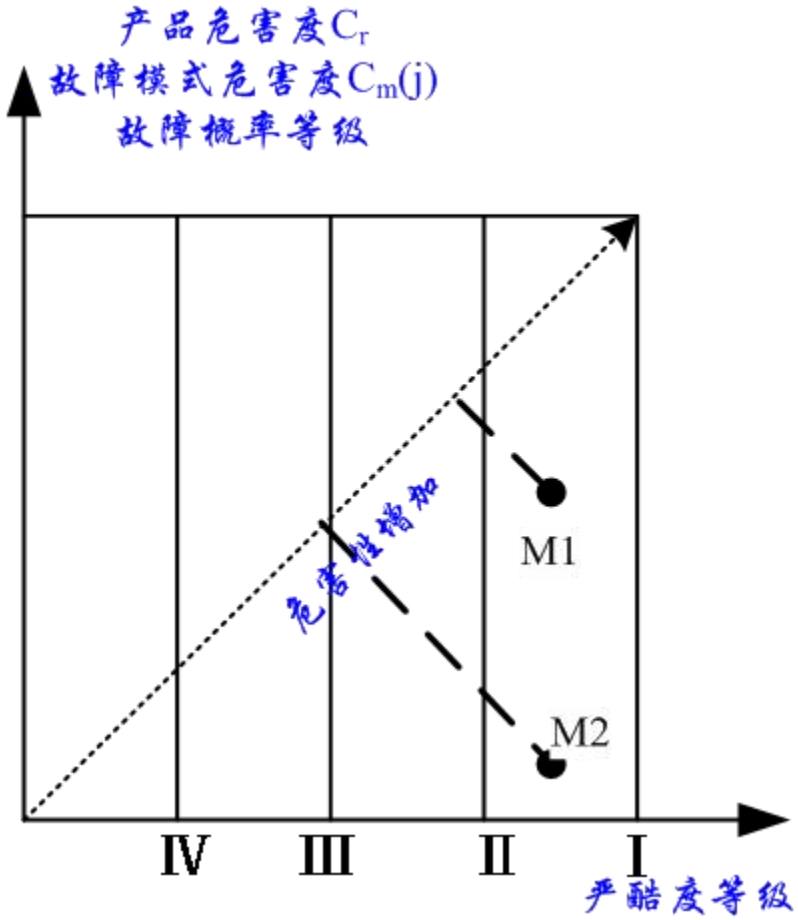
- 危害性矩阵法

分类：定性和定量

危害性矩阵图

绘制危害性矩阵图的目的是比较每个故障模式的危害程度，进而为确定改进措施的先后顺序提供依据。危害性矩阵是在某一特定严酷度级别下，产品各个故障模式危害程度或产品危害度相对结果的比较。

与RPN一样具有指明风险优先顺序的作用。



FMECA结果输出

FMECA输出

- 单点故障模式清单
- Ⅰ、Ⅱ类故障模式清单
- 可靠性关键件、重要件
- 不可检测故障模式清单
- 危害性矩阵图等
- FMEA/CA表

6.4.3 可靠性分析方法

FTA

- 故障树的基本概念
- FTA工作要求
- 常用事件、逻辑门符号
- 故障树分析
- 危险分析
- 构造故障树
- 故障树分析方法
- 其他方法

FTA基本概念

故障树定义

故障树指用以表明产品哪些组成部分的故障或外界事件或它们的组合将导致产品发生一种给定故障的逻辑图。

故障树是一种逻辑因果关系图，构图的元素是事件和逻辑门

- 事件用来描述系统和元、部件故障的状态
- 逻辑门把事件联系起来，表示事件之间的逻辑关系

故障树分析（FTA）

通过对可能造成产品故障的硬件、软件、环境、人为因素进行分析，画出故障树，从而确定产品故障原因的各种可能组合方式和(或)其发生概率。

- 定性分析
- 定量分析

FTA基本概念

FTA目的

帮助判明可能发生的故障模式和原因，发现可靠性和安全性薄弱环节，采取改进措施，以提高产品可靠性和安全性；

计算故障发生概率；

发生重大故障或事故后，FTA是故障调查的一种有效手段，可以系统而全面地分析事故原因，为故障“归零”提供支持；

指导故障诊断、改进使用和维修方案等。

FTA特点

是一种自上而下的图形演绎方法；

有很大的灵活性；

综合性：硬件、软件、环境、人为因素等；

主要用于安全性分析。

FTA工作要求

在产品研制早期就应进行FTA，以便早发现问题并进行改进。

随设计工作进展，FTA应不断补充、修改、完善。

“谁设计，谁分析”

故障树应由设计人员在FMEA基础上建立。可靠性专业人员协助、指导，并由有关人员审查，以保证故障树逻辑关系的正确性。

应与FMEA工作相结合

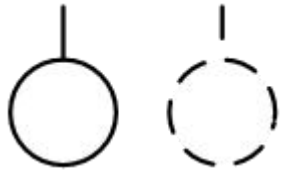
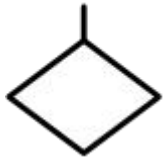
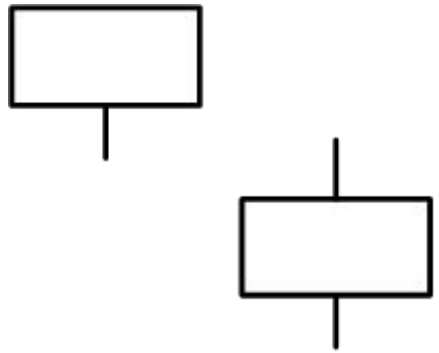
应通过FMEA找出影响安全及任务成功的关键故障模式（即I、II类严酷度的故障模式）作为顶事件，建立故障树进行多因素分析，找出各种故障模式组合，为改进设计提供依据。

FTA输出的设计改进措施，必须落实到图纸和有关技术文件中

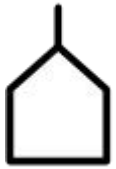
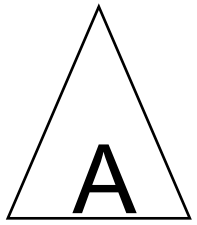
应采用计算机辅助进行FTA

由于故障树定性、定量分析工作量十分庞大，因此建立故障树后，应采用计算机辅助进行分析，以提高其精度和效率。

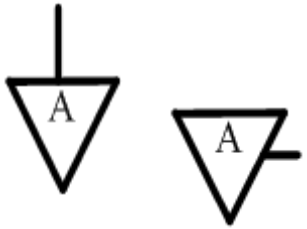
故障树常用事件符号

<table>
<tr><th colspan="2">符号</th><th colspan="2">说明</th></tr>
<tr><td rowspan="2">底事件</td><td>  </td><td colspan="2">元、部件在设计的运行条件下发生的随机故障事件。
◆实线圆——硬件故障
◆虚线圆——人为故障</td></tr>
<tr><td>  </td><td colspan="2">未探明事件
表示该事件可能发生，但是概率较小，勿需再进一步分析的故障事件，在故障树定性、定量分析中一般可以忽略不计。</td></tr>
<tr><td rowspan="2">  </td><td>顶事件</td><td>人们不希望发生的显著影响系统技术性能、经济性、可靠性和安全性的故障事件。顶事件可由FMECA分析确定。</td></tr>
<tr><td>中间事件</td><td>故障树中除底事件及顶事件之外的所有事件。</td></tr>
</table>

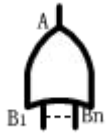
故障树常用事件符号

符号	说明
	开关事件：已经发生或必将要发生的特殊事件。
	条件事件：描述逻辑门起作用的具体限制的特殊事件。
	✓入三角形：位于故障树的底部，表示树的A部分分支在另外地方。 ✓出三角形：位于故障树的顶部，表示树A是在另外部分绘制的一棵故障树的子树。

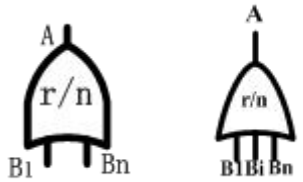
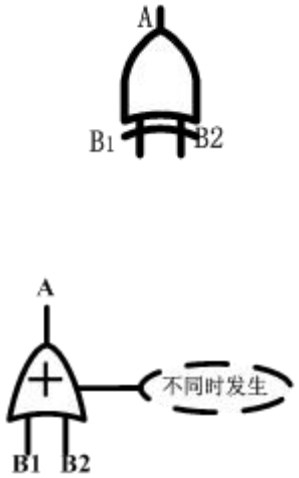
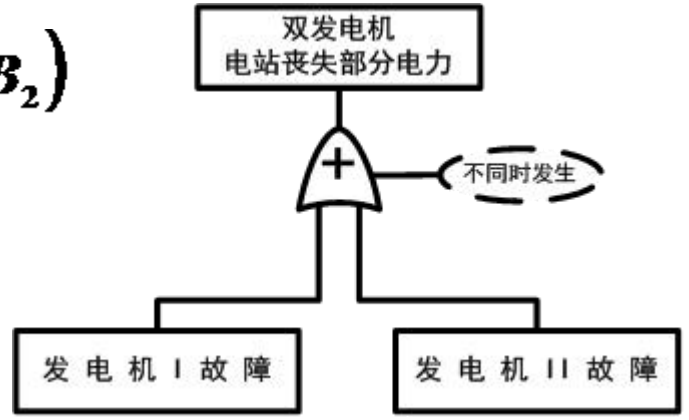
故障树常用逻辑门符号

符号	说明
	**相同转移符号（A是子树代号，用字母数字表示）：** 左图表示“下面转到以字母数字为代号所指的地方去” 右图表示“由具有相同字母数字的符号处转移到这里来”
	**相似转移符号（A同上）：** 左图表示“下面转到以字母数字为代号所指结构相似而事件标号不同的子树去”，不同事件标号在三角形旁注明 右图表示“相似转移符号所指子树与此处子树相似但事件标号不同”

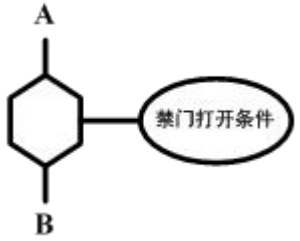
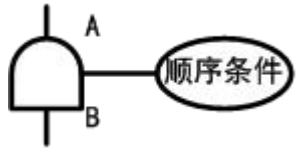
故障树常用逻辑门符号

符号	说明
与门   	✓$B_i(i=1,2,\ldots,n)$为门的输入事件，A为门的输出事件 ✓B_i同时发生时，A必然发生，这种逻辑关系称为事件交。 ✓用逻辑“与门”描述
或门   	✓当输入事件中至少有一个发生时，输出事件A发生，称为事件并。 ✓用逻辑“或门”描述

故障树常用逻辑门符号

符号	说明
	表决门：n个输入中至少有r个发生，则输出事件发生；否则输出事件不发生。
	异或门：输入事件B1，B2中任何一个发生都可引起输出事件A发生，但B1，B2不能同时发生。相应的逻辑代数表达式为 $A=(B_1\cap\overline{B}_2)\cup(\overline{B}_1\cap B_2)$ 

故障树常用逻辑门符号

符号	说明
	**禁止门：** 仅当“禁门打开条件”发生时，输入事件B发生才导致输出事件A发生； 打开条件写入椭圆框内。
	**顺序与门：仅当输入事件B按规定的“顺序条件”发生时，输出事件A才发生。**
	**非门：输出事件A是输入事件B的逆事件。**

故障树分析

建树步骤

广泛收集并分析系统及其故障的有关资料→选择顶事件→建造故障树→简化故障树。

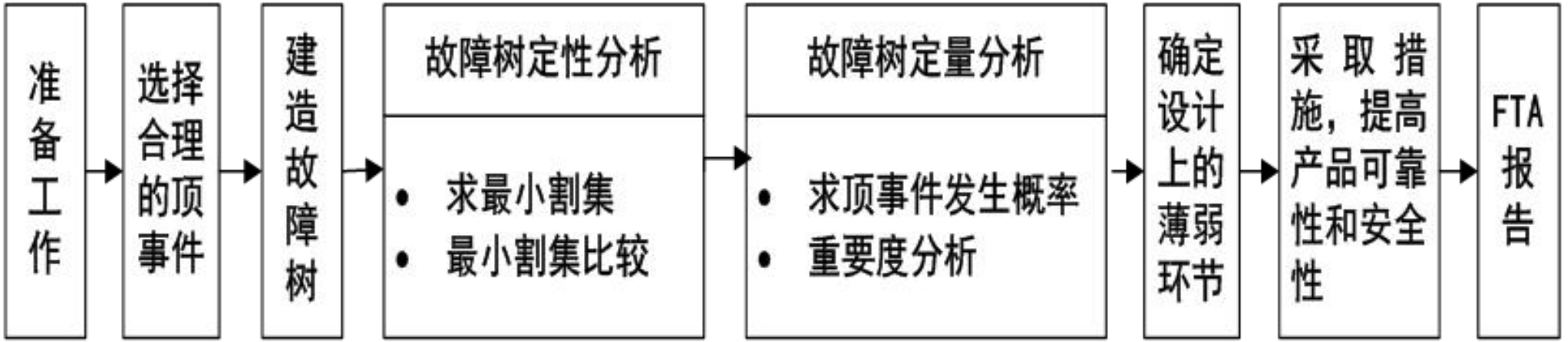
注意事项

明确建树边界条件，简化故障树；

故障事件应严格定义。

分析步骤

建立故障树→ 故障树定性分析→故障树定量分析→重要度分析→分析结论：薄弱环节→确定改进措施

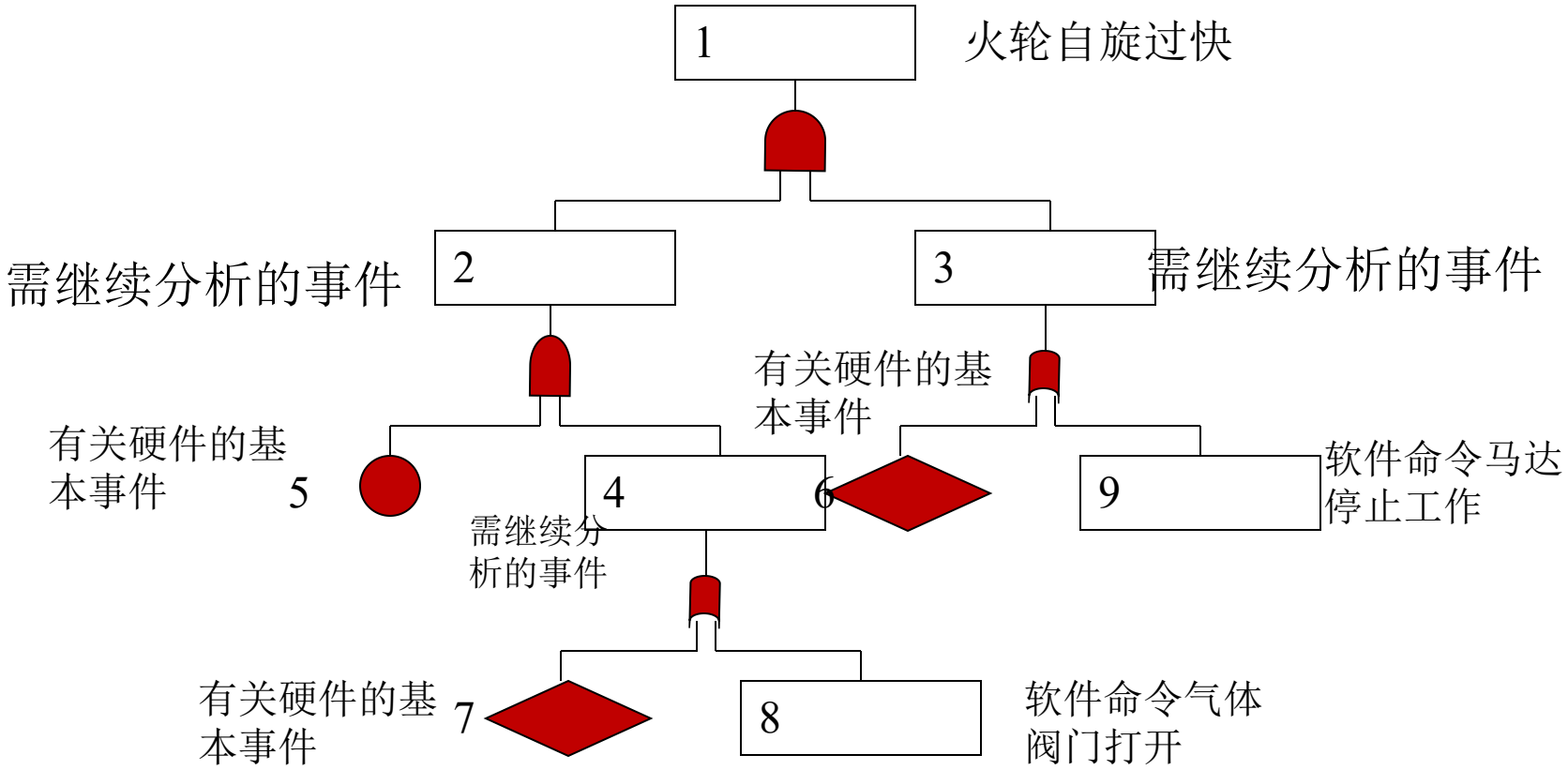


危险分析

- 危险分析的目的是确定系统可能出现的各种不安全状态，从安全的角度确定哪些失效是可以允许的，确定哪些失效是不允许的，确定在危及安全的失效发生或可能发生时，如何使系统处于失效—安全状态
- 在对系统的软件进行危险分析时，不仅应考虑软件的控制失误，而且必须研究软件和系统其它单元的接口
- 对系统的软件进行危险分析时，还必须把操作者的因素考虑在内，要仔细地考虑控制系统中的人机接口

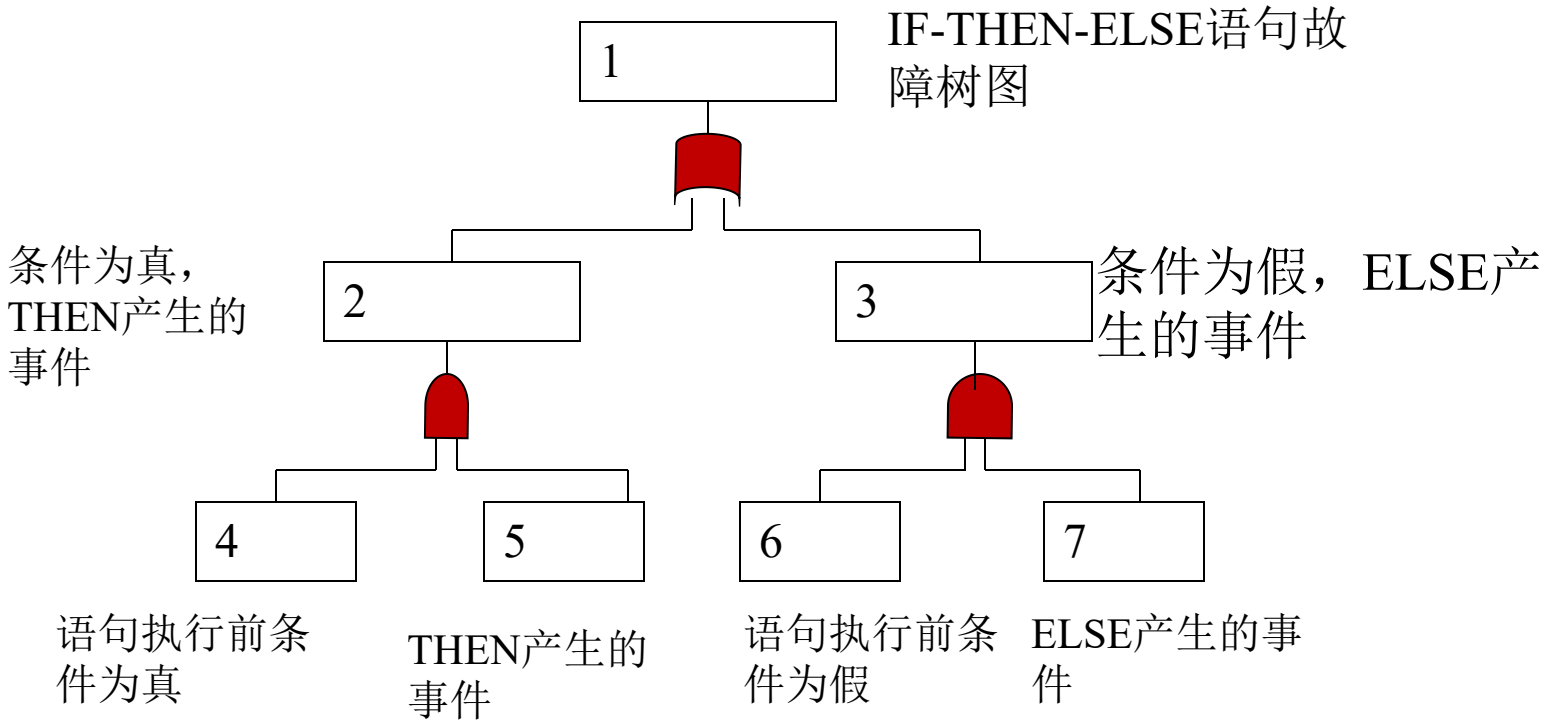
构造故障树

- 与硬件故障树的构造方法一样



构造故障树

- IF-THEN-ELSE语句故障数图



故障树分析方法

- 可以计算每一个基本事件发生的概率是已知的，顶端事件的发生概率就可以计算出来的
- 对故障进行灵敏度分析，找出顶端事件影响最大，最灵敏的基本事件。
- 找出关键性的安全事故发生的原因，可以用于指导软件测试，确定测试的重点和内容，使系统的安全得到更充分的保证
- 局限性在于工作比较繁琐

其他方法

软件潜藏分析法

- 硬件潜藏回路是电子设备中潜藏着的非期望的电流及信号通路，它们的存在可能导致意外事故的发生，或妨碍预期功能的实现，甚至造成灾难性后果
- 它产生的原因在于设计系统时的疏忽，或者由于目前系统过于复杂

存在潜藏回路的设备，在一般情况下没有任何异常迹象，设备不但能够通过常规的验收程序而交付使用，甚至可能长时间的工作，但是一旦时机适宜，就会发生。

谢谢大家！