洁净室软件工程

H. D. Mills M. Dyer R. C. Linger

众所周知，软件工程的主要目的是提高软件的开发效率和软件质量。近年来发展起来的洁净室软件工程（cleanroom software engineering）提出了用统计的质量控制方法管理软件开发过程，引起了人们的普遍关注。已有的实践经验表明，采用这种方法不仅可以明显缩短软件的开发周期，降低软件的开发费用，还可以使软件有较高的质量，特别是有较高的可靠性。

本文作者之一是IBM公司的H. D. Mills。他在70年代初期，对结构化程序设计思想的发展和普及起过重大作用。在软件开发管理方法方面提出了著名的"主程序员组"的方法。他的"洁净室软件工程"的思想，是他的主程序员组思想的进一步发展，将对软件工程的发展产生重大影响。

最近的经验表明，软件生产可以在统计式质量控制下进行，并能对提交的软件提供有关可信的可靠性统计数字。IBM的洁净室开发过程[1]揭示了软件的数学验证与软件的统计测试之间存在的令人惊奇的协同作用，也揭示了人在数学上所犯错误同人在排错时所犯错误之间的主要差别。

利用洁净室开发过程，可以在统计式质量控制下生产软件。如同洁净室硬件开发一样，开发过程中优先考虑的是预防错误发生，而不是排除错误（当然，未能预防的错误应该排除）。这种优先体现在系统测试之前，先用人进行数学验证并以此代替程序排错。

下一个优先考虑的是在系统一级上通过典型用户测试为软件质量提供有效的统计认可。质量度量是采用在一定的时间（实时或处理机时间）内所提交产品的平均无故障时间。这种认可考虑了产品在提交前的系统测试过程中可靠性的增长。

为了获得开发过程中质量控制的效益，洁净室软件工程要求这样一种开发周期：在增量式开发中产品的制作和认可齐头并进。这使得制作过程能够根据早期的认可结果进行更改，以达到预期的质量要求。

洁净室工程的实践

我们采用洁净室工程的三个典型实践项目是：IBM语言产品（40000行代码），美国空军委托开发的直升飞机飞行程序（35000行代码），和NASA委托开发的空间运输规划系统（45000行代码）。在完成这些项目的过程中我们的最大发现是：由人进行的验证，尽管也可能出错，但在软件开发中可以用来代替排错。甚至人的非形式验证也能使软件具有足够的健壮性，而不需事先排错，就可进行系统测试。

典型的程序增量是5000到15000行代码。随着经验的不断丰富和自信心的增强，预料增量可能明显加大。所有这三个项目都表明，这种方法的生产率高于或等于通常的软件开发过程，因为由人进行验证并不比排错需要更多的时间，尽管这种验证在整个开发周期中进行得比较早。

形式化设计方法和以数学为基础的验证相结合，能产生积极的开发效果：产品中占全部90%以上的错误能在首次执行前发现，而传统的方法在首次执行前只能发现60%的错误。这种效果可能直接与更加留心和注意设计，而不急于编码和依靠测试来弥

得产品质量有关。

另一个令人鼓舞的趋势是能降低错误总数达一半之多，这主要是由于洁净室开发过程注重错误的预防而不是错误的检测。软件产业的错误率平均值是每千行代码有50到60个错误，这个值减少一半将是非常可观的。

IBM语言产品(Cobol/SF[1])的经验特别有教益。这一高级技术产品的复杂性不亚于编译程序，我们先用形式化方式说明后，再用进程设计语言进行设计。说明书与设计书的长短之比是四比一。设计书中每一个控制结构都用形式化的、以数学为基础的小组检查法进行验证，所以产品非常健壮。在开发的第一阶段（20000行），在测试中只发现了53个错误。

正确性验证是这些项目的基础，为了使验证更为简单，许多程序都进行了重新设计。生产率达到平均每人月生产400多行代码，这主要是因为与传统的开发方式相比它大幅度地减少了测试时间。

在马里兰大学的一项可控实验中，学生们开发了一个消息处理用的公共项目（1000到2000行），该项目表明，即使是第一次采用洁净室开发过程，其生产率和质量要比交互式排错和集成方法更好。[3]

管理的观点

初看起来，统计式质量控制和软件开发似乎是矛盾的。统计式质量控制大概适用于制造行业，特别适用于批量制造事先已规定并设计好的部件或产品。软件开发似乎是单产品的逻辑过程，不具有统计属性。如果软件在某种条件下出错，那么它将在这些条件出现的所有场合下出错。

然而，从管理的观点重新考虑一下统计式质量控制过程，我们可以发现，借助统计式质量控制进行软件开发的做法具有显著的管理效益。

但是，由于软件及其开发不具有统计属性，那么统计性从何而来呢？这种统计性来自软件的使用，而不是其内在属性。统计式质量控制下的软件工程不仅要求规定软件的功能性行为，而且还要求规定软件的统计使用。

洁净室软件工程是把软件开发置于统计式质量控制下的一种实践过程。许多现代制造技术已充分地说明了统计式质量控制过程的重要性，这些技术把输出的抽样直接反馈到控制质量的过程。一旦统计式质量控制原理得到运用，管理部门可以看到开发过程，并能控制过程的变化来控制产品的质量。

洁净室开发过程能使我们准确地组织规格说明书、设计和测试间的开发工作，并且使过程的每个组成部分的责任更加明确。这种组织方法提高了管理部门监督开发过程的能力。经验不足的软件管理人员经常犯这样的错误：不去认识和暴露软件开发早期出现的问题（如硬件规格说明书的不稳定性，人员经验不足，和设计方案不完整），并错误地以为以后能解决和处理这些问题。洁净室开发过程能促使这些早期发生的问题公开化，给所有管理人员以解决这些问题的机会。

洁净室开发过程要求以稳定的规格说明书为基础。因为规格说明书在开发初期常常得不到充分了解或验证，所以初看起来，洁净室开发过程似乎不适用。但事实上，洁净室开发过程的原理很有用，它能促使规格说明书的缺陷公开化，并从管理上控制规格说明过程。

只要把开发过程看作是一种试错（trial-and-error）过程，调整规格说明书的不完备性就只不过是再多一次试错而已。不过，这将会削弱规格说明人员与开发人员间的责任心。最好的方法是根据早期的在每次反复中都保持稳定的规格说明书开发软件。这样在规格说明和开发之间能建立起明确的责任，并从管理上控制规格说明书的变化。

统计式质量控制

统计式质量控制是以生产者和接收者间的协议为出发点的。协议的最关键部分（不管

是明显的还是隐含的）是怎样度量质量，特别是统计式质量。对于具有直接的物理的、电的或其它度量的简单产品来说，协议很简单。例如，某种细丝中的99%的电阻都在一固定值的10%以内。然而软件是相当复杂的，其质量度量不同于简单的产品，它要求对怎样度量统计式质量有新的理解。

即使对于最简单的产品，也没有绝对顶好的质量统计度量。例如，有多种计算统计平均值的方法：算术平均值，加权平均值，几何平均值和倒数平均值等，而每种方法在不同的场合下都可能是较好的。

任何情形下，最终都归结为从业务和管理上进行判断。在大多数情形下，这种判断实际上是根据经验和先例自动产生的，但它毕竟是一种判断。在软件的情况下，判断没有先例可资依据。这是因为在统计式质量控制下开发软件的概念才刚刚提出。

Currit, Dyer, 和Mills[1]一文给出的对于软件质量进行认可的新基础[4]是依据一种新的软件工程过程。这一基础需要有软件规格说明书和关于软件使用情况的概率分布，然后规定一个测试规程和从测试数据结果获得的计算，以便使提交的软件具有认可的统计式质量。

这个新的基础代表了一种公平合理的科学工程方法，可用以度量软件的统计式质量。对于较简单的产品，没有绝对的最佳，也没有脱离开业务和管理的判断的逻辑依据。但在签订合同时，在尚无合理的条款存在时，可以用软件的统计式质量作为合同条款。

获得软件质量认可离不开在统计测试过程中从使用情况的概率分布测得的可靠性。认可是业务常规过程。在软件认可中，有一个发现事实的过程，随后按预先的规定进行计算。软件除了可靠性外也还有其他度量（如可维护性和性能）。因此软件质量认可是一种业务度量，在生产和接收软件中是全盘考虑的一部分。

一旦有了度量提交软件统计式质量的基础，自然就要建立相应的统计式质量控制的管理过程。原则上，我们的目标就是找到一种方法，能反复地给出软件开发过程中的最终的度量结果，在必要时对开发过程进行修改，以达到所要求的统计式质量水平。

洁净室开发过程是为实现这一原则而设计的。它要求软件以增量方式开发，以便在开发过程中对统计式质量进行实际度量，并通过补充测试和/或改变过程（如增加检查和配置控制）来改进被度量的质量。

数学验证

如果没有数学验证，软件工程就只不过是图有虚名。当Dijkstra在早期的软件工程会议上提出结构化程序设计的思想时[6]，他的主要动机就是通过只使用少量的基本控制结构并消除goto语句来缩短程序的数学验证长度。

许多一般结构化程序设计者们，阉割了其严密的数学验证部分，而只赞赏较容易的无goto部分，这实质上也就抛弃了结构化程序设计的许多实际优点。结果，许多人对不要goto语句津津乐道，而不知道工程软件中的数学验证的基本原理是什么，哪怕已经发现有了这种原理。

与此相反，如果懂得了数学验证的严密性，那么就会使程序员个人或集体的行为发生改变，不管程序是否用形式化方法进行验证。数学验证要求有精确的规格说明书，以及关于此规格说明书的正确性的正式论据。

不难看到两个主要行为效果是：第一，程序员（和管理员）之间的通信变得更加准确，特别是关于程序的规格说明书的通信。第二，鼓励开发最简单的程序，达到规定的功能和性能。

假如一个程序看起来很难验证，就应该修改程序本身，而不是修改验证。其效果是能够以很高的生产率开发出几乎不需要进行排错的软件。

洁净室软件工程就是在统计测试之前用

数学验证来代替程序排错。数学验证是由人进行的，它以标准的软件工程实践[4]（例如在IBM软件工程研究所中所教的内容）为基础。我们发现人的验证和统计测试之间的协同作用效果极佳，因为数学上的出错性质和排错的出错性质很不相同，那些由于数学上出错带来的错误要比由于排错出错带来的错误更容易用统计测试发现。

也许将来软件的自动验证将会实用化，但也没有必要非得等到那一天才认为数学验证的原理是有价值的。

用洁净室开发过程验证和其他传统排错技术开发同一项目，从实验数据来看，洁净室开发过程验证的软件显然具有较高的质量。对于验证过的软件，发现的错误较少，错误的严重程度较轻，而且用来发现和改正错误的时间较短。验证过的产品还具有更好的现场质量，所有这些都是由于设计时更加细心的缘故。

一些早期的洁净室工程项目（其软件产品的大约一半功能经过验证）表明，经过验证的软件的错误数只占1/4。此外，在严重的错误中不到10%应由验证过的软件负责。这些发现证明：验证过的软件含有更少的错误，并且错误也是较简单的，对产品运行影响不大。

洁净室开发中采用的由人进行数学验证的方法称为功能验证，它与大学里通常讲的公理化验证方法很不相同。它以功能语义为基础，并把软件验证尽可能直接地转换成对集合和函数的一般数学推理。

使用功能验证和尽早把验证转换成对集合与函数推理的动机出于验证规模增大的问题。能用三行普通数学记号写出来的集合或函数，得用300行英文来描述。在300行英语中出现的人为错误的可能性要比三行数学记号多，但是验证的范例是相同的。

通过以集合和函数形式进行验证，就可以为扩大推理规模奠定基础。大的程序有许多变量，但只有一个功能。Mills和Linger[6]给出了用集合、栈和队列而不用数组和指针来验证大程序的另一方法。

尽管功能验证比起公理化验证初教起来要难，但它可以用来对顶层设计中长达一百万行的系统和对底层中只有一百行的程序进行推理。这样的推理是有效的，因为在自顶向下设计的大系统中，用功能验证方法所作的回溯很少。

洁净室软件工程

尽管洁净室软件工程初看起来象一场革命，但它实际上是软件开发的进化，这种进化在于取消了排错，这是因为过去20年中，越来越多的程序采用设计语言进行设计和开发，它们必须加以验证而不是去执行。因此，相对来说，先进的开发小组现在用在排错上的工作量要比用在验证上的工作量少得多。即使对非洁净室开发来说也是如此。

这种软件开发的进化还在于统计测试方面，因为为了得到更高质量的程序，典型用户测试在整个测试的工作量中占有越来越大的比例。并且如上所述，在由人作的验证和统计测试之间有着惊人的协同作用：人们在进行验证时可能出错，但是留下来供系统测试的错误比起排错所留下的错误更容易发现和改正。

在早期的一个洁净室开发项目中，分别用验证和排错方法开发了不同的软件部件。结果表明，用于改正验证过的软件错误所用的时间，仅仅是用于改正排错过的软件错误所用的平均时间的五分之一。在验证过的软件中，开发者基本上可以不再求助于排错（例外的情况少于0.1%）来找出和改正发现的错误。

验证与统计测试这二种方法相结合是可行的，由此有可能提出一种在统计式质量控制下进行的新的软件工程过程[1]。为此，我们定义一种新的逐步增量开发生存期，以便实现功能的结构化规格说明和统计用法。结构化规格说明书是一种形式化的规格说明书（一个关系或者有序对的一个集合），对于

每个逐步增量开发产品，它可分解为一些子规格说明书的嵌套集合。结构化规格说明书不仅定义了最终的软件，而且必须提出软件增量实现和统计测试的计划。

软件要求的逐步求精和分解不断产生出软件设计的各个层次。在分解的每一层次上，以数学为基础的正确性论据保证了有关设计的准确度，以及产品要求的不断集成。这种工作步骤是建立规格说明书，对它进行设计，并在继续作下一步分解之前检查设计的正确性。

洁净室开发过程的设计方法使用一组个数有限的设计原语来刻划软件逻辑（顺序、选择和重复）；还采用模块和过程原语把软件设计组装成产品；对软件数据需求的分解采用一组数据结构原语（集合、栈、队列）以保证产品设计具有强类型数据操作；用专门定义的设计语言编写设计文档，并能直接变换成标准的程序设计形式。

在洁净室开发模型中，需要用到设计知识的结构测试被形式化的验证所代替，但是功能测试仍然保留。事实上，功能测试可以用于两个目的：1）表明对产品的要求在软件中已正确地实现，2）为产品可靠性预测提供基础。后者是洁净室开发过程特有的功能，来源于它的统计测试方法，支持从测试到操作环境的统计性推断。

洁净室开发过程的增量型产品开发期在整个产品开发过程中（而不仅仅在开发完成后）支持软件测试。这样，就可以根据执行情况，对产品质量作不断的评价，并且在必要时调整开发过程以改进产品质量。

每当产品准备好后，即可用统计测试对产品可靠性作出统计评价。软件过程分析和反馈可以用来使随后的产品达到可靠性的目标（例如，加强验证检查，使更多的中间规格说明书形式化）。在系统测试过程中，随着错误的发现和纠正，也可以对逐步成熟的系统进行可靠性增长的估计，以便在最后推出产品时提供认可的系统测试的软件可靠性。

Cho[4]还建议在统计式质量控制下进行软件开发时，把正确输出与所有输出之比率作为一种度量。他把软件看作是一个生产输出，而不是生产产品本身的工厂。正确输出在所有输出中所占的比率与平均无故障间隔时间直接有关，在这里把时间作为输出生成的规范。这种规范化在洁净室开发过程中是一种可行的方法，在大多数系统应用中用别的规范化可能更有意义。

洁净室开发过程与Cho的思想之主要不同点在于在开发过程中对可靠性增长的认可模式。另一个主要差别在于是否坚持在系统一级的典型用户测试前由人进行数学验证而不进行程序排错。正象Mills所说[9]，由人进行的数学验证对于实现高生产率来说不仅是可能的也是实际可行的。用在验证上的时间可能比用在排错上的时间少。

统计基础

软件人员习惯于谈论软件中的错误数，通常用每千行代码中的错误数进行衡量。现在，一般软件在交付时每千行有一到十个错误。采用好的方法论可以使每千行中的错误数少于一个。但是在考虑软件可靠性时，这些数字并无直接关系，而且还会使人发生误解。由于用户看不到软件中的错误，而只能看到执行时的故障，所以用故障的间隔时间来度量更为合理。

如果每个错误都有相同的或相近的故障比率，那么软件中的错误数和执行过程中故障的间隔时间就具有直接的关系。错误个数减半意味着故障比率减半和平均无故障时间加倍。在这种情况下，减少错误数将自动提高可靠性。

IBM的每个主要软件产品，无例外地都有一个极大的错误-故障比率的变化范围。在稳定的产品中，故障比率从故障间隔时间为18个月到5000年不等。一半以上错误的故障比率是故障间隔时间不少于1500年。改正这些错误将使错误数减少一半以上，但产品故

障比率的减少却是微乎其微的。更确切地说，在改正60%以上的错误后，故障比率最多只下降不到3%。这些对于软件可靠性传统常识的令人惊奇的否定，归功于Adams[10]多年辛勤工作所获得的数据。

为了更精确地讨论软件的错误和故障，我们假定存在一个规格说明书及其软件。当软件运行时，可以把它的行为同规格说明书对比，找出不一致之处（故障）。这些故障可能是灾难性的，它使软件无法继续运行（例如，异常终止）。另一些故障可能是严重的，从那以后得到的每个回答都是错误的（例如，数据库遭到破坏）。不太严重的故障是指软件可以继续运行，并且至少有部分行为是正确的。这些例子表明，故障主要有三种不同程度的严重性：

- 终止性故障
- 永久性故障（但非终止性故障）
- 偶然性故障

即使是终止性故障或永久性故障，也都可以随后重新启动，因此我们可以想象有一个很长的运行历史，在发生故障的时刻上作上标记。很清楚，运行历史与软件的初始条件（和数据）和随后的输入（如命令和数据）有关。运行历史有很大的随意性，但是为了讨论方便起见，不妨假定典型的历史（使用的情景）。

对于相同的初始条件和相同的使用历史，软件的行为是确定的，产生同样的输出（和同样的故障）。但是，事实上，如果软件被多个用户按多种历史使用，那么使用历史通常是不相同的。因此，我们把使用历史的概率分布也作为结构化规格说明书的一部分，典型的作法是把它定义为随机过程。

使用历史的概率分布可以用来导出故障历史的概率分布，并可用它定义和估计故障间隔时间，无故障时间区间，以及诸如此类随机量。所以，即使软件的行为是确定的，但其可靠性也可以相对于它的统计性使用来定义。使用历史的这种概率分布为软件质量控制提供了统计学基础。

对统计式质量的认可

对于已提交的软件来说，估计可靠性的简单方法是使用平均无故障时间，即在统计测试过程中计算出故障间隔时间的平均值。然而，对于在洁净室开发过程下开发的软件来说，问题更为复杂，理由有二：

1.对于每一洁净室开发过程增量，系统测试的结果可能表明对软件要进行修改以改正测试中发现的故障。

2.当交付每个洁净室开发过程增量时，将把未测试过的新软件加到已测试的软件上去。

事实上，为改正一个交付增量中的故障而做的每次修改，都产生出一个新的软件产品，它与原来的软件十分相像，但是具有不同的可靠性（有意使之更好，但也可能更差）。然而，这种改正后的软件产品在它被后继者取代之前只受到极有限的测试，因此统计估计可靠性的可信度也相应地受到影响。

所以，为了收集每个交付增量的测试经验，我们定义了一个可靠性变化的模型，它有两个参数M和R（参阅Currit, Dyer, 和Mills[1]）。对软件作c次修改后，平均故障间隔时间$MTTF=MR^c$，其中M是初始平均故障间隔时间，R是观察到的每次软件修改后改进的平均故障间隔时间的比率。

虽然Currit, Dyer, 和Mills[1]为此模型给出了各种技术依据，但是对于由开发者最后向用户交付的软件的最终认可，还是应该以契约为基础。此外，由于无法知道参数M和R是否绝对正确，我们一般只根据测试数据来统计地估计M和R的值。对估计值的选择以统计分析为基础，但是参数的选择也应考虑认可的契约依据。

可靠性修改模型和对它的参数的统计估计，这两个契约依据的作用在于给生产者和接收者（卖方和买方）提供一个共同的客观途径来认可提交的软件的可靠性。这种认可是科学的、统计性的推论，它是通过对由开

发者保证是正确的测试数据进行预先写明的计算得到的。

原则上，软件可靠性的估计值不光是一种计算平均无故障时间的方法，而且还考虑了统计测试期间的修改行为。当测试数据确定后，就可以在每次修改之后估计可靠性。若改正是成功的，那么通过随后的测试，可靠性估计将不断改进，从而给达到可靠性目标提供客观的定量证据。

这种客观的证据本身就是为达到可靠性目标而对软件开发进行管理控制的一个基础。例如，过程分析可以揭示未预料到的错误来源（如对所用的硬件理解不够），并为以后的增量着想适当地改进过程本身。最终认可的中间"演习"给管理以反馈，从而达到最终目标。

处理各个交付的增量也应作为开发者与用户间契约基础的一部分。也许最简单的处理方法就是独立地对待各个增量。然而，最终认可的更高统计可信度来自概括各增量间测试经验。简单的概括方法是对分别处理的增量进行管理判断。

处理各个交付增量的较复杂方法是，把软件的各个新交付部分的故障给与率模型化，并为各个交付增量逐一提出分层的估计。当后面的交付增量被测试时，早些的交付增量可能已成熟。这种可靠性改进的成熟速率可用来估计为达到预定可靠性级别所需的总测试时间。

对于项目管理来说，平均故障间隔时间及其变化速率是一个有用的决策工具。对测试中的软件，既有对平均故障间隔时间的估计，又有已知的平均无故障间隔时间的变化速率，因此对可靠性的判定可以依据生存期的费用估计，而不只考虑市场影响。

在软件提交后，发生每一故障的平均费用中必须包括修复故障所花的直接费用和给用户造成的间接费用（后者可能要大得多）。交付后的这些费用可以根据预期的故障数来估计，并同交付前额外测试所花的费用进行比较。根据使全生存期费用最少的原则来判断继续测试是否合算。

参考文献（略）

〔朱　力、张　然　译自《IEEE Software》September 1987 pp19—24仲　源校〕

（上接22页）

左端具有该系统那部分的所有规则，并识别在对应的右端的所有子图来弄清这种改动对下一阶段产生的潜在影响。

这是一种描述软件开发过程和关系的新方法，所以象"基本"性质（如阶段中的控制流）一样，阶段间不存在已确立的"基本"关系。因此，我们对构成一个阶段间模型的规则形式不做更多的假设。如果模型建造者希望阶段间任意复杂的关系，那么可以使用相应复杂的规则来描述这些关系。然而，当使用一种具体的软件开发方法时，所有规则可能表现出一些具体的模式。这种情形的最明显例子出现在"逐步求精"的功能分解过程中。这个过程由只能扩充系统最低级动作的一些推导步骤表征。这些扩充的目的是更加详细地表达构成低级动作的活动。描述这一过程的图重写规则在左端必须恰好有一个动作结点，它被右端的一个非约束图模型代替。逐步求精的所有图重写规则将适合这一模式。

（未完待续）

〔仲　源译自《IEEE Trans. Software Eng.》№8, Vol. 14, 1988, p1128-1144　声　钟校〕