**读 书 笔 记**

**目 录**

[（1）测试经济学 2](#_Toc3635552)

[（2）测试的价值 Matt Heusser 3](#_Toc3635553)

[（3） Lei Feng（瑞典皇家技术学院）：“基于学习测试的案例研究” 4](#_Toc3635554)

[（4）优化软件测试投入的方法（Michael Larsen） 6](#_Toc3635555)

[（5）软件测试自动化最新进展 7](#_Toc3635556)

[（6）从测试自动化到智慧测试（Jeff offutt） 7](#_Toc3635557)

[（7）组合测试引论（D. Richard Kuhn） 9](#_Toc3635558)

[（8）大数据与网络表征 12](#_Toc3635559)

[（9）软件工具认证考虑 12](#_Toc3635560)

[（10.1）数据科学家通过历史可以学到什么？Aaron Lai (1) 13](#_Toc3635561)

[（10.2）数据科学家通过历史可以学到什么？（2） 16](#_Toc3635562)

[（11）服务化测试 PankHuri Mishra 18](#_Toc3635563)

[（12）理解测试债 Ganesh Samarthyam 39](#_Toc3635564)

[（13）软件测试大数据的思考（一） 46](#_Toc3635565)

[（13）软件测试大数据的思考（二） 47](#_Toc3635566)

[（13）软件测试大数据的思考（三） 48](#_Toc3635567)

[（14）动态语言程序中的开放对象的自动分析 50](#_Toc3635568)

[（15）多人决策过程中的人际欺骗：通信模式分析的新认识 51](#_Toc3635569)

[（16）安全关键软件的重用与系统集成的形式化方法 53](#_Toc3635570)

[（17）关于软件重用的几点思考 55](#_Toc3635571)

[（18）软件评审工具及工作原理 57](#_Toc3635572)

[（19）装备试验需求的变化对软件测试的影响与对策 59](#_Toc3635573)

[（20）行为与交互中的时间与自相似结构：从序列到对称与分形 60](#_Toc3635574)

[（21）737MAX事故引发的思考 62](#_Toc3635575)

# （1）测试经济学

测试经济包括资产和负债两个方面。一个项目发现的软件代码缺陷数是资产，测试后交付用户试用发现的代码缺陷数、未关闭的缺陷数、投入正式使用后发现的故障数乘以损失影响因子，1到10，是负债。

资产减去负债就是静收益，再除以资产，就是收益率。

比如，测试发现了800个缺陷，其中55个不做修改，试用发现25个缺陷，正式使用发现5个缺陷，总影响因子是3，则净收益是705，收益率是88%。

好的测试项目就是得到高的收益率。

**讨论：**

在发现代码问题数一定的情况下，关键是降低负债。

首先，要降低不修改缺陷数。不修改缺陷通常是被认为影响不大或发生可能性很低的问题。这个和我们通常的理解不同，因为我们不是从用户和产品的角度看这个问题，而是从测试的角度看。

其次是加大力度分析试用和正式运行场景下可能发现什么问题，从而针对性地设计测试。我觉得这一点往往被测试团队忽视，往往只关注系统环境和真实环境的差异，不太重视最终用户可能的使用方式，特别是一定长时间内的使用方式。

支持环境/工具需求思考：

以需求文档为基础的测试工具现在有很多了，但是辅助测试团队快速抽象、建模软件系统真实使用操作的工具还是太少，影响测试项目的收益率。

# 

# （2）测试的价值 Matt Heusser

提高价值最蠢的办法是降低测试人员的收入，导致测试人员生产率降低。

测试人员的身心投入程度是测试项目成功的关键。优秀测试人员会花很多时间使用、琢磨被测软件，会发现不那么投入的测试人员花再多的时间也发现不了的软件问题。

好的测试人员花在使用被测软件的时间应该在70%，出色的应该在90%。

软件测试的价值在以下三个方面体现：

1、帮助得到好的需求

2、快速发现最重要的缺陷

3、为决策层和开发人员提供有意义的信息

**思考：**

我想这要求测试人员具备以下能力：

1、被测系统的用途、使用环境、可能的意外情况有深层的了解，而不只是对软件需求规格说明的理解，有能力审视、发现软件需求的内容不足，而不只是格式或一致性等方面的问题。能够从测试人员的角度发现软件运行可能的隐患，而不仅限于符合性评价；

2、应对缺陷模式有深刻理解，清楚什么是最重要的缺陷，对每个软件来说，重要缺陷都是不同的，既要关注发现缺陷的数量，也能快速发现软件的关键问题，测试优先级考虑；

3、通过在无穷测试路径中选择极为有限的测试并给出测试结论其实是非常困难的，通过获得怎样的信息降低决策风险对不同的软件是不同的。

总之，测试人员的个人素质、投入程度对于高价值的测试是决定性的。感觉现在的测试管理倾向是首先不要犯错误，一切不足都推给客观因素，比如需求文档质量、环境制约等，长此以往很可能会拉低整个测试行业的价值水平。只有不断提高满足客户需要的水平，才能真正提升软件测试的价值。

**服务内容思考：**

协助测试实验室认识其委托方的需求；

针对委托方需求协助制定实验室发展路线图；

协助实验室发展路线图设计、提供技术和管理平台；

协助实验室评估委托方需求的满足程度，进而发现不足找到新的需求。

这个过程是动态的，因为委托方需求会不断提升、涌现的，其他工作必须不断调整。

委托方需求与实验室能力是相互促进的，但是源动力还是委托方需求。

有些需求几乎是永恒的，比如更少的测试经费，更短的测试时间，更高的测试质量。哪个实验室在这些方面有突破，就会脱颖而出，赢者通吃；哪个服务商能够更有力的推动实验室达到这个目标，哪个服务商也会脱颖而出。

# Lei Feng（瑞典皇家技术学院）：“基于学习测试的案例研究”

**3.1**

对于软件测试的发展趋势是从自动化向智能化发展，这是被很多人认同的观点。不过我一直以来都很困惑，对于软件测试来说，到底需要机器学习什么？更具体地说，通过学习有可能得到什么类型的有价值的软件测试模式？

论文研究了三个案例，一是电子商务应用，涉及到负载均衡和数据管理；二是沃尔沃公司一个新型ABS控制系统，由主控嵌入式软件和传感器组成，每个车轮有一套传感器；三是瑞典一公司的金融服务软件，包括风险评估和控制。

案例2和去年全国大学生软件测试大赛的题目非常近似。因为很多人都熟悉这个题目，这里不妨多介绍一下，看看机器学习的测试到底有没有可能击败大学生选手。

案例2有三个需求：

1、如果刹车板被踩下，且车轮速度大于0，则对应的ABS组件驱动刹车装置的输出大于0；

2、如果刹车板被踩下，且车辆速度大于每小时10公里，而防抱死传感器输出为抱死，则对应ABS组件驱动刹车装置的输出为0；

3、如果刹车板和油门板都被踩下，则车辆减速

测试这三个需求，人工设计一般需要多少测试用例？（包括不同的测试输入组合数）猜一下基于学习的测试工具对这三个需求，自动生成、执行、与预期结果（这个也是工具自动生成的）比较的测试输入组合数是多少？提示一下，总执行时间40分钟，需求2没有通过测试，发现了传统手工和自动化测试没有发现的缺陷。

这个是基于模态逻辑黑盒需求描述的软件测试，基于模型和基于学习的测试相结合，扩展了一点时态描述能力。状态是有限的，通过学习压缩测试空间，注意，每个车轮传感器和控制逻辑既独立又关联，不是组合问题。

制动防抱死系统（antilock brake system）简称ABS。作用就是在汽车制动时，自动控制制动器制动力的大小，使车轮不被抱死，处于边滚边滑（滑移率在20%左右）的状态，以保证车轮与地面的附着力在最大值。

ABS系统的发展可追溯到20世纪初期。进入20世纪70年代后期，数字式电子技术和大规模集成电路迅速发展，为ABS系统向实用化发展奠定了技术基础，许多家公司相继研制了形式多样的ABS系统。自20世纪80年代中期以来，ABS系统向高性价比的方向发展。有的公司对ABS进行了结构简化和系统优化，推出了经济型的ABS装置；有的企业推出了适用于轻型货车和客货两用汽车的后轮ABS或四轮ABS系统。这些努力都为ABS的迅速普及创造了条件。ABS系统被认为是汽车上采用安全带以来在安全性方面所取得的最为重要的技术成就。

记得刘畅教授在28组织的软件测试技术研讨会上分析了一个汉莎航空公司飞机事故的例子，就是飞机着陆时的轮子控制问题。他当时采用的分析方法主要是枚举，会上讨论时我还特别提到形式化描述问题

因为需求1和需求2是交替发挥作用的，这样才有滚滑结合的最佳制动效果。论文篇幅受限，不可能描述所有需求，对本例就是滑滚比20%（谢谢张总的补充）。有兴趣的可以考虑一下这个软件测试需要设定哪些关键场景和控制要素，如何覆盖以提高测试的有效性。

**3.2**

针对黑盒需求测试的基于学习测试自动完成三个基本步骤：测试用例生成、测试执行和测试预期比对。重点是生成大量高质量的测试用例。

基于学习测试工具的核心是模型检查算法和学习算法，其中学习部分主要是针对模型进行，即增量模型推导。

被测软件与两个算法集成一起，构成交互式反馈环。模型检查、学习和随机生成三种用例生成方法交替使用，以提高执行效率。模型生成的依据是形式化用户需求描述，并选取正确性验证的伴随案例。每次迭代模型都会精化为新的模型，以确保迭代持续进行。如果给出有限且足够描述被测软件的信息，而学习算法能够正确地学习，则基于学习测试就可以当作一种完备和可信的测试方法。但是实际软件系统往往很大，有限时间内不可能完成学习。而增量学习算法只关注与需求描述有关的被测软件部分，可大幅提升基于学习测试的有效性。

基于学习测试与其他方法相比主要有两点不同：关注测试而不是验证，使用优化算法。作者已经开发出实用的基于学习测试工具用于实际软件系统。有些基于模型的测试工具依赖设计模型如UML生成测试用例，而基于学习测试工具对自身模型进行逆向处理，因此修改代码后不需要手工同步修改模型，特别适合敏捷开发。

基于学习测试的前提：能够使用确定的有限状态机对被测软件建模。

目前工具支持的用户需求建模语言是命题线性模态逻辑，用来描述安全性属性，即不能违反的情况，和执行属性，即期望的动态行为。

工具的模型检查器是通用的MuSMV，学习算法是通用的IKL算法。

结论：

基于学习测试已经可以实用，但还可以使用更好的学习算法改进；

基于学习测试可用于不同的软件系统领域和问题。

未来考虑：

采用更好的模型检查器，即一阶线性模态逻辑，不过也考虑更简单但不那么精确的划分测试方法。

# （4）优化软件测试投入的方法（Michael Larsen）

我想行业最习惯的方法就是收集、确认“测试需求”，最好是条目化的，最好是有详细的用户使用指南，从而“追踪”每条软件要求设计针对性的测试需求。如果测试项目输入达不到理想要求，则强迫达到或协作达到。

遗憾的是，很少遇到理想的测试输入，即使研制单位是行业的领先者，研制过很多类似产品，即使研制单位通过了5000二级、三级认证。于是很多单位的工作转向“帮助”他们提供理想的测试输入，而这被研制单位看作是额外负担。

既然在工作目标上达不到双赢，缺少对方的积极性，目标不能顺利达到也就成为必然。

我们可以借鉴一下作者弄清五个问题的思路，将关注点从通过测试转移到降低软件使用风险上，从而形成共赢和合力。

一、确定用户最关心的问题：产品给谁用？用来解决什么问题？测试策略能否回答第二个问题？（对于装备来说就是形成怎样的战场能力）

二、优先关注形成战场能力风险最大的问题。可能是兼容性、可能是易用性、可能是接口

三、确保提出合适的问题。用户特点、使用场景、现有相关系统、与类似系统相比的优势等

四、理解软件的关键问题和问题因素。很多测试人员会把系统崩溃归为关键问题，其实关键作战原则的违反、战场情况没有清晰提供给用户从而影响关键作战决策等，可能更为关键。

五、完美是“好”的敌人。可能会发现很多小问题，小的性能问题。但是测试的目的是利用有限的资源，尽可能好地帮助交付管用、可靠的装备。关注使用风险价值高的软件问题。

我参加过不少项目评审，测试组的大纲汇报往往直接进入测试细节，往往没有深入理解上面的五个方面，我想这是当前一些测试项目质量不够理想的重要因素。

# （5）软件测试自动化最新进展

1、哪些测试活动可以自动化？

2、语音识别系统开发与维护中的自动化软件测试运用

3、分布式程序的运行时验证

4、关于降低测试成本：一个回归测试案例

5、软件信息安全性测试案例研究：静态代码分析

6、运用Cuckoo搜索确定测试用例优先级

7、软件质量方法论概述

8、分布式功能的基于模型测试

9、具有学习能力的Web应用测试

10、网络化服务的测试

# （6）从测试自动化到智慧测试（Jeff offutt）

手工测试，由人来确定输入值、测试步骤和预期结果等。需要人工完成测试用例的执行、补充、修改和删除。是“单细胞”测试。

自动化测试，在人和测试数据生成器的协作下确定输入数据，并满足确定的测试完成准则（覆盖率），测试步骤形成自动化的测试脚本，通过断言确定预期结果，支持快速和反复执行。是“多细胞”测试，带有一点非常初步的智慧色彩。

自动化测试使用“可达-感染-传播-可视”模型。即测试用例激活缺陷，使软件进入不正常状态，再传播的不正常最终状态，通过与测试预期比较，可揭示失效。

自动化测试的关键，是可控制性和可观察性。即软件的输入、操作、环境变量、外部数据存储的确定是否容易，软件的输出、对环境和其他硬件和软件组件的作用的揭示是否容易。

自动化测试的本质是基于模型的测试，将模型层的抽象测试用例转换成实现层的具体测试用例。比如采用有限状态机模型，通过节点、有向边和事件刻画被测软件。通过模型和测试准则形成测试需求，通过测试需求形成抽象测试用例。 另一方面，被测软件的具体实现可转化为测试组件，测试组件和抽象测试用例的结合可转换为具体测试用例。

这样测试人员的工作就成为：找出模型元素所对应的软件代码，并映射到测试自动化工具，工具自动创建具体测试用例后，进行一定的调试。

借助工具完成测试用例生成所需的时间是手工进行测试用例生成时间的四分之一左右，但是后者会引入很多前者不会引入的错误。

据试验，一个232行的C++程序，手工完成满足有限状态机模型覆盖要求的240个测试用例，需要661分钟，借助工具需要145分钟。

测试预期结果的确定非常重要，对于C++来说，需要考虑模型定义的状态常量、状态迁移时调用方法的对象成员变量、方法的返回值、做为参数传递给发调方法的对象成员变量等。

据研究，由于测试预期结果的设计问题，大学生、研究生、全职软件工程师和全职测试工程师的失效遗漏率高达21%到39%。

而自动化工具通过插入大量断言，可以大幅降低遗漏比率。

“多细胞”测试的主要问题是：不知道测试用例的存在原因，不知道什么时候应该运行，不知道什么时候需要修改和退役。

智慧测试用于解决这类问题，具有自我认识和自我判断的能力。

每个测试用例必须说明其用途：为什么要创建这个测试用例？测试用例用于检查什么？测试用例的需求是什么：功能需求、准则（例如有向边覆盖）。需要一种语言来描述。

测试用例必须在特定的场景和社会中。这里的场景指测试用例自我执行的运行框架，社会是指就状态和变更检索环境和其他测试用例的能力。

智慧测试可以：主动自我运行，测试失败后主动发出通知，当软件变更后能确定是否运行、变更、退役或不动作，当测试用例变更后能够要求测试人员进行必要的帮助，以确定输入、预期结果等。

智慧测试的先决条件：测试自动化，软件到测试用例的可跟踪性。

在技术上需要：描述用途的语言，测试用例选择行动的规则集，测试用例执行的环境、查询资源库的能力、测试用例开发框架等。

智慧测试替测试人员作出一部分思考

# （7）组合测试引论（D. Richard Kuhn）

软件故障有很多是“交互故障”，即只有当两个或更多输入值相互作用时才会发生故障。NASA应用出现的故障，67%是单要素故障，93%是二要素故障，98%是三要素故障。其他领域的应用，80%左右是二要素故障，90%作用是三要素故障。更多要素的交互故障占比很小，六要素以上的交互故障基本可以忽略。

组合测试可以在不考虑程序结构的情况下，评估测试用例设计的充分性，同时能够以最少的测试用例覆盖更多的交互故障发生空间。

组合测试覆盖度量有以下几个指标：

一、简单t要素组合覆盖，即给定一组n个变量测试用例集合，n个变量的t要素组合数，除以所有可能的变量-取值配置数。

举例：

对于a、b、c、d四个二值输入的软件，测试用例集合中4个测试用例：

a b c d

0 0 0 0

0 1 1 0

1 0 0 1

0 1 1 1

对于二要素，共有4个可能配置：00、01、10和11。在总共6个二要素组合中，ab、ac、ad和cd不能满足覆盖全部4个可能配置的要求，只有bd和cd满足。这样，这组测试用例集合的“简单2要素组合覆盖率”就是33%。

对于三要素，共有8个可能配置：000、001、010、011、100、101、110和111。在总共4个三要素组合中，abc、abd、acd和bcd都不能满足覆盖全部8个可能配置的要求，那么简单3要素组合覆盖率为0%。

2、简单（t+k）要素组合覆盖

满足t要素组合覆盖率的测试用例集合也会具有一定高阶要素组合覆盖率。

例如将前面的例子进行扩展，得到：

a b c d

0 0 0 0

1 1 1 0

1 0 0 1

0 1 1 1

0 1 0 1

1 0 1 1

1 0 1 0

0 1 0 0

对这个测试用例集合，简单2要素组合覆盖率为100%，而3要素组合覆盖率也可以达到25%，4要素组合覆盖率为0%。这样这个测试用例集的简单（2+1）要素组合覆盖率是25%，简单（2+2）要素组合覆盖率为0%。

**3、三元密度**

将上面（t+k）中的k取1。三元密度就是t➕（t+1）元组覆盖值除以（t+1）元组所有的可能取值。

对于上例，既满足二要素的组合覆盖要求，同时还覆盖了4变量三要素全部24个可能配置中的10个，所以这个测试用例集的（2+1）元组密度=2+26/32=2.81

**4、变量取值配置覆盖**

即变量取值覆盖的百分比。对于第一个例子，对于4变量二要素的全部24种组合，19种被覆盖，只有ab=11、ac=11、ad=10、bc=01和bc=10没有被这个测试用例集合覆盖，其中只有bd和cd覆盖所有4种取值。那么该测试用例集合的配置覆盖率就是19/24=79%。

小结

1、组合测试对于发现交互故障非常重要

2、引起故障的参与变量数通常较少，一般2到3个，最多6个

3、组合测试既考虑了测试充分性，也兼顾可行性，一定程度上可缓解组合爆炸问题

4、通过算法可以用较少的测试用例数满足组合测试覆盖率要求

5、既满足低阶元素组合测试覆盖率，又尽可能多地覆盖高阶（通常为高1阶）元素组合，是更好的测试用例集

6、通过度量指标可以客观评价针对功能、接口、安装等关键常用测试类型的测试充分性，降低测试风险

7、组合测试覆盖针对黑盒测试，可一定程度弥补逻辑测试在程序规模、测试级别和程序设计语言、源程序可获得性等方面的限制，具有更好的适用性

8、组合测试不太适合时序问题等其他类型引起的故障。

# （8）大数据与网络表征

在真实应用中，大数据通常以关联形式存在。对于软件评测数据，故障数据当然不能脱离具体的应用场景、测试用例、需求模式、产生后果等而孤立存在。对软件评测大数据网络进行高效分析和处理至关重要，也是软件测评大数据的核心挑战。

基于网络的基础性作用，本来期望可以发挥重要作用，但事实并不乐观。腾讯有百亿级的社交网络，阿里巴巴有百亿级电子商务网络，但是不管是基于规则的浅层次分析，还是更前沿的网络分析算法，事实证明对于大规模网络都不可能在实际中得到大规模运用。

瓶颈在哪里？首先，网络的“边”给网络的分析和处理带来组合爆炸式巨大挑战。其次，“边”使得节点之间存在耦合，产生巨大通信代价，难以并行化计算。此外，网络除了用节点和边表示外，还有另外的隐形空间驱动网络的形成和变化，比如实践中测试人员和开发人员知识的积累，同类软件测试数据的可比性等。而这种隐空间是不可观测的。

核心问题是，是否可以将网络拓扑空间嵌入到隐含向量空间中，从而用向量空间中的距离替代网络分析中的边，以便分布式并行计算，突破网络分析和处理的瓶颈。

而“网络表征学习”或“网络嵌入”就是致力于如何将网络拓扑空间嵌入到向量空间中的研究领域。

目前这方面的研究已经有一些进展，期待可以用来推动软件测试大数据应用取得实质进展。

# （9）软件工具认证考虑

软件工具广泛用于辅助开发、验证和控制其他软件的很多领域。本文所说的软件工具，指用于开发、转换、测试、分析、生产或修改其他程序及其数据，或其文档的计算机程序或功能部件。

软件工具需要认证是因为：

1、工具与工具的软件不同，构成一个独特领域，因此工具开发者和用户都需要工具的专门指南；

2、工具常常由非工具用户的团队开发，工具开发团队往往不了解软件开发指南，如DO-178C或DO-278A，认证可以有助于工具开发团队避免模糊和误导情况出现；

3、本文件为机载和地面软件提供指南，也可用于其他领域和安全评估过程。

为了尽可能提高本文档的适用性和灵活性，定义了5个工具认证级别，但不是所有级别都适用于各个领域。

工具生命周期应形成以下文档：

1、说明工具及其认证需要的计划：《软件认证计划》；

2、所有活动的总结和状态，包括工具的认证状态《软件完成总结》；

3、软件环境的配置索引，包括工具标识和工具认证数据：《软件生命周期环境配置索引》。

本文假设工具用于裁减、降低或自动化软件生命周期过程，而无需验证工具的输出。因此工具认证的核心与被裁减的过程所对应的目标关联。

工具通过自动化需要完成的活动，可预知地执行人员容易出错的功能，在软件开发过程中辅助进行分析，潜在地提高系统安全性。但是如果工具不恰当地执行其预期完成的功能，工具中的错误有可能对软件功能产生负面影响。为了避免这种风险，保证工具功能的完整性，需要通过合适的过程开发和验证工具，这也是工具认证的目的。

工具认证是获得工具证明证据的必要过程，而证据只能在需要批准的项目背景下确认（工具认证与使用工具的软件关联。所以不要轻易说工具通过认证，A项目可以用的工具，B项目未必可以使用。不要对客户说太外行的话。一句夸大的话，客户会怀疑其他的话也有水分）。

工具认证工作与工具错误对系统安全性和软件生命周期过程的影响的不同而不同。工具错误对系统安全性的影响越大，工具的认证越严格。

工具认证过程可以对工具的功能、整个工具和一组工具进行。

# （10.1）数据科学家通过历史可以学到什么？Aaron Lai (1)

摘要：数据分析是技术进步和全球化的必然要求。数据科学家通过对问题形式化增加价值。必须深入了解问题背景，因为不完整的答案比不正确的答案还要糟——会产生误导。本文提出三种新的分析工具：制度、数据、策略，用于以可解决的方式定义问题。

**1. 引言**

Benjamin Disraeli说过，“我们最期待的很少发生，最不期待的往往发生。”度量分析、统计建模、数据挖掘非常重要。预测建模采用统计或数学技术预测某个目标组的未来行为。与预报不同，预测模型独立于时间，只受随机因素影响。预测建模假设人们作为一个组，在同样的条件下具有同样的行为。个体未被观察到的性格会产生偏差和错误。

数据科学不限于预测，它通过数据得到有意义的结论，包括统计分析、算法建模、数据可视化等。大数据需要创新问题求解方法。

我们认为，随着技术的发展，模型构建过程也会发生变化。适者生存，而不是强制生存。预测方法只有在得到恰当、创新的运行，才会得到闪光的结果。竞争压力使得预测变化变得太重要了，以至于需要专业人员从事预测工作。而常见的情况却是一知半解的分析师不加思索地直接使用现代分析工具。创新才是基础。

我们从三个不同角度研究模型的构建过程：制度、数据和策略。“典型”数据科学家要做什么？没有经验的分析师往往盲目运用从课本上学到的东西，而不知其所以然。

SAS永远完全自动化的预报系统，可以创建并适配一系列ARIMA模型，通过Tableau分析数据，可以得到用什么类型的图表最为适合的建议。有很多网站和公司允许用户提供问题和源答案。

**2. 案例1 战争资金**

1694年英格兰。王室处于严重的经济危机中，且没有简单的解决办法。

**2.1 背景**

威廉统治者从1066年以来的主要收入来源，是皇家资产和联邦土地出租系统。当时王室滥用地租系统，以至于贵族需要保护租地者的权利。到1688年荣誉革命之后，政府开支从1618年的50万英镑上升到1695年的620万英镑，债务1618年的80万英镑上升到1695年的840万英镑。

**2.2 问题描述**

政府债务急剧增加，即使几场耗资巨大的战争已经结束，仍然没有展现任何和平的曙光。王室需要大量经费建立军队准备未来的战争。王室的信用标准已经恢复，因此可以再举债。但是以税收做背书的长期贷款仅仅一年就用尽，利率飙升到14%。由于这些债务是以资产背书的，而皇家财政部已经用尽其高质量的资产。

**2.3 如果是我们处于那样的情况该怎么办？**

政府收入有两个来源：税收和举债。通过标准建模方法可以建立用于研究税收的经济学模型。税收最大化政策最大限度地利用人们的交税能力。

另一方面要估计举债容量。通过宏观经济学模型评估财务强度，以不至于超出信用资产。

**2.4 结局**

结果是两项创新帮助了降低成本、提高举债能力。一是金匠发明的杠杆储备，二是英格兰银行发明的并购。当时人们把金子和其他值钱的东西存给金匠，以得到保值，需要时可凭证书取出。因为只有金匠知道确切的存储数据，因此他们发现可以通过签发证书向王室和大众出借资金。利用存贷比例和利差管理风险。前者是运营风险，后者是信用风险。存贷比控制在50%到60%，利差则取决于不同的收益流，例如海关、东印度公司货物、酒类等。银行大量运用贴现（延迟兑付换取收益）改善长期债务的循环状况。

通过运用杠杆储备增加了借款供给，通过高质量的资产背书提高了投资吸引力。因此可以重新对这个问题建模。很多优化算法可以用来解决这个问题。通过选择分析还可以发现谁会购买什么类型的资产。通过蒙特卡洛仿真和基于代理的建模，还可以研究不同消费模式下的动态交互。

在对解决方案建模时，需要注意主代理问题。皇家财政部是为王室服务的，不是为投资大众服务的，因此任何解决方案都应是一种投资者视点的信用解决方案，需要确保政府最终有能力偿还债务。

英格兰银行还制定了很多措施保护投资者利益。

战争需要很多钱。英国具备便宜地借入大量长期债务的能力，是英国能够击败法国称霸世界的主要原因。

**2.6 商业运用**

这种并购模式被通用汽车公司用来向代理商提供资金支持，以很低的利率向消费者和代理商提供刺激性汽车贷款。而销售量的增加又进一步降低了生产成本。这种制度性的安排后来成为汽车行业的标准实践。

# （10.2）数据科学家通过历史可以学到什么？（2）

1. 伦敦瘟疫爆发

这是1854年8月的普通一天。百老街400号的刘易斯太太正在水中为她的孩子洗尿布。接下来她把水倒在屋前的污水坑里，全然没有意识到这个小小举动会引发水坑周边方圆250码内700条生命的消失，因为她的孩子感染了霍乱。

1.1 背景

霍乱最早发现于1817年。1831年8月，霍乱使圣彼得堡44万中的1万人死亡，1853年到1854年，英格兰和威尔士超过2万人死于霍乱。尽管人们已经对早前发生在印度的霍乱流行进行了大量研究，但是没有人对霍乱有真正了解。当时人们普遍认为霍乱通过空气传染，有意思的是尽管传播理论是错误的，但是人们却能够通过统计发病率和死亡率说明如何和到哪里去寻找病因。Jakob Henle在1840年认为霍乱由微生物引起，其理论也得到其他研究的支持，但是直到1865年Louis Pasteur的实验才建立了流行病学。Snow曾怀疑是水的质量问题，但是经过一些显微研究，他并没有发现霍乱微生物。

1.2 如果是我们处于那样的情况该怎么办？

Snow非常重视分析，他也是分析流行病学的先驱者之一。William Farr也是当时的一位流行病学者，他注意到霍乱死亡案例不仅源自27种不同类型，而且死亡者教区、年龄和职业也各不相同。

如果是我们，则可以开发一些包含所有变量的逻辑模型，看看是否可以支持或否定当时流行的理论。但是很难开发一种综合性的模型，因为不能直接测试传染和毒气假说。根据医生的说法，器官问题并不直接引起霍乱，而在传染过程中空气存在的霍乱扰动对生冷材料会产生作用。当时毒气学研究的重要成果可能源自虚假的关联，根据这种关联可以导出的模型显示，贫困是霍乱的最重要因素。

1.3 结局

Snow把每个死亡案例都标绘在地图上，发现第一，所有死亡案例与水泵的距离都很近；第二，死亡案例距离百老街水泵更近，第三，他从多个水泵处采集水样，百老街水泵的水看起来最干净。另外他还有两个“负面”支持数据：喝啤酒的工人聚集区没有死亡案例，有高墙的囚犯囚禁地没有死亡案例。

Snow假设的成功在于关注点集中，他的预测只有少量不符的案例。Snow亲自到现查调查预测不符案例，直到取得满意的解释。他坚持认为，霍乱使消化系统疾病，所有征兆都可以通过消化道长期失液解释。从而他得出结论，霍乱的感染源是口腔，而不是呼吸道。

对于数据科学家来说这一点非常重要，因为我们的结论必须与所有其他信息一致，包括模型内和模型外的信息。结论不仅要在统计意义和逻辑上成立，还必须观察结果一致。如果发现有与常识矛盾的地方，更常见的是你犯了错误，而不是你发现了新大陆。

Henry Whitehead试图通过调查否定Snow的结论，但是结果却与Snow的分析一致。喝了百老街水泵水的人，58%都得了霍乱，没有喝的人，只有7%得了霍乱。另一项工程调查发现，污水坑一直存在向百老街水泵轴漏水的现象。

1.4 商业应用

谷歌地图开通了数据与地理信息结合的很多可能性，从餐馆导引到海地大地震的能量释放。绝不能低估将信息显示到地图上带来的影响。这就是数据创新的力量：以创新的方式采集、使用和展示数据。

谷歌、微软和苹果都在紧盯快速增长着的空间信息市场。我们预计，随着基础设施能够支持随时随地的大数据采集、存储和分析，空间分析和数据可视化将会保持持续增长势头。

但是数据可视化只提供解决方案线索，并不能提供解决方案本身。拿到几乎一样的调查数据，甚至类似的地图，Board和Snow却得出完全不同的结论。为什么？因为Board通过传统方法进行分析。他们都是杰出学者，他们是把事实往模型里套，而不是修改模型以适应事实。Snow的成功在于他特别关注异常案例。人们往往轻视异常案例的重要性，往往不去深究出现这些异常的根源。我们往往责怪客户没有按我们模型预测的那样行事，而不是反思模型的局限性。

运用空间分析和数据可视化时，需要注意的是我们是在说服客户，而不是困惑客户。正如《纽约时报》一篇文章解释的那样，一位陆军团长在伊拉克战争时会花大部分时间制作PowerPoint幻灯片。（《纽约时报》：“我们遇到了敌人，他就是PowerPoint“）

# （11）服务化测试 PankHuri Mishra

**摘自《trends in software testing，软件测试发展趋势》**

**摘要：**服务化测试TaaS正在被软件工程界接受，大家认识到测试外包对于既要保证软件产品开发进度，又要保证在不影响产品质量的前提下降低成本是一个可行的选择。结果，少数公司已经开始提供自动化TaaS过程的工具。本文介绍这种新出现的概念，给出一般模型，解释TaaS所涉及的步骤，以及如何通过云技术实现TaaS。目的是通过有效利用资源提供测试服务。本文还提出了一种测试外包的计价模型。

**1 引言**

软件是执行特定操作的一组机器可读的指令。随着各种各样的设备进入人们的生活，人们比以往任何时候都更加依赖软件。从唤醒睡梦的闹钟，快速便捷的交通工具，提取现金的ATM，人际交流的智能手机，到用于娱乐的电视，几乎我们身边的所有一切都运行着软件。

软件在人们的生活中发挥着这样重要的作用，因此保证软件质量是至关重要的。测试是验证任何产品质量的过程。举一个纺织工业的例子：购买布匹时，我们期望布匹禁用，尺寸合适，不易掉色。在销售布匹之前，销售者有责任测试布匹的这些基本性质，换句话说，销售者要对布匹的质量负责。

软件测试是在把任何软件销售给客户之前，软件工程师必须确保其质量和正确性的过程。软件测试中的任何折扣都会导致巨大的资金损失，有时甚至造成人员伤亡。在苏格兰，一架Chinook直升机由于软件缺陷造成坠机，导致29名乘客全部丧生。阿丽亚娜5火箭本来要将卫星送入地球轨道，但是软件缺陷导致27秒钟的延迟，最终造成任务失败，3亿7千万美元打了水漂。美国标准与技术国家研究所的一项研究发现，软件缺陷每年造成美国经济595亿美元的损失。为了说明软件测试，举一个飞机票预订系统的例子。保证系统信息的正确性是系统拥有者的责任。

每个软件开发公司都会雇佣有特殊技能的测试人员进行产品测试。测试在软件生命周期中大约占25%。测试人员的责任是：

1. 建立测试环境。对于飞机票订票系统，需要笔记本、掌上机、屏幕尺寸不同的移动终端，确保用户界面适合所有设备。
2. 编写各种功能的预期行为和用于验证的测试用例。对于飞机票订票系统，要保证所有航班的信息，包括可供预订的座位，是正确的。测试计划应覆盖验证系统所提供所有功能的测试用例。
3. 一旦产品就绪，需要在各种设置和环境下进行测试。对于飞机票订票系统，需要测试其在各种设备设置上的可使用性、可扩展性和性能。

软件测试有不同类型，主要有：

1. 功能测试：测试系统的输出，验证其是否与需求规格说明描述的一致。
2. 集成测试：测试各个软件模块，验证集成后的组合功能。对于飞机票订票系统会有各种模块，比如用于显示航班信息的系统用户界面和用于存储航班信息的数据库。重要的是要验证系统显示的信息与存储在数据库中的信息一致。
3. 正常性测试：只测试基本功能，是功能测试的子集，在没有时间进行完整测试采用。对于飞机票订票系统，显示正确的航班和可订座位是主要功能。
4. 回归测试：模块修改后对整个产品进行测试。对于飞机票订票系统，重要的是保证增加的新航班信息或对模块的修改不会有系统信息不正确的风险。
5. 压力测试：在重负载下测试，确定在哪个点上系统的响应时间下降或崩溃。对于飞机票订票系统，在任何时刻，成千上万用户试图订票时，系统应以合理的响应时间处理用户请求。
6. 使用性测试：测试新用户是否能容易地使用软件。必要时产品应给出提示和帮助。飞机票订票系统应给出必要的帮助和警告信息。

传统测试模型面临一些挑战，例如：

1. 高成本：通常积累测试经验，购置或开发自动化工具，维护测试基础设施的成本很高，对于中小企业更是如此。但是测试资源并不是在整个软件产品生命周期都会使用。
2. 资源灵活性低：传统模型不能在需要的时候补充测试专家，不需要的时候遣散测试专家。软件公司在进行压力测试时可能需要特定专家，但是对其他测试类型可能不需要。
3. 高级测试环境支出高：高级工具、先进设备和平台需要经常更新，费用不低。
4. 缺少特殊测试专家：测试人员需要大量实践才能熟练使用各种测试工具和技术。比如大量接触安全领域的测试人员才具备最新安全威胁和可能的攻击知识。很难雇佣这样的测试人员完成数量很小但是非常重要的测试。
5. 难以完全实现测试自动化：通常对大型应用采用手工测试很容易出错。

有什么办法能够继续得到传统测试模型带来的好处，同时又把挑战转移给专门从事测试的公司呢？

比如布匹制造商可以关注制造，而把验证布匹质量的工作交给其他外部公司，共享设备和测试人员，以获得更大的投资回报。

TaaS是一种外包测试模型。一般来说，产品拥有者或服务消费者也完成一些需要代码知识的内部测试，而其他测试由外部专家完成。

**2 服务化测试**

服务化解决方案由于不需要基础设施投资并得到更好的回报而越来越受到欢迎。对于测试，高级测试工作需要经验、大量资源和昂贵的测试工具。按使用付费模型使软件公司能够在需要的时候得到所需要的。所以TaaS解决方案开始流行。首先需要了解TaaS提供测试服务需要什么。下面介绍实现TaaS所需的基础设施。

**2.1 TaaS体系结构**

TaaS是一种基于服务提供商和消费者交互的模型。服务消费者向提供商提出请求，提供商满足其请求。一般需要考虑用户界面友好性、数据安全性、服务可用性，最重要的还是费用。图1给出了TaaS体系结构。

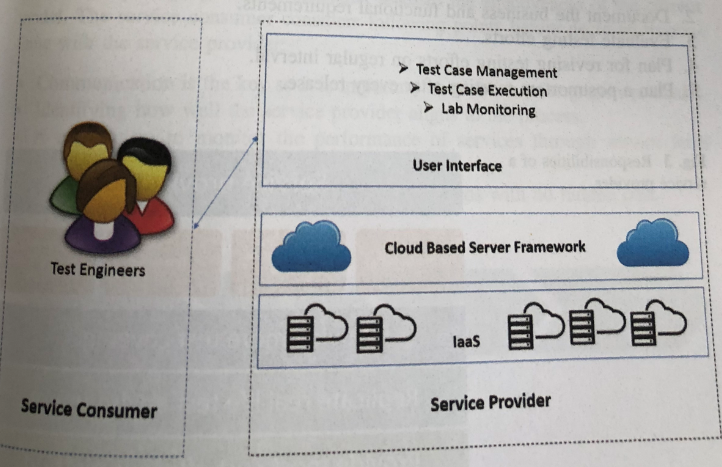


图1 TaaS体系结构

**2.1.1 服务提供商**

服务提供商向其他公司提供测试服务，主要服务包括测试咨询、策划、自动化、执行和管理。图2给出了服务提供商的职责（略）。

策划测试策略的主要考虑包括：

1. 质量：多个用户要频繁与应用或产品交互，对数据的使用性和正确性要求很高，所以服务质量是第一位的。服务提供商建立现实世界环境测试应用。对于任何TaaS提供商来说，质量都是一个关键考虑。
2. 效率：服务提供商通过集中化测试、工具优化、整体资源有效化来降低成本，更快测试，更早发现问题。
3. 收益：服务提供商关注积累内部经验，以提高收益，扩展测试领域。

图3给出了服务提供商的责任。



图3 服务提供商的责任

在策划上投入时间会得到长期收益，可避免日后出现服务提供商与客户之间的矛盾。在策划过程中，服务提供商需要考虑以下因素：

1. 确定产品可以运行的所有平台。
2. 编写业务和功能需求文档。
3. 评估测试工作量。
4. 策划定期修改测试工作量。
5. 策划每个发布后的总结会议。
6. 确定有效沟通机制。例如通过电子邮件沟通软件问题。

**2.1.2 服务消费者**

服务消费者使用外包模型获得服务，并相应地支付费用。自动化回归测试、安全性测试、性能测试、监视并测试基于云的应用、测试大型ERP软件等，都是最适合TaaS模型的，因为这些都需要经验和基础设施。TaaS也适合没有资源投入到测试技能并期望集中精力到产品开发的小型企业。

如果服务消费者转移到TaaS模型上，则需要建立定义完备的过程，确定适合TaaS的具体测试域，将开发团队与TaaS模型融合。因此，准备高层转移路线图对于获得TaaS模型的好处是很重要的。

消费者不能一步到位，而是必须策划一系列阶段，包括：

1. 确定当前测试模型，考虑是把整个测试还是把特定测试转移到TaaS上。
2. 根据测试需求和性价比选择服务提供商。
3. 通过较小的发布周期和试点启动转移。
4. 在发布周期之前明确定义测试需求。

图4给出了TaaS模型中的消费者角色（略）。

经过试点，消费者评估当前模型的有效性。服务消费者在决定继续使用服务提供商之前应考虑以下因素：

* 沟通是个关键因素，建立一个合作模型非常重要。
* 确定服务提供商遵守该过程的程度。
* 通过服务水平协议（SLA）监视服务质量。
* 支付机制适合消费者需要，没有阴性成本。

**2.1.3 服务提供商和消费者之间的交互**

服务消费者要与提供商共享测试需求，包括产品详细信息、所需的测试类型和时限。根据需求，服务提供商计算总成本并制定测试计划。根据报价，消费者与提供商就总成本进行协商。消费者也可以选择提供商提供的服务。合同签订后，提供商执行测试，并提交测试结果。图5给出了服务提供商和消费者之间的交互流（略）。

**2.2 TaaS的有利因素**

以下讨论对TaaS有利的一些因素：

1. 过去软件公司把测试看作是提高产品质量的一种投资，现在软件公司有选项降低测试成本，可以考虑使用外部专家执行测试。
2. TaaS框架有能力并行向多个客户提供测试服务，能够优化测试资源的使用。
3. 以前由于网络带宽的限制，软件公司倾向于一地产品开发和测试。近年来网络质量的迅速提高使不同地点的团队无缝协同成为可能。

表1给出了传统模型与TaaS之间的差别。

表1 传统模型与TaaS的比较

|  |  |  |
| --- | --- | --- |
|  | 传统 | TaaS |
| 支持的敏捷/瀑布等方法论 | 支持 | 支持 |
| 测试环境 | 手工 | 按需配置 |
| 测试数据 | 手工生成 | 动态提取 |
| 测试框架 | 手工+自动化 | 自动化 |
| 测试工具 | 采购 | 按需配置 |
| 测试文档 | 不要求 | 必备 |

TaaS使得服务消费者关注其核心强项，不必在测试上分心，不需要投入时间和人力创建和维护测试环境和资源。TaaS肯定是测试外包卖出的一大步。

**2.3 从传统测试到TaaS的转移**

从传统测试到TaaS转移也有风险。图6系统化地给出了转移流程（略）：

* 确定目标：服务消费者需要理解转移到TaaS的目标，可以是降低成本、得到专家服务等。
* 文档需求：服务消费者需要在列出需求上做大量工作，包括被测产品细节信息、支持的设备、性能考虑、安全性、扩展性考虑等。
* 服务提供商选择：根据需求选择在该测试领域有经验的服务提供商，研究服务提供商的过往记录，招标、评标。
* 质量检查：服务消费者需要定期就定义的要素评估服务提供商的工作质量，服务提供商根据评估发现的问题进行整改。最后消费者向提供商提出建设性的反馈意见。

**2.4 TaaS的局限性**

图7归纳了TaaS主要局限性（略）。

* 数据私密性：服务消费者需要提供私密数据和信息，可能存在泄露风险。
* 安全性：涉及国家安全或公司关键信息的数据。
* 停工：由于服务提供商是远程的，依赖互联网。需要考虑网络或硬件制约因素。

**3 TaaS基础设施**

通过构建一种框架获得从获取需求到提交测试结果的完整TaaS实现，包括测试计划生成、估价、测试执行、与消费者的交互。图8给出了一种TaaS框架。

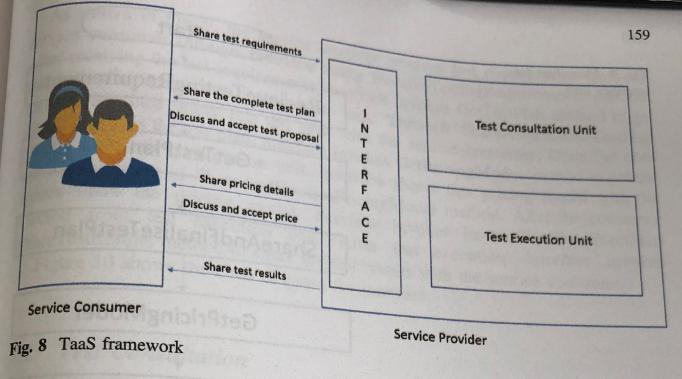


图8 TaaS框架

在图8中，服务提供商向服务消费者提供一个接口，通过接口消费者提出请求和测试需求。服务提供商内部处理该请求，并提供一个测试计划。服务消费者确认测试计划后，服务提供商给出报价。服务消费者同意报价后，服务提供商执行测试用例并提交测试结果。

**3.1 接口**

接口隐藏内部处理。如图9所示，服务提供商使用ReceiveTestingRequirements方法获取测试需求，然后接口调用GetTestPlan方法从测试咨询单元获取测试计划。通过ShareAndFinaliseTestPlan方法与服务消费者共享测试计划。服务消费者批准该测试计划后，接口调用GetPriceingModel方法从测试咨询单元获得报价。接口在ShareAndFinalisePricingModel方法中与服务消费者共享报价模型。消费者同意报价模型后，接口调用执行单元中的InstructTestCaseExecution方法。测试执行完，接口调用ShareTestResults方法，与服务消费者共享测试结果。

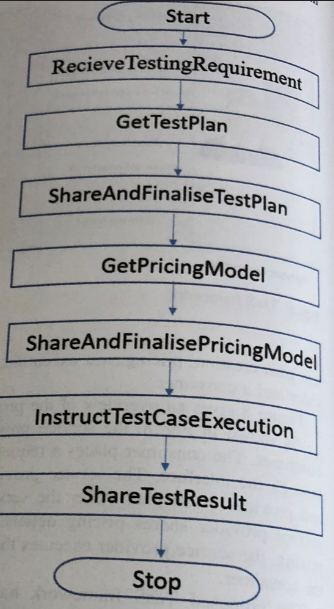


图9 接口单元的流图

图10给出的是接口的类图。

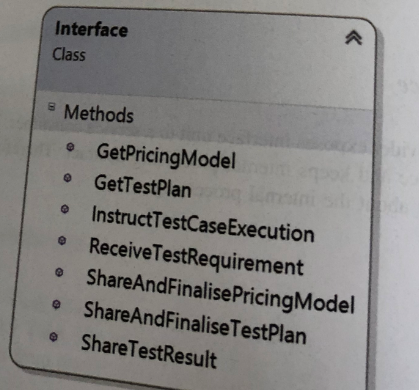


图10 接口的类图

**3.2 测试咨询**

测试咨询单元主要根据消费者的需求建立测试计划，并相应地产生报价。图11给出了测试咨询单元整体情况。

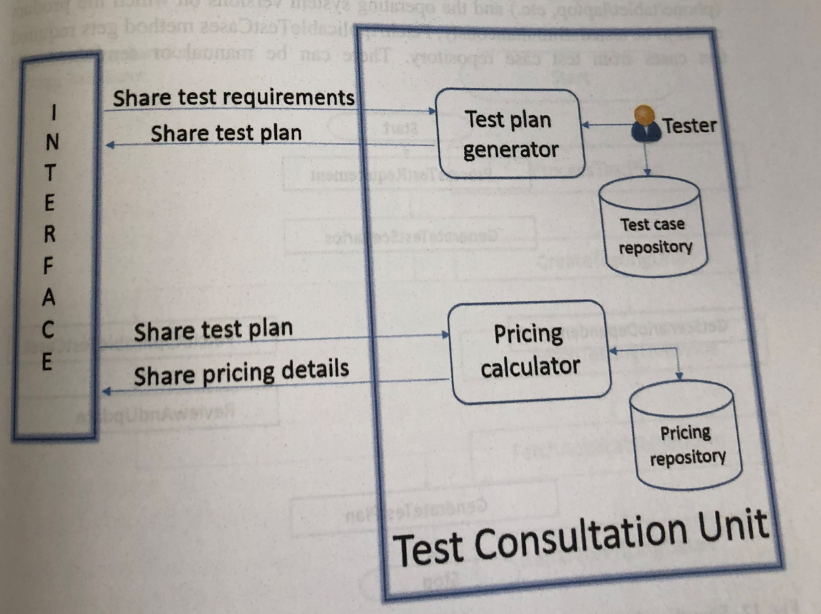


图11 测试咨询单元

图12给出了测试计划生成器的流图。图13给出了TestPlanGenerator类图。

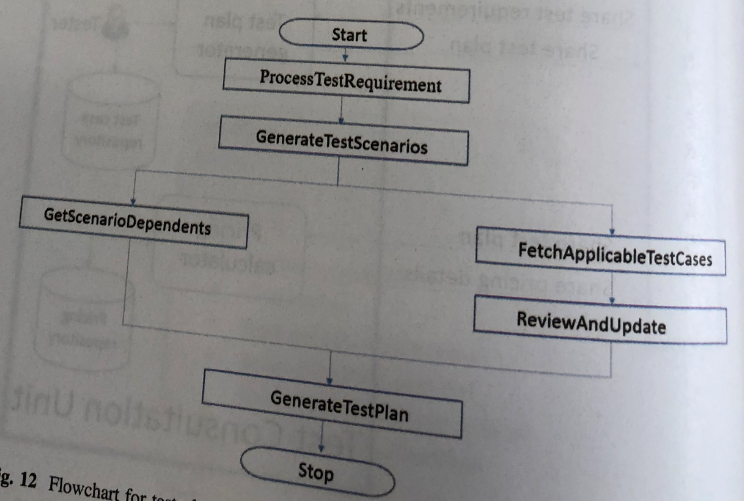


图12 测试计划生成器流图

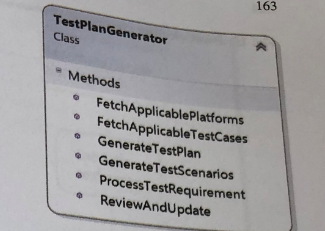


图13　TestPlanGenerator类图

图14给出了报价计算器的流图。图15给出了PriceCalculator类图。

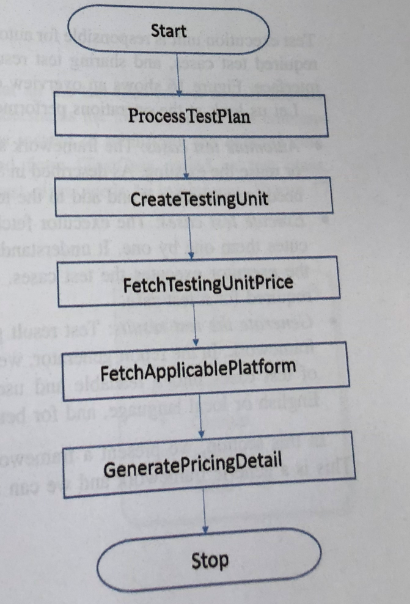


图14 报价计算器流图

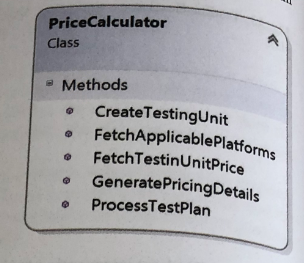


图15　PriceCalculator类图

**3.3 测试执行**

测试执行单元负责自动化或重用测试用例，执行所要求的测试用例，根据接口共享的测试计划共享测试结果。图16给出了测试执行的整体情况。

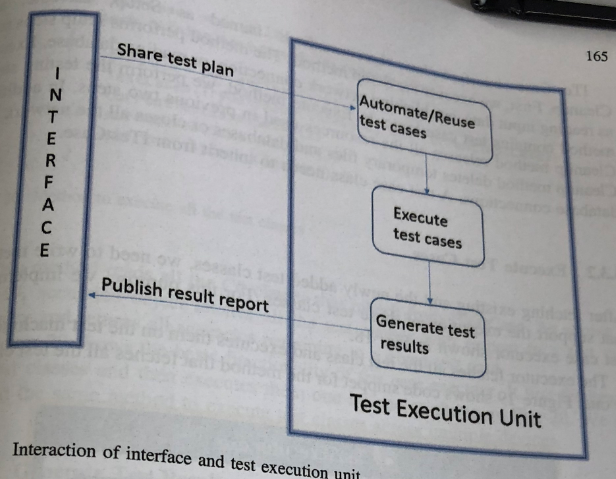


图16 测试执行单元与接口的交互

**测试执行单元执行以下操作：**

* 自动化测试用例：框架支持补充新的测试用例和修改或重用现有的测试用例。如果是增加测试用例，测试人员需要进行自动化，并补充到测试用例库中。
* 执行测试用例：执行器提取所有适合的测试用例，并逐个执行。执行器理解各种测试用例格式，负责测试用例所需的所有设置。
* 生成测试结果：测试结果生成是框架的最关键的部分。在报告生成器中，要把测试用例的执行结果转换成可读和用户友好的格式。需要使用简单语言，为了提高可读性，还需要插入表格和图形。

以下TaaS框架是采用C#实现的，用户也可以使用其他语言实现类似的解决方案。

**3.3.1 自动化测试用例**

为了自动化测试用例，要求所有测试用例都遵循某种定义完备的规则，通用框架根据规则理解文件。为了增加新的测试用例，测试人员需要定义一个类，继承自ITestCase。ITestCase描述测试用例需要实现的方法。图17给出了ITestCase。

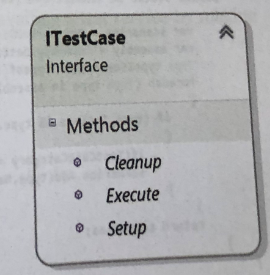


图17　ITestCase类

首先执行Setup方法，执行诸如输入数据、建立网络连接或读取数据库等设置操作。Execute方法包含测试用例逻辑，执行测试步骤。Cleanup方法释放上两步使用的所有资源，此外还删除临时文件和数据库，关闭所有网络和数据库连接。

**3.3.2 执行测试用例**

提取了现有和新增加的测试类后，需要编写支持执行这些测试类的方法。图18给出了测试用例执行器。

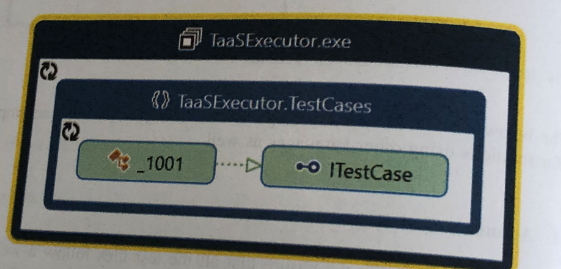


图18 测试用例执行器

这个执行器提取所有测试类并在测试机上逐个执行。图19给出了提取测试类的代码片段（略），其中的场景对象提取测试用例对应的所有测试类，并返回指定测试用例的所有场景。

图20给出了测试用例执行器的基本功能。可以扩展该方法以在多个设备上执行测试类。

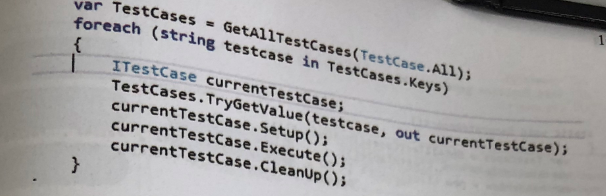


图20 执行所有测试类的方法

**3.3.3 生成测试结果**

测试结果包括执行了的测试用例数、输出、没有通过测试的错误细节、所用时间等。图21给出了生成测试结果的代码。



图21 用于生成测试结果的方法

**4 实现**

本节通过一个具体例子，即飞机票订票系统的测试，讨论TaaS框架的实现。

图25给出了几个测试需求。

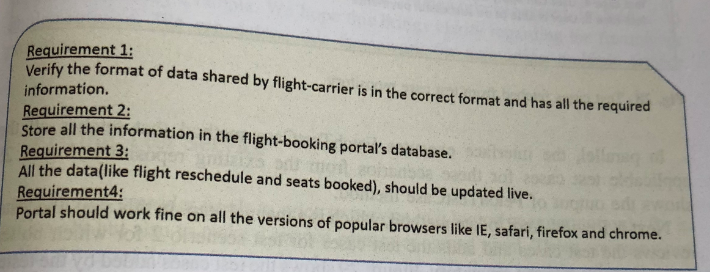


图25 测试需求

图26给出的是通过测试需求生成的测试场景。

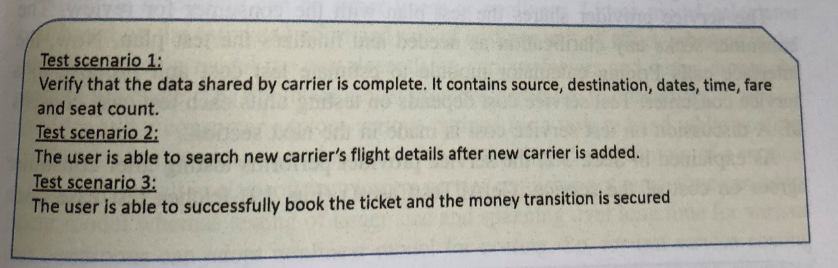


图26 测试场景

图27给出了根据测试需求和测试场景生成的依赖。

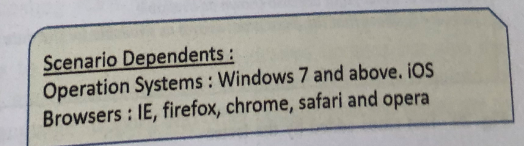


图27 场景依赖

图28给出了从现有测试用例库中提取的针对这些场景的所有适用测试用例。

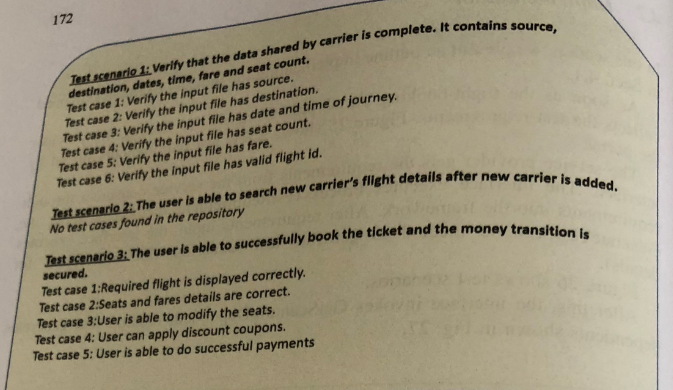


图28　从测试用例库中提取的测试用例

图29给出的是由测试人员补充的测试用例。

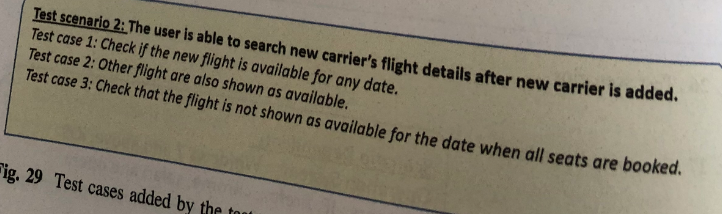


图29 由测试人员补充的测试用例

图30给出了在不同平台上执行不同测试场景的测试计划。

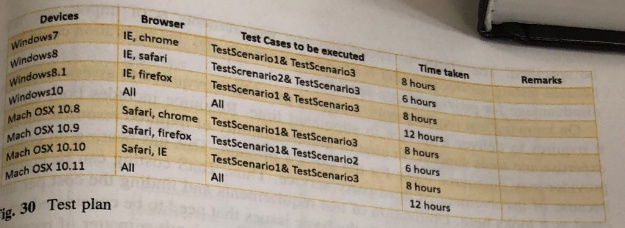


图30 测试计划

**5 测试服务报价（略）**

**6 云端TaaS**

NASSCOM报告预期全球软件测试外包市场将从2010年的300亿美元增长到2020年的500亿美元。测试服务需要寻求提高投资回报率的工程方法。云技术提供了一种资源共享方法，以下讨论针对TaaS利用云技术方面的问题。

**6.1 优点**

测试服务提供商需要多种硬件和软件平台提供测试服务，但是自己可能没有这样的资源。而且技术发展很快，维护多个版本的资源成本很高。云模型提供了一种简化解决方案，以管理资源使用成本，因为服务器、硬件和其他应用都部署在云端。用使用付费模式，可以根据实际需求，增加或减少所使用的基础设施。

**6.2云端实现**

图31给出了在云端上部署TaaS模块的示意图。

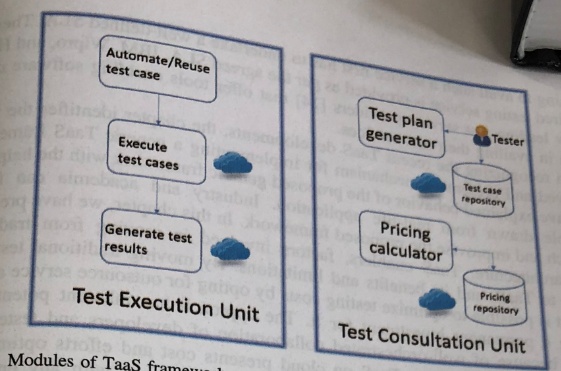
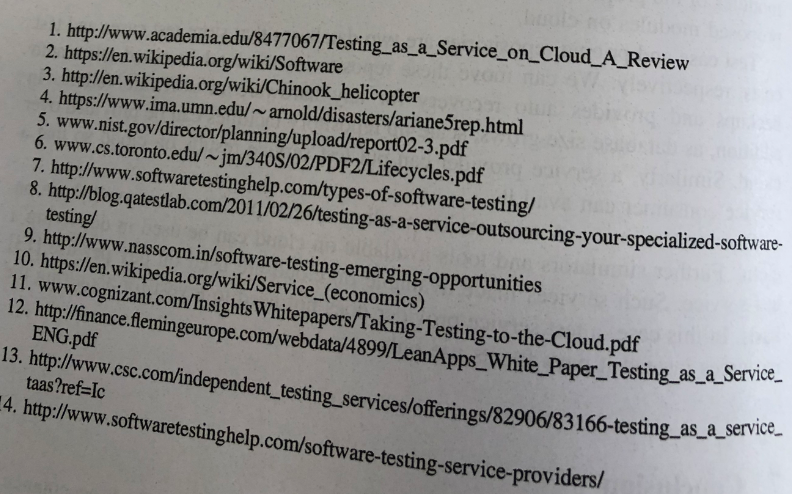


图31 云端上的TaaS框架模块

**7 结论（略）**

参考文献：



其中9，“软件测试出现的机会”，11，“将测试转移到云端”，14,“软件测试服务提供商”等文献可以补充阅读

# （12）理解测试债 Ganesh Samarthyam

**摘自《trends in software testing，软件测试发展趋势》**

**摘要：**技术债指团队有意或无意为了获得项目的短期收益而做出的技术决策。测试债是近年来新出现并引起软件界关注的概念。本文讨论测试债的组成要素和偿还测试债的策略，并通过案例研究讨论在项目中如何发现“测试发霉”及重构测试偿还债务。本文适合IT公司的经理和领导，以及相关研究者阅读。

**1 引言**

**1.1 技术债**

Ward Cunningham于1992年最早提出“技术债”，指为了短期提高开发速度对软件质量造成的长期影响，借用经济学上的术语说明短时间的获益如果不偿还将带来的长期伤害。

借贷就会产生财务债，如果不偿还，债务利息会不断增长，直至偿还不起导致经济破产。技术债也是如此，如果在开发过程中有意无意走捷径就会产生债务。如果通过重构补偿这种捷径，则债务不会引发问题，但如果任凭技术债持续积累而不采取偿还措施，则会使软件“技术破产”。

软件项目中的高技术债会在多个方面影响项目，整个公司都可能会在债务重压下寸步难行。如果不理会技术债，轻者不能及时对客户的问题作出响应，重者打乱公司的发展节奏，即变更成本随技术债的增加而增加，导致生产率下降。公司要在有限的时间内更关注功能，而不是质量，导致产生更多技术债，从而陷入恶性循环。技术债还会打击开发团队士气。

以下是一些产生技术债的常见因素：

* 进度压力
* 缺少高水平工程师
* 缺少文档
* 过程和标准运用不足
* 没有测试套件
* 补偿措施滞后
* 知识不足

技术债可以在需求、体系结构、设计、编码、测试、构建、文档、基础设施和版本控制等多个方面产生。图1给出了技术债的几个主要方面。通常如果代码债被高度重视就会产生技术债。最近设计债、体系结构债也开始引起重视。

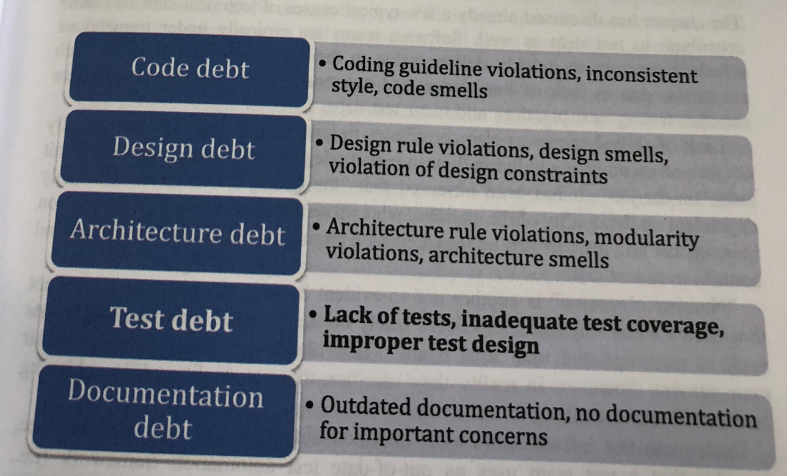


图1 几种维度不同的技术债

如果走了与测试有关的捷径就会出现测试债。比如如果完全没有做单元测试（对于老项目是很常见的）就会产生测试债。这是因为不完成单元测试是一种捷径，虽然可以加快开发进度，但是会对软件质量产生长远影响。

1.2 测试债的重要性

经常发现经验不足的公司欠下测试债，最终因为高企的测试债而不得不为生存挣扎。本来是想先人一步将产品推向市场，为此公司完全关注开发功能，但是到了测试阶段，开发人员只是快速地在自己的机器上测试了一下最常用的场景，而不是充分、综合的测试。通过史诗般的努力，团队可能会在相对很短的时间内成功地开发并交付能够运行的产品，甚至快速占领相当大的市场份额。不过最初的成功常常并不能持久。在开发第一个版本时欠下了大量测试债，比如没有创建并执行自动化测试，或执行的测试覆盖率很不理想。到了第二个版本时，团队首先必须集中力量通过在测试上大量投入来偿还欠下的测试债。任何延迟还债或没有重视还债都会导致产品的可信性下降，如果债务过重，还会导致开发难以推进，出现“技术破产”，放弃产品研发。

1.3 测试债的一般原因

软件团队一般会面临进度的巨大压力，需要接口拿出价值，团队常常因为没有测试债常识而牺牲测试最佳实践，并欠下测试债。

工程师经验不足也是产生测试债的一个原因。开发人员常常没有接受过单元测试的训练，大多只是“在岗”学习如何测试。而单元测试的质量严重依赖编写测试用例的工程师水平。水平不高的测试人员编写的难以维护的单元测试用例是产生测试债的一个因素。

“数字魔咒”是另一个产生测试债的不那么明显的常见因素，很多软件团队仅仅关注诸如测试覆盖率和总测试用例数这样的数字，关注让管理层满意，结果是拿到“数字”却牺牲了测试质量。

测试债的另一个原因是测试基础设施工具不足，例如使用过时的测试框架，使测试用例编写效率不高，难以升级。

其他因素还包括测试策划不充分、估计偏差过大等。

**2 测试债分类**

2.1 单元测试

单元测试有很多捷径会导致测试债：

* 单元测试不充分
* 单元测试不明确：单元测试是活着的文档，有助于理解被测代码的功能。如果单元测试数据不完整，会对对理解被测试的理解造成困难
* 带外部依赖的单元测试：单元测试应该独立执行，应该与外部依赖因素隔离。比如单元测试直接访问数据库而不是进行隔离这种依赖性，即使下层功能没有问题也可能由于外部原因，例如网络问题造成测试失败
* 单元测试中的断言不合适：很多情况下，断言的错误或非优化使用会导致测试债。例如陷于“数字魔咒”的团队为了提高覆盖率而不使用必要断言。类似的，断言条件重复使用对增加测试价值也没有意义。甚至有时给出的断言条件是错误的。有时测试条件需要人工检查，而没有写入断言语句，结果每次测试都需要人工干预。

2.2 探索式测试

探索式测试方法不依赖正式测试用例的定义。这种测试中，测试人员根据直觉和知识而不是正式的测试用例设计开展测试。开发团队发现探索式测试很有吸引力，因为其富有创造性、不受限制并且费效比高。探索式测试对于安全性测试等很有效果，因为这种测试需要测试工程师具有大量经验和创造性，以发现安全方面的脆弱性。

但是过度依赖探索式测试也会带来问题，部分因素包括：

* 随着应用的复杂度提高，探索式测试的成本和工作量也会增加，比结构化测试的难度也更大。
* 测试经理很难理解和监视测试工作进展。
* 重新执行测试既困难也开销大

完全依赖探索式测试会产生测试债，但如果探索式测试结合手工测试和自动化测试就会取得效益。如果做了这种结合，以下因素会产生测试债：

* 不准确的估计：由于更多的残留缺陷，基于探索式测试的测试结果评估可能导致额外的返工，直接影响维护成本
* 测试人员经验不足：探索式测试的主要优势源自测试人员的经验
* 文档差

此外，如果探索式测试的过往测试活动记录得不到很好维护，还会带来其他问题，例如工作量估计问题等。

2.3 手工测试

在手工测试中，测试工程师按照测试规格说明描述的步骤执行测试用例。以下是一些与手工测试有关的常见测试债：

* 有限的测试执行：通常可用的资源和时间对于完成完备、深入的测试都是不够的，所以只会执行测试用例的一个子集
* 不恰当的测试设计：手工测试是很费时费力的过程。测试用例可能设计为大量变化，包括不同的数据组合，需要测试人员执行所有这些组合。由于这非常耗费人力，测试人员只执行少量比较容易的场景
* 缺少测试评审：

2.4 自动化测试

自动化测试不需要手工介入，执行预先脚本化了的测试用例并检查执行结果。单是只依靠手工测试不做自动化测试本身就会产生测试债。自动化测试是有效回归测试的关键。以下是与自动化测试有关的测试债常见因素：

* 不恰当或不充分的基础设施：在与客户基础设施不同的基础设施上实施测试可能产生预期外的结果，从而降低对系统的信心。测试应能够使用习惯的硬件或设备。例如对于嵌入式系统，测试常常使用仿真器或模拟器。虽然可以简化测试，但是省去在实际硬件设备上的测试环节会产生测试债
* 没有编码标准：自动化测试需要遵循公共标准，如果不采用公共编码标准会增加测试代码的维护工作
* 未修改的测试用例：不对过时的测试用例修改会降低对测试用例的信心，降低测试质量，增加维护工作
* 运用记录回放：使用记录回放工具测试GUI应用是很容易的。但是使用这样的工具也有缺点，例如对UI的一点改变都会需要对测试更新。已经有比记录回放更好的替代工具

**3 测试债的管理**

在现实中出现测试债是不可避免的，对于管理测试债重要的是要跟踪测试债，并定期偿还，使其受控。关键是采用精心设计的实用方法评估和偿还测试债。

不过需要注意的是，有时出现测试债是好事，因为这使得团队能够达到其目标。例如，可能不能完全解决测试评审中发现的所有问题，因为必须满足交付时限。这样会产生测试债，但是只要已经制定了计划在未来的发布中解决测试评审中发现的问题，并且这种计划被恰当地跟踪和执行，那么这种测试债就是可接受的。

另外一点也很重要，就是要避免测试债的积累。做到避免要靠提高开发团队对测试债的认识。例如关注“清洁编码”实践，定期测试评审等。

3.1 偿还测试债的一般过程

管理测试债的关键是偿还测试债。通常软件项目会逐渐积累相当多的测试债，在这种情况下需要评估测试债的程度，并做出偿还计划。现实中不大可能只关注测试债，根据我们的经验需要采取以下步骤：

1. 量化测试债，取得管理层支持，执行大范围的偿还
2. 周期性地偿还技术债
3. 避免增加测试债

3.2 管理测试债的战略性方法

管理技术债只靠战术方法是不够的。战略性方法包括改造工作方式。

* + 1. 采用针对测试代码的有效编码实践
* 结对编程
* 清洁编程
* 在不增加或删除测试用例的情况下，使测试代码更易理解

这些因素相互影响和关联。

* + 1. 实行有效的测试实践
* 10分钟构建。对于减少修改开发人员机器上的代码以迁移到目标环境所需的时间，快速而完备的构建是很重要的。根据经验，构建、测试和部署软件（包括编译代码、运行测试用例、配置机器，如注册设置、拷贝文件等，启动软件）应该只需要10分钟，而很多情况这些工作需要几个小时。通过采用合适的实践（例如增量编译）把这些工作压缩到10分钟数量级是可行的。
* 无脚本测试自动化。传统GUI测试方法是使用记录回放工具，虽然上手很快但是很难维护。通过编写脚本的GUI测试自动化工作量大，技能要求高，而且维护也比较困难。新出现的方法是无脚本的测试自动化，通过GUI由对象和行动生成测试用例，能够更好地适应GUI的修改、技术演化，能适应后期修改，相对比较容易维护。

**4 案例研究（略）**

**5 未来方向**

* “测试臭味”这个概念已广泛用于描述测试债。测试臭味有三种不同的类型：代码臭味，即存在于测试代码中的臭味；行为臭味，即测试用例执行时对测试结果有不良影响；项目臭味，即整个项目的健康状况。单元测试中的代码臭味已经引起普遍重视，其他两种臭味尚未引起业界重视。
* 主流集成化开发环境包括Visual Studio、Eclipse和支持自动化代码重构的IntelliJ UDEA。虽然已经出现了检查测试臭味的工具（例如TestLint），但是还没有支持自动化改造测试臭味的工具或IDE。
* 已经有技术债量化工具，包括针对编码债和设计债的，例如SonarQube，但是还没有针对测试债的工具。
* 测试债是向管理层说明测试过程走捷径成本的最佳元素，因此测试成熟度评估框架和方法可以纳入测试债因素。

# （13）软件测试大数据的思考（一）

软件测试积累了大量数据，直觉上，应该有可能把这些数据存在的规律、知识、模式发掘出来，反过来指导软件测试工程实践。这已经成为业界共识，而且不少人也从多年前就开始了相关研究工作，建立了不少测试样例库、故障库，甚至还有更全的库。但是到底从这些数据中发现了哪些有意思（我们以前没有意识到的）的知识、模式、规律，这些课题组总是闪烁其辞，我的理解，就是没有值得一提的东西。我见到的所谓大数据应用结果，大多都是些宏观的分类统计，确切地说，连决策支持（DSS）的水平都算不上，与人们对大数据应用的期望相差甚远，对具体的新的软件测试项目设计的作用不大。

我想出现这种状况一定是有地方出了问题，我想尝试梳理一下。

首先，软件测试大数据应用必须提出工程应用需求，第二，工程需求抽象、转换为科学问题，第三，通过科学工具，包括数学工具，建模工具等解决科学问题。

比如机器人、通信协议的测试，可以把应用抽象为米莉机，就是输入、输出和状态都是有限的自动机，且任意时刻只处于一种状态中，状态的迁移只取决于当前状态和输入。另一些应用的测试则可以抽象为摩尔机，即状态转换只依赖于输入。接下来，通过算法寻找米莉机的同步序列或归位序列，作为测试用例，验证系统可以进入的最终状态，或是其他预期行为。

类似地，我们期望通过软件测试数据的处理得到什么呢？我听得最多的就是测试用例推荐，当然这也是其他领域大数据应用的目前热点之一，互联网上已经有很多了，主要是各种商品和服务的针对性推荐。

那我们不妨想一想，我们自己学习软件测试的过程，是不是仅仅观察很多以前的测试用例或是软件问题报告呢？显然不是。对于我们这种基于需求测试模式为主的测评实验室来说，从需求映射为测试用例的技巧才是学习的关键，也是评价一个测试员能力水平的重要方面。好的测试员为什么优秀？主要体现在对具体需求的理解，能够结合自己的经验、常识甚至直觉，提出满足测试目标的合理的测试用例。当然，不同的测试级别对知识的要求层次差别还是很大的。根据一般的测试模型，测试级别越高，测试目标越具体。比如级别低的单元测试，只需要理解要完成的具体功能，与具体的应用没有很大的关系，因而比较容易抽象出通用的测试经验，甚至自动化。相反，系统测试强烈依赖特定的软件应用场景，类似的软件用在不同的场景中，会要求差别很大的测试设计。

# （13）软件测试大数据的思考（二）

因此我认为更为抽象的单元测试对于测试用例推荐，一般来说更容易。但是单元测试主要还是由软件研制单位完成，因此单元测试的用例推荐对软件测评实验室价值不高。对于大数据应用来说，单元测试用例推荐可能是比较好的起点，关注服务对象主要是软件研发单位。

同时，软件单元测试与其他测试级别相比也是最成熟的，目前唯一有效的软件测试国际标准就是单元测试标准，针对单元测试的自动化、半自动化工具也是最丰富的，在软件测试目前已有的四种主要方法中，包括基于规则的动态分析、基于测试数据输入的动态测试、基于符号运算的运行时检查、基于模型的软件逻辑正确性验证等，也是运用最成功的，因此大数据要想给出一些更具启发性的推荐也是比较困难的，但是如果大数据能够更深入地结合被测软件单元的结构信息，换句话说，大数据不仅要使用从单元测试数据中提取的模式信息，还要使用具体被测软件的结构或设计信息，推荐有价值测试用例的可能性更大。特别地，针对某些单元测试难度比较大的子问题，比如中断处理、变量定义/使用路径验证等的测试，获得突破的可能性更大一些。总之，面向的问题域越宽，给出高价值测试用例的可能性越低。我想大数据的发展策略应该是从特殊到一般。以往软件测试大数据应用效果不佳，我觉得很可能与开发策略有关。

不少人对我说，软件测试大数据应用效果不佳的主要原因是数据不够多，暗示只要测试数据足够多大数据应用的效果必然改善，尽管谁也说不清多少数据才算足够。我对这个想法深表怀疑。举个例子，大数据也好、数据库知识发现也好、人工智能、机器学习、模式识别等也好，当取得了一些重要突破后，无一例外地都把应用目标转向股市和类似的领域。直觉上这些技术应该在股市上有所作为，因为股市数据量不可谓不大，数据中的模式不可谓不多，已有的传统分析方法运用也几乎到了极致。但是事实是，即使是红得发紫的阿尔法狗也在中国股市潜水多时后黯然离开。从地球奔向月球是美好愿望，但是必须得有合适方法，你对自行车再挖空心思地改造，也永远不会把你带到月球。我想大数据也是一样。如果软件测试领域的复杂度可以与股市比较，再对着自行车费心思就大可不必了。关键是降低问题域的宽度和难度，大而化之的所谓软件测试大数据应用，我认为目前是不可行的。

# （13）软件测试大数据的思考（三）

软件测试的主要工作，是把软件需求映射为软件测试。大数据应用也可以表述为辅助测试人员，直至自动完成这个映射，大数据能力的表征也体现在对测试人员的辅助、启发效果。如果给出的推荐是测试人员没有想到，而且对测试具有重要性，则得正分；如果推荐是重复性的，比如被其他测试用例包含，则得零分；如果推荐没有可行性或必要性，甚至对测试人员的设计产生干扰，则得负分。这种奖励/惩罚机制可以构成大数据自我学习、进化的基础。

这个机制与测试人员的成长类似。测试大纲写好后经过内部评审和外部评审，经过几轮迭代不断完善。在理想情况下，测试人员会汲取教训和经验，同类问题出现概率越来越小。把这个测试设计进化过程送给大数据，即同一个项目的多个测试设计，以版本高低顺序排序输入给大数据，呈现出完整进化过程，可以是大数据的一个起点。因为测试设计在评审中被提出修改意见的落实结果，本质上就是一种推荐，而且是正分推荐。

不过问题远没有这么简单。测试设计评审，实际上也是评审同行背对背地进行从测试需求到测试设计的映射，并把自己的映射与被评审的测试设计产品进行比对，发现产品不足的过程，仅仅了解测试设计改进情况，而不知道为什么需要进行这些改进，很难获得实质性的经验。因此大数据了解测试需求等依据，理解测试设计多个版本的映射差异，对于大数据的模式发现来说可能是不可避免的。

于是下面的问题就变成大数据如何对测试需求进行理解。我听到不少人说，测试需求的理解通过自然语言处理解决，因为现实世界的测试需求主要以自然语言描述的形式给出。实际上，对测试需求的理解本身对测试人员的能力要求就是很高的，很少有测试人员仅仅根据测试需求文档就能很好理解被测软件的，与软件研发人员的沟通是很难避免的。这意味着大数据还要理解不同阶段的软件设计文档、甚至系统设计文档、项目技术方案、立项论证报告等等相关文件。即使这些条件都具备，通用的测试需求大数据理解也是非常困难的，可能降低难度的方向，应该是限定软件类型、使用领域等，达到一般测试人员的平均理解水平还是有可能的。

在测试需求大数据理解上，我觉得还有两个问题很难回避，一个是需求文档中的图表，一个是预期的测试环境初步想定。

图表对于测试人员对测试需求的理解至关重要，大数据直接理解图表中给出的丰富语义信息、结构信息、拓扑信息，我想还是有很大困难的。如果需要人工把图表进行自然语言转换，即使是能够做到，对大数据的实际使用也会是个很大制约。如果这个问题没有一个足够满意的解决方案，我想大数据对于软件测试可能弄不好就是那个自行车了。

测试需求到测试设计的映射还有一个重要条件，就是测试环境的初步想定，而测试环境又是在映射过程中逐步确定的。一般来说，测试需求决定测试环境，测试环境影响测试需求。但是测试环境的确定还受到很多技术、管理、成本、进度、可用资源等多方面的制约，可能需要比较复杂的综合考虑，既要考虑测试的充分性、有效性，还要考虑测试工程的可行性。而且测试环境的确定常常是一个逐步迭代的过程，测试任务的不同，使得通常不存在一个具体的理想目标测试环境模型。而如果要把测试环境作为测试设计大数据推荐的一部分，可能难度很大，因为测试大纲不同版本本身可能就包含测试环境的演化。可能的解决办法是优先确定测试环境，虽然这对测试人员来说也不是完全不能接受，但毕竟是对大数据的一个重要制约。

# （14）动态语言程序中的开放对象的自动分析

由于动态语言，例如JavaScript中的对象是开放的，因此对其进行静态分析具有很大难度。开放对象的属性（又称为字段、实例变量等）是不确定的，而且是可迭代的，开发人员可以通过编程增加、删除和修改已有对象的属性。由于具有这种灵活性，开放对象使得动态语言使用者能够通过对象操作过程建立框架，从而压缩代码规模，提高代码重用度，改进程序的灵活性和可扩展性。

以下说一段JavaScript代码。这段代码有条件地为对象r增加来自对象s的属性，像这样的代码片段可以在实现类和特征系统的框架中以各种形式重复：

for （var p in s）

if （p in c）r【p】=“conflict”

else r【p】= s【p】；

由于对象r、s和c中的具体属性是未知的，所以到代码结束时我们不能完全确定对象r的结构，但是可以通过初始r、s和c的结构推断出r中的所有属性集合中的每个属性一定是以下三种之一：

第一，如果这个属性在s和c中都存在，那么这个属性的值就是“conflict”；

第二，如果这个属性在s中有，但是在c中没有，那么这个属性的值就等于s中的对应属性值；

第三，如果这个属性在s中没有，那么r中的这个属性值就保持不变。

因此，导出这种划分是解决开放对象属性分析的关键。当对象属性不能静态得出时都会出现这种开放对象属性分析问题。不幸的是，开放对象抽象问题极大地增加了静态分析的难度，因为对象不再是具有固定的属性集合，而是具有数量不固定的属性。因此，对象的抽象必须不仅要抽象属性点的取值，而且还需要对属性本身进行抽象。这种抽象必须能够把多个属性合并到一种单一的抽象属性。这种开放对象抽象问题不能使用像Java这样闭合对象语言的抽象方法进行简单修改。

解决这个问题的办法之一，是通过“开放对象堆（HOO）”抽象，在没有具体属性知识的情况下，精确地表示和推导开放对象操作程序的特性。构建一种集合的关系抽象域，以推导对象属性的划分。这种方法的一个实现已经可以验证大量使用开放对象的动态语言框架的描述，证明了这种方法的有效性。

下一步的工作是扩展这种方法，以便能够推导属性之间及其对应取值的关系。这种关系能够精确确定最终哪个值被确定化，并证明具体取值的特性，即使其给出的只是最终状态。

# （15）多人决策过程中的人际欺骗：通信模式分析的新认识

人际欺骗是两人或多人之间一种复杂、动态的交互，欺骗者故意传递信息以使信息接受者产生虚假信念或结论。与其他人际信息交换形式一样，人际欺骗也是一种传递、接收、再根据对信息的反应更新消息的迭代过程。在人际欺骗过程中，欺骗者试图主导整个信息交换过程以达到目的，同时避免被识破，会施展很多不同的策略，比如迷惑、欲言又止、绕圈子，或采取更自信、更具说服力的方法战胜对方。反过来，潜在被欺骗的一方如果有怀疑，也会调整自己的策略以发现真相，例如试探更多信息，或不动声色地观察怀疑对象但保持被动姿态以隐蔽自己的意图。由于人际沟通是一种不断调整的动态过程，欺骗者和被欺骗者之间的通信会有很多不同形式，并会随着欺骗的推进不断演化。

通信的这种重策略、动态调整特点为分析带来很大挑战。对交互的开始部分直接进行聚类度量和分析很有可能对欺骗交互的走向产生误判，欺骗的开始部分往往和后面发生的没有多少关系。当面对多人欺骗通信时，这种分析难度更大，因为在网络环境下消息交换会呈指数增长。欺骗者会对被欺骗者一对一或一对多地发送消息，而被欺骗者有可能反馈，也可能不反馈，可以被动潜水，也可以充当消息发送者的角色。这种复杂性也能够说明目前关于多人交互欺骗的知识为什么如此稀缺。

通过商品化通用模式发现和分析工具THEME试图发现欺骗通信中的规律。本研究基于以下特定多人通信场景：欺骗者和被欺骗者的角色是给定的，通过分析工具到底是否能够得到人际欺骗通信的新认识，以期推动对团队沟通、自闭症儿童行为、家庭内部冲突等方面的研究。

当前的研究，一个欺骗者试图欺骗两组行骗对象，一组对骗子有怀疑，一组对骗子保持天真，这样一个实验床供THEME发现训练观察者确定的非语言行为数据中的模式。这样的模式发现分析可以确保提供研究多人欺骗通信策略和动态特性的全新角度。

通过理论方法确定、采样、模式搜索参数设定、独立因素控制、自报告和非语言行为观察等环节，通过THEME发现的模式有：行骗对话通常持续25分钟，以前许多实验室研究的结论只有几分钟；虽然发现了很多模式，但是跨组的模式具有很强的趋同性，不同对话的欺骗模式重复率高达88.7%，说明欺骗的高度结构化、重复性，特别是对话开始和结束部分，但是中间部分体现出高度交互性，且行骗对话参与者往往多于两人。

兴趣点1:欺骗者的主动策略。根据IDT（人际欺骗理论）逻辑解释行骗个体的策略行为，骗子至少采用两种策略行为。最常见的是“蒙骗”，即有选择性地屏蔽一部分消息，使受害者对虚假可能性感兴趣，以达到行骗目的。通过模式分析可以证实：模式化的交互和鼓动反馈倾向。

第二种策略更具主动性，主导多人交互，通过主动传播虚假前提或结论引导集体走入歧途。通过模式分析可以证实：行骗者更喜欢交互，尤其是与主动发言相关的行为。

在多人通信中，骗子采用主动行为和主导通信行为更为突出。

兴趣点2:受害者的特点。根据IDT逻辑，当个体发现被骗，往往会掩盖，或最低限度地承认被骗。与骗子一样，受害者的行为也有不同的模式。一种是不那么主动，仔细观察其他成员的行为，不主动发起交互以识别骗子。第二种是采取试探行动，与其他成员发起对话以发现骗子。在网络环境中，更倾向于采取试探行为。

下一步的研究

IDT指出欺骗行为是动态的，意味着行为及其模式是随时间变化的，交互早期的模式不会保持不变的。这样行为模式本身就变成变化的主体。有的模式后来会消失，又会出现新的模式。欺骗者关注的是其最终目标，可能会改变沟通模式的惯用特征，与讲真话的模式具有很大趋同性，而讲真话的模式具有更好的稳定性，随时间的变化较小。THEME通过分析对话的某个转折点前是否有更多模式出现来解决这个问题，还有可能发现内部相关性是加强还是减弱，模式分布是否相对稳定，在会话的开始或结束时模式是否会更集中。THEME也可以回答更多策略或非策略行为是否使用在会话的开始和结尾。这些都有助于多人通信中的模式复杂性，理解模式的模式。

# （16）安全关键软件的重用与系统集成的形式化方法

重用和集成是大多数当代计算机科学和技术的基础。重用和集成以互补的方式支持软件和系统的低成本设计，并且带来高性能。低成本源自通过直接重用将已有知识资本化，而高性能源于通过转换和集成重用对已有知识的改进。重用和集成也是一对互补概念，重用考虑组件的创建和开发，而集成关注可重用组件如何交互以得到可重用的集成体。因此，集成是一种重用分析和新模式综合过程。而形式化方法则恰好能够提供合适的场景，以严格的方式处理重用和集成。形式化语言能够无歧义地进行模型描述，而形式化验证技术又可以支持重用和集成机制的核查。

在这个背景下，形式化方法被第一次用于在开发过程的早期阶段证实或证伪所设计组件的安全性属性，后期则用于帮助确保组装后组件的一致性。模型检查是一种最常用的验证技术，能够很好地兼顾检查的可靠性和使用复杂性。在用抽象化的布迟自动机表达的线性时序属性，可采用基于规格说明的测试。测试准则可度量语义相关性，还可以度量已有测试用例相对规格说明中需求的验证质量。因此可以根据所要求的准则自动生成测试用例。对于“交互组合片段”的规格说明和模型检查，有一种完整的建模和分析过程。首先定义“组合片段”的符号语义，然后展开形式化分析过程，包括完成验证和确认两个任务。这种方法的一个优点是可以解释对形式化分析的实际反馈。案例导引抽象细化方法可用来压缩验证工作量。复杂系统动力学中的安全部件是验证过程的考虑因素。这种方法的优点是不要求在每个细化步骤后都重新进行可达性分析，而是把细化潜入到可达性分析过程中，以避免对同一模型行为进行重复检查。一种更强的形式化方法用于光学系统的成像属性验证的理论证明，该属性要说明组合后的系统继承了同样的线性代数特性，而单一光学系统的特性已经得到验证。

时序特性对于软件和系统设计来说至关重要，因此需要特殊的形式化手段进行规格说明描述和验证。多服务器范例能够在同一个规格说明中处理多种语义，这种方法的目的处理复杂系统的形式化问题，为此，扩展了“时间petri网”以支持与使能逻辑相关的两种不同语义的表示，即时间语义和门限语义。根据规格说明需要，这两种语义能够用petri网解释。这种方法能够非常紧凑地表示模型语义，大量压缩了状态定义所需要的数据。时间petri网还可以用作标记语义，以描述“交互时序总图”和“时序图”。为了验证这类软件的模型正确性需要使用时态特性。首先由设计人员采用对象约束语言（OCL）描述，然后再被转换为时序计算树逻辑（TCTL）公式，最后由导出的时间petri网通过模型检查来评估。这些都需要分布式和并发描述来解决实时软件建模和形式化。基于过程的语言BPEL配合分布式实时系统模型定义的操作语义，提供了真正的并发语义以处理语言描述中的时序行为。这样，通过模型检查就能够验证基于时变值的时序逻辑性质了。

重用和集成对于数据仓库和数据挖掘领域也是关键概念，其构成信息恰当存储的基本框架，以便于信息抽取，提高信息的可用性。一般来说，信息提取依赖于用户需求，但是选择构建系统的恰当组件却是很难的任务，特别是需要考虑非功能需求时。为了解决这个问题，借助本体方法，提出一种把用户需求分解为低层需求的方法。满足功能需求的组件集合被检查是否满足非功能需求，以进行筛选。这种模型集成了必要的信息，包括用户需求和解决方案之间的跟踪链，以便提高系统的进化能力和可维护性。有关数据仓库的另一个问题涉及异构数据源的数据集成问题。依靠主流方法可以解决，包括抽取、转换和装载，从异构数据源形成一个完整的数据存储。提出了一种树状层次结构数据或信息的集成方法。开发主流方法过程具有很高的复杂性，而新方法能够压缩映射和转换数据所需要的时间和工作量。该方法自动处理并简化所有可能的数据转换，包括分级自引用和平面表示等数据转换问题。

基于重用提取知识的能力近年来开始引起人们的关注。基于链的分类搜索可提高引用的准确性。一项实验研究比较了潜在和非潜在方法，并根据实验观察提出了新的方法，即以新方式利用潜在技术，将两种潜在方法与一种现有的非潜在方法结合起来，将潜在特性方法和潜在链方法进行了统一表示。结果说明，非潜在方法通常具有最好的性能，但是这种混合方法在网络非常稀疏的情况下能够提供更高的准确性。此外，人们也对知识组件为基础的规格说明和推理支持越来越感兴趣：如何维护无冲突知识，而新加入的知识有可能与现有知识矛盾。提出了一种实用和现实的计算方法解决在任何时间段内的知识矛盾问题。

这12篇高质量的论文作者都是重用与集成形式化方面的专家，是从132篇大会录用论文精选出来的，可较好反映出这一领域的最新研究成果。

# （17）关于软件重用的几点思考

重用对于软件开发和测试都具有普遍意义，每个软件人员自觉或不自觉地都在进行大量重用工作。不管是编码还是测试，不管是写文档还是写幻灯片，甚至起草短短的各类评审意见，不使用剪切-粘贴操作的几乎没有。

毫无疑问，重用给软件人员带来极大便利，不光是节省时间，更通过继承过去的经验而使新的工作产品质量得到保证。今天的软件人员不仅占有大量的各类模板，大量编码规则和标准条款，还会大量使用搜索工具，真正敲入的文字或代码比例正在越来越少，大部分代码或文档都已经在过去的项目中被多次使用、并经过反复验证。我们在实际软件测试项目工程中，经常听到人家说某段代码已经定型，没有修改过或只改过很少接口，不需要或只需要少量测试就可以了。即使是全新的代码，其中也有很大一部分来自其他类似软件项目，包括编码范例、文档描述、容错处理方法，甚至评审方法和技术。

然而给软引用由于不恰当的剪切件人员带来巨大便利的背后是悄悄的巨大风险。据统计，软件缺陷的-粘贴的竟占到70%。如果说给软件质量带来好处的因素总分是800，那么软件重用占到的比例会高达350；反过来，如果影响软件质量的因素总分是1200分，软件重用会占到650分，虽然都是权重第一，但是如果搞不好，给软件质量的消极影响往往会更大一些。阿丽亚娜5发射失败就是一个惨痛教训。欧空局实施的过程管理不可谓不严，完成的软件测试不可谓不充分，层层通过的软件评审和评估不可谓不系统，但就是一个对型号4的软件重用没有充分考虑硬件变化带来的影响最终酿成大祸。在系统总结教训时，有这样的一条：软件问题出现的部位，是所有领域专家都认为是最保险、最不可能出错的部位，处理方式的正确性已经在领域专家中形成定式，没有人提出任何质疑、反思。结论是，重要评审仅仅依靠领域专家是不够的，恰恰非领域专家有可能提出更多在领域专家看来可能有些可笑、但却是通过转换思考角度才能发现的微妙问题。

在现实中，剪切-粘贴过来的东西往往还要做些替换，往往还与其他部分存在某种不协调、不一致而需要调整，往往还有一些可有可无或多余的部分需要删除。甚至是很重要的外部评审幻灯片，即使并不长，细细看来，发现剪切-粘贴带来的错误往往并不困难，尽管上会前这些幻灯片都经过反复审阅。

那么问题的关键就是如何控制软件重用的质量。控制的关键又在于识别软件重用引入问题的机理、途径，从而找到解决问题的有效方法。

我认为，软件重用引入问题的关键是由于重用的便利性使得软件人员忽略了必要的分析工作，多多少少有思维定式，加上一些盲目自信和乐观，从而不知不觉欠下技术债。这种债迟早是要偿还的，只不过是偿还形式不同而已。

具体来说，软件重用有两个只要步骤：重用和集成。重用指面向未来的使用而特意进行的专门部件开发。比如模板，就是针对以后的多次使用而进行的开发。集成是利用可重用部件合成完整工作产品的过程。而重用和集成都可能引入软件问题。

首先，可重用部件本身就可能存在问题，即使经过测试、经过实际使用，仍然不能保证验证的充分性，更何况新的使用场景并不一定与老场景完全一致。据统计，面向重用的可重用部件与非面向重用的部件开发成本相比要高出4倍，才能满足重用的质量要求，因为一旦重用部件存在缺陷检测起来更加困难、更难处理，因此要尽可能零缺陷。比如如果模板本身存在问题，重用后的影响就会放大很多。其次，很多时候被重用部件并没有考虑某些重用场景，某些特性并不适合无条件重用，有时又必须补充必要的特性。比如测试文档模板可能没有考虑部分工作外包的特殊性等。

至于集成，引入质量问题的可能性就更大了，涉及到可重用组件筛选、匹配，必要的修改，必要的验证等。其中组件筛选和匹配至关重要。组件功能静态描述和行为动态描述是实施筛选的前提条件。自然语言很难承担这个任务。基于模型的形式化描述看来是可行的选择。首先模型能够准确地描述组件最核心、最本质的内容，从而保证筛选的大方向不会有大的问题。既然模型是对组件的某种程度抽象，因此不可能表示出所有细节，特别是一些容错处理的细节，而这些细节对于组件重用又可能是至关重要的。比如某电信交换机组件，可以比较方便地使用米莉有限状态机描述。但是有些容错处理方式却很难通过一个统一模型表述。比如交换机主叫用户，回铃音收到8次转入告知无人接听状态。但是如果收到3次回铃音后不再收到信号怎么办？可以转到超时等待状态再转无人接听状态。如果超时门限未到，第4个回铃音收到，那么是不是该继续回铃音计数状态？如果第5个回铃音又迟迟未到，超时是继续累积还是重新计算？极端情况是不是迫使主叫等待过长时间？

这样的细节建模会非常繁琐。测试用例包可能是一种替代方法，通过预期输入与预期与设计、需求约束的符合性，提供筛选依据。

我认为，即使采用了比较完备的形式化描述方法，必要的人工分析都是必不可少的，必须建立与技术手段和重用风险相匹配的管理过程和活动规范，比如必要的评审、分析、验证等。

另外要形成软件重用和集成的规范，强化面向重用的组件开发，强化集成的分析验证。开发好的可重用组件，更安全地集成新产品

# （18）软件评审工具及工作原理

C语言智能代码审查（ICICLE）是最早出现的软件评审工具，可支持代码评审，并辅助评审者完成个人准备和集体会议工作。ICICLE的同步通信能力支持集体会议。传统代码评审采用手工形成文档，因而费时费力，还可能产生不一致的记录。

ICICLE具有以下优点：

1、发现常规错误，使代码评审者能够解放出来，集中精力检查对需求、规格说明和设计的实现正确性；

2、提供各种形式的有关被审查代码的知识，包括领域和环境知识，提供各种分析信息，比如交叉引用信息；

3、使评审者能够很容易地在一个窗口汇总有关源代码，不必逐个拷贝多个不同的文件；

4、通过共享窗口界面使代码评审者无纸化地完成各自职责。

ICICLE涵盖评审过程的两个阶段，即个人评审和集体会议评审。会议评审大家集中在一起，每人一台计算机。每位评审者都可以对任一代码行输入意见。研究发现，这种计算机支持的会议形式会对代码评审会议的推进产生很大影响，比如把代码缺陷统计记录交给会议助理，会议主持人则引导评审者将关注点集中到源代码中感兴趣的部位。评审团队要对评审记录集体认可。代码作者应参加代码评审会并回答问题。任何其他评审者都可以参加并共享会议讨论。

工具采用多窗口界面处理文档，中心显示窗口显示实际源代码，并为每行代码加上编号、标记以及反馈意见。这种结构使评审者能够同时审查多个部分的代码。工具的管理功能可以控制评论窗口的意见状态。

在准备阶段，评审者独立对任一行代码输入个人意见，对于每行代码可以设置没有意见、待查、需会议讨论等状态。个人可以对自己的意见进行编辑，也可以被评审团队采纳、修改、删除。还有一种标记表示缺陷、警告、威胁、违反规则。此外工具还具有交叉引用和超文本形式的知识浏览功能。前者可以提供对象信息，比如变量信息，后者辅助评审者获取领域相关知识。

在会议阶段，主要提供三个功能：共享窗口显示、提出意见和关注点导向。

利用第一个功能，主持人把所有评审者的关注点集中到要分析的模块上。在会议进行过程中，主持人可以同步锁定所有窗口。当主持人滚动、变换不同文件显示窗口时，全体人员的窗口也同步自动操作。不过主持人也可以通过分屏功能，使评审者能够既有会议统一同步窗口，也有个人操作窗口。

当评审者提出意见，所有评审者界面都会弹出意见文本框，并表决对意见的赞同和反对。最后可以把所有评审者各自评审意见汇总输出。

工具使任何评审者都可以通过移动光标，在其他评审者的屏幕上显示一个标记，以在其他评审者的屏幕上指点需要关注的内容。

会议结束后工具可生成缺陷统计报告。

类似软件评审工具还有Scrutiny、协同软件审查（CSI）、InspeQ、CSRS、需求跟踪工具（RADIX）、软件工作产品异步审查器（AISA）、Web审查（WiP）、InspectA、HyperCode、同/异步软件审查支持工具（ASSIST）、细粒度软件审查工具（CodeSurfer）、CORD、基于代理的软件工具（Agent-Based software tool）、基于互联网的审查系统（IBIS）、VisionQuest等，可查阅各自研发者的网站。

# （19）装备试验需求的变化对软件测试的影响与对策

随着装备试验工作的改革，包括装备软件测试在内的需求势必会同步发生变化。从技术方面看大致可以归纳出以下几个趋势：

一、性能试验、应用试验和在役考核为核心的新的试验划分，比改革前更强调完整覆盖整个生命周期，通过试验总案的编制要求，更强调各类试验的环环相扣、层层推进和密切关联，从关注点看，重心从单一指标验证向面向具体使命任务能力验证转移，即使是指标也更突出摸清底线、边界，以及复杂电磁环境等复杂使用条件下的验证。对于软件测试来说，将对测试条件设计提出更高要求，以便在尽可能接近使用条件下测试软件。对软件测评团队来说，不仅要能够深刻理解指标要求本身，还需要深刻理解指标和能力之间联系，和使用之间相互联系，以更准确地测试和评价被测软件。另一方面，测试团队还应具备低成本设计搭建测试环境的能力，包括外部激励源、软件行为数据采集器定义、实现和验证能力。

第二，由于各类试验的综合化，凸显软件测试自身特点和价值的要求势必更高。我理解，软件测试设计要更体现实验室特点，体现环境可控，试验高度可重复的特点，体现覆盖指标可度量的特点（过去强调结构覆盖，未来将更强调场景覆盖），体现环境可控渐变，以观察软件响应的变化趋势，以达到摸清底数和边界的要求。我估计，随着试验总案体制的执行，软件测试实验室有可能需要有能力定义、实现和验证软件测试环境的较强技术力量，需要有较强的技术开发和创新能力。这需要软件测试实验室有好的技术资产积累、开发平台和经验积累，当然也为推出独立的测试开发综合平台提供了空间。

第三，软件密集型装备的应用试验和在役考核依我看也很可能需要软件测试实验室的技术支撑，包括试验回放分析、故障和缺陷分析、充分性分析等。目前针对软件密集型装备应用试验特别是在役考核的试验评估手段还相对落后，这也可能是软件测试实验室能力发展的一个方向。

第四，试验总案的实行，意味着从研制到使用大跨度、多试验的数据集成。综合数据采集和分析能力将成为软件测评实验室核心能力之一。其中最重要的恐怕是数据模型的建立能力。我想软件测试实验室未来录用新员工时提出R语言要求也是很有可能的事。

第五，试验设施共享共用战略的资金投入和政策执行共用的推进，会促进软件测试实验室向更细分工发展，实验室专业化分工合作会更为普遍。

第六，接近论证单位、技术总体单位的软件测评实验室，优势会更为明显，因为其对使命任务、使用场景理解上的便利、接近最终用户的独特优势会进一步放大。基地是唯一全部参与三项试验的机构，基地软件综合验证能力可能会对整个装备软件测试细分行业产生引领作用。

# （20）行为与交互中的时间与自相似结构：从序列到对称与分形

模式概念正被广泛使用，以至于多数现代数学家都把数学看作是关于模式的科学。但是如果不进一步说明模式的类型就去搜索模式是没有多少意义的。即使是最简单的模式，有时在描述非常离散、看起来很复杂的自然和数学现象时，也会有一定的意义。例如著名的数学序列—斐波那契序列，即下一项的值是前两项值的和。

由具体类型模式组成的T系统，经过扩展和补充，形成T模式，可用作发现、分析和描述行为和交互中的实时结构的工具和组件。采用THEME软件，通过发现和分析T模式，发现和分析重复模式。

为什么采用分层模式发现方法？

T模式实际上已经超出最初考虑的人际交互场景，例如对大脑神经系统中的多神经元触发网络的研究和DNA分析。这与包括大脑在内的自然和生理结构有关。

现代科学说明，宇宙中从最小的到最大的结构，主要是数量相对较小的推拉力量之间的平衡结果。从小如亚原子粒子，到大如宇宙星系，尽管尺寸、复杂性差别极大，但是各个层次上的元素都可以看作是这种无处不在的推拉力量的作用结果。因此，不能简单地聚类，而是需要平衡这种相互对立的力量，形成层次结构化的聚类，就像已知宇宙复杂性分层，以及模式的模式的自相似模式一样。

要理解世界，发现模式的重复是最重要的。科学探索自然中的重复模式，做为模式科学的数学，想象和数学描述现实世界往往并不明显的重复模式。通过计算机，可以开放搜索算法探索难以想象的整个人类历史。显然，对行为数据的分析必须以对行为结构的理解为基础，因而限制了对行为中现实存在的结构做出假设，此外还需要了解可以得到的数据和可用的分析工具。

当前可以自动采集大量数据，从而产生出一个新的研究领域，即社会物理学。我们同样也不能忽视对小数据的统计分析，小数据反映的只是少量类型事件的少量发生。采用T模式模型和算法，可以对小数据进行有意义的量化和结构化分析，比如只持续几分钟的复杂交互，其中的模式可能只重复两次，也会是其可以凸显发现出来的。

以下是采用T模式方法，通过分析观众的反应模式理解电影艺术。

8分钟的短片《父亲与女儿》描述这样一个故事：一天，父亲不知何故离开了他的小女儿，独自划船走了。女儿骑着自行车一次次地来到父亲离去的水岸边。她显然渴望着父亲的回来，但是父亲始终没有回来。短片中的女儿逐渐变老，场景也相应变化。最后，女儿终于变成一个老妇，蹒跚地从水岸一步步走向水中，结束自己的生命。这时短片切换到梦境：海水变成了草地，草地上躺着父亲的空船。她走到船中躺下。当她站起时又看到了父亲，她在向父亲奔跑过去的同时又变回原来的小姑娘。

短片可以看作是由一系列事件组成，分为吸引和推开两类。吸引类事件包括：冲突、消解、渴望、相遇、效果和变化等，推开类事件包括幽默、符号化、对位法等。

邀请32名观众观看短片，16名观众身上装有传感器以记录其心率、皮压等数据。32名观众面前安装探头，记录每个人的表情。

没有人怀疑媒体具有吸引观众的潜能，但是以前却对观众对媒体的激励如何反应知之甚少。故事情节对于产生观众情绪具有重要作用，因此通过大量故事讲述技巧和结构化情节调动观众情绪，以吸引观众。对于虚构故事，构思及其与现实世界的关联以一种比复杂现实世界更易于捕获和理解的方式呈现。因此为了理解电影艺术就必须理解情绪调动技巧以及电影放映时现场观众的相应情绪变化过程。

通过具有T模式发现功能的THEME软件研究观众对《父亲与女儿》这部只有音乐没有一句对白的短片观看时的情绪变化，重点关注观众的生理反应和面部表情变化与电影情节展现之间的联系。

《父亲与女儿》设置了两种冲突，核心冲突是父亲的离去与女儿的内心矛盾。随着情节的推进，观众逐渐明白女儿不能从失去父亲中解脱出来。电影设置了反映核心冲突的三个情节：一，影片开始时的父亲离去；二，影片结尾时的女儿自杀，三，女儿躺在船上平静地睡去。影片也给出三个显著冲突消解情节。一度女儿似乎可以得到解脱，包括和男友一起到岸边，坐在男友自行车的后座上。另一处是现实到梦境的切换，梦境中一起都成为可能。最后是女儿在船上躺下，观众会认为女儿死了，但是这时她却站了起来并与父亲重逢。

女儿一次次地骑车来到父亲离去的岸边，对着空旷的海面张望，然后再骑车返回。虽然场景不同，但是失望却是不变的。这种不变情节的反复呈现暗示父亲的离去影响了女儿的一生，而女儿的解脱尝试却是徒劳的。女儿对水面的张望表现出渴望，而骑车返回则表现出徒劳。影片展现出女儿在骑车路上会遇到其他骑车人，每次相遇各自的年龄变化揭示时间跨度是女儿的一生。

影片通过吸引性事件吸引观众进入故事情节，又通过推开类事件给观众留出想象空间，从而使观众得到精神满足。

# （21）737MAX事故引发的思考

波音新机型737 MAX接连发生故障现象相似的空难，很快在网上出现了比较深入的飞机失效原因分析。尽管透露出来的分析可能还有这样那样的不足，可能还会还带来这样那样质疑，不过做为软件测试行业人，还是应该引发一些思考。

思考一：MAX存在的致命问题，通过比如性能试验、应用试验或在役考核能发现吗？通过现在我们通行的软件测试实践能发现吗？

我的观点，这个问题本来是应该在试验中发现的，而最有可能发现的机会应该在性能试验中的集成或系统试验阶段，特别是在极限、复杂、极端条件科目下的试验。相对来说，应用试验发现的可能性要小很多，毕竟这个问题的激活是个极小概率的事件。这个问题虽然源自系统设计，但是这个设计却是几乎完全是由软件定义的，因此性能试验中的软件测试是应该发现这个问题，并预测到问题可能带来的严重后果。

其次，目前我们通行的以软件需求规格说明为主导的软件测试发现这个问题的可能性很小。因为这种方式的软件测试重点在设计功能的验证，很少考虑功能的适合性和可达性。而MAX的这个问题恰恰不是功能实现正确性问题，而是功能在特定条件下的适合性问题，以及补救功能的可达性问题。

思考二：如果要通过软件测试发现这个问题，现行的测试设计需要做哪些考虑？

我认为，危险状态措施有效性的验证设计是决定性的：既然新机型发动机尺寸增大，重心后移，易出现机头扬起。为此设计的救命功能，即强制压机头扬起设计，以及对抗这个救命功能的解除功能是否在可以预见到任何条件下都是适合的，是不是存在不适合的条件？这样的条件发生概率是不是可以忽略？行业现实是，软件工程以软件任务书为主要启动点，不大与系统工程的反向沟通不足，系统工程与软件工程割裂比较严重。对应到软件测试上，就是对软件需求的过度信任，不大重视使用背景下的功能适合性验证。装备试验领域改革剑指的三个主要问题：脱离应用、强调功能忽视能力、底数不清，本质上都与功能适合性验证有直接关系。目前行业通行的软件测试类型中，与装备使命任务直接关联的就是安全性测试了，其他测试类型与使命任务的关联事实上都被软件需求或任务书阻断了，客观上与改革方向渐行渐远。

结论：MAX问题由软件测试发现的可能性最大，理应由软件测试发现。软件测试当前通行实践发现这种与任务而不是功能强相关的软件问题的可能性很低。

思考三：提高软件测试发现MAX的可能性，软件测试需要从哪些方面提升？

第一，具备理解软件功能与设备，乃至分系统、系统使命任务和使用场景的关系的能力，以设计出带有恰当使用背景下的软件重要功能，特别是安全性等用户关注的软件特性验证。比如本例：一旦空速主传感器失效，启动机头压低自动控制程序在什么情况下会进入危险状态？关闭自动驾驶启用人工驾驶模式后，5秒后强制恢复自动控制，这种设计是否足够可靠避免进入危险状态？在最坏情况下，彻底关闭自动驾驶功能的功能是否足够方便启动？最坏情况下的功能延迟，在飞机的什么状态下，比如速度、仰角等条件下，会产生什么严重后果？

第二，需要独立的验证基础设施建设，在本例中包括环境条件的数学模型、典型驾驶员操作行为和反应过程基本模型。改革对试验系统的总体目标包括：面向实战，全程覆盖，独立公正，高度权威。除第二点外，其他所有都与软件测试基础设施建设密切相关。基础设施不是单指机房、测试工具、计算机或网络设施，更重要的是有大量成套数学模型、基础比对基准数据等。这个是行业目前最缺的。没有这些基础设施，就谈不到测试场景想定的适合性，更谈不上高度权威。特别想强调“独立公正”的认识。目前行业只强调人员独立、经费独立，我认为最重要的还是技术独立，不仅包括测试方法的设计，还包括测试环境的独立。而测试环境往往又对测试方法的运用产生很强的制约作用。现实是，目前的软件测试过于依赖研制方的环境，测试技术独立性大打折扣。我觉得这不仅是实验室资源投入的问题，更是发展建设理念问题，否则不会十多年过去，我所提到意义上的基础设施，几乎看不到哪家实验室有多少实质上的推进。

第三，专业化分工。积累离不开专业细分，工程化的前提是专业化和过程重复。东打一枪西打一枪，技术上不太可能得到系统化的发展。改革对条件技术的总体设计思想是要达到地域分布、逻辑一体，建设理念要从型号任务牵引向以试验能力建设牵引为主转变。这个建设思路推进的必然结果就是更明确的专业化分工。如果实验室不能适应这个发展要求，就很有可能处于不利地位。当然，市场因素会起很大作用，我只是说实验室需要为可能的改革做好准备，做好脱颖而出的准备。

第四，行业协作。分工的结果必然是合作。改革提出的统一台账、军民融合、通用设施共用等的前景，我觉得虽然美好，但是实现起来真不会容易。总而言之这是一个发展趋势。不能低估决策者的决心。我想，先走一步，小范围、少方面的合作，可能会得到支持，也会得到实惠。不过首先你得有别人愿意与你合作的本钱。所以说回来，还是需要自己闷头提高自己的能力才是根本，不过需要注意的还是扬长避短，用足用好自己独特优势。

思考四：软件测试发展主要任务

改革意味着围绕目标的任务再分配，也意味着利益再分配。出现了新的玩家，明确新的路线图。软件测试要想成为改革的赢家，或至少不让盘中的蛋糕缩小，那么首先就得显示出对改革的新价值。新价值在于对改革目标的理解，更在于对目标确确实实、清晰可见的贡献。确确实实意味着实验室要善于总结，清晰可见意味着善于宣传。

新价值不仅要体现在绝对值上，更体现在与兄弟玩家的相对值上。竞争是不可避免的。软件测试的优势在于实验室试验优势，在于测试过程高度可设计、可控制、可评价，在于高的复杂场景覆盖，在于有破坏性的极限验证。改革规划的组织控制、测试测量、环境构设、分析评估和基础保障体系建设规划，距离软件测试行业差距还是很大的。软件测试是否能够发挥更大的作用，又与条件建设密切相关。蛋糕总体就那么大，软件测试怎样争取到更大的份额，主要还是取决于决策者对软件测试在改革中的贡献预期。不管是五年建设规划也好，据此制定的年度计划也好，都离不开软件测试现在展现出的现实价值和潜在预期。

随着装备信息化程度的不断提高，软件测试的作用也理应提高，软件测试理应肩负更大的担当，没有担当的提高不可能有地位的凭空提高。

更新日期：2019年03月16日