星云测试系列产品

白皮书

（2020 版）

www.teststars.cc

目录

上篇--星云精准测试

下篇--Wings 单元测试自动编码引擎

上篇

星云精准测试

（2020 版白皮书）

第一章 精准测试诞生的背景

20 年前（2000 年），上网是一件很酷的事，叫做“网上冲浪”，主要是几个门户网站占据绝大多数注意力；20 年后（2020 年），我们已经全方位“浸泡”在软件的海洋里，很多大型软件系统已经超过亿行，架构模型越来越复杂。未来的 20 年（2020-2040 年），随着各种智能应用的层出不穷，软件系统内部逻辑会不可逆转的越来越复杂，而外部操作会越来越“傻瓜”。用户一个眼神或者一动脑，就能够切换出不同的功能需求。同时中国的软件产品化进程，近些年来正在蓬勃的进行，大量的公司开始研发自己的产品。伴随着关键核心软件的国产替代，使得原本应用级难度的开发正在向高复杂度和高质量的产品型研发转型。这个转型的过程，注定将产生对于高端测试工具的巨大需求。

软件缺陷发现得越晚，其处理费用即呈几何性激增，而不是线性增长。如何从庞大复杂的系统中迅速及时地找到故障所在，一直是行业的一大难点。无法对大型软件进行有效的质量把控，就无法真正构建与维护大型软件。目前国内软件测试基本处于两种状态：一是绝大多数企业采用功能（黑盒）测试，二是小部分对软件产品有高可靠性要求的软件，企业会使用代码级的白盒测试工具。但这两种传统测试办法在目前的软件日趋智能化复杂化下，弊端越来越明显。

功能（黑盒）测试技术，使墨菲定律在所难免。由于无法获知程序内部逻辑结构，这种测试办法对软件可靠性要求不高的应用来讲问题不是很大，但是对于大型金融机构、智能工业、航天军工等关键系统，就意味着时刻携带隐形的巨大风险。为此，功能测试后期需要极高的人力投入才能完成复杂逻辑的用例分析和设计。然而对于黑盒测试来说，程序越大，杀虫剂效应越明显。而行业内当作银弹的自动化测试，当自动化程序本身规模扩大以后，它的维护本身就存在了很严重的问题。低效的黑盒测试状态，最终将影响和制约未来整体软件发展。

代码级（白盒）测试工具一般重点应用在研发阶段的单元测试上，满足了客户的部分高可靠性需求，但由于其价格高昂、技术老化，仅适合于小规模迭代瀑布式开发的软件，无法完成复杂的系统级别的测试以及分布式基于云的测试，更不适应敏捷迭代的开发模式。另外白盒测试工具基本都是国外产品，通常这些产品无法完成深度的定制化功能实现以及快速的用户响应，代码安全更是一个较大的问题。

所有先进的科技都是具备前瞻性眼光的。随着国内军民各项大型核心软件系统的上马，研发一种面向高复杂度大型软件、自主可控的高性能智能精准测试平台的必要性，逐渐凸显。在这个时代背景下，2012 年初，星云测试团队开始了筚路蓝缕、心无旁骛的研发征程。精准测试是个交叉学科，里面涉及到编译器、测试分析、图形技术、高性能通信与存储，软件的研发等多项底层技术。

经历无数个不眠之夜对技术难点突破的煎熬与最佳解决方案的反复推敲，星云精准测试产品在诸多方面率先实现了重大技术创新，成功突破了白盒测试使用难度大、价格高昂的桎梏，有效消弭了国外高端测试产品垄断的壁垒。星云精准测试产品更偏向于软件测试业界的“灰盒测试”，即用简单的黑盒操作办法，可以同时得到单元级和系统级的精准测试数据。

“星云精准测试”不负众望，在众多性能上大幅超越国外进口高端白盒测试工具产品，并在数据追溯、覆盖率可视化、智能回归、智能缺陷定位、分布式数据穿透与追踪等特性上有突出贡献。星云精准测试，既继承了传统功能测试前期的高效率运行区间，又能在后期通过系统的数据，让开发、测试充分协同，完成全程高效的测试与数据分析。

（1）将测试团队的价值放大，能够将开发与测试更加紧密的连接起来，互为支撑。

（2）采用精准、可信的测试技术，测试管理的难度大幅度降低。

（3）降低企业对人员的过度依赖，通过系统适应人员的变更。

（4）为企业实现测试数据资产的有效积累和分析，打下坚实基础。

“星云精准测试 VIP 大企业离线版云平台”在整体测试功能上的优异特性，成功获得了一批重要大型企业的高度认可及产品采购。

星云测试的首发版本为：穿线测试 ThreadingTest，2014 年 6 月 6 日上线，侧重于系统级白盒测试技术，测试用例和代码逻辑的双向追溯技术，测试示波器技术，覆盖率可视化技术。

2015 年 8 月 6 日，“穿线测试”正式更名为“星云测试”。在继承穿线测试整体技术上，星云精准测试增强了回归测试用例的自动选取技术，缺陷最后执行时序分析、智能缺陷定位、敏捷环境下多版本白盒测试数据的聚合、聚类分析、结合代码结构与动态数据的测试漏洞检出、代码安全特性，全面的测试管理特性等几十种优秀功能。

目前有“星云精准测试 VIP 大企业离线版云平台”、“星云精准测试 PASS 在线云平台 www.teststars.cc”、“全自动测试用例驱动生成系统 Wings”等多种工具产品。

星云精准测试旗下产品平台有 Horn、Paw、Shell、Wings 等系列产品。适用语言和平台暂为：

星云精准测试使用项目条件表					
产品	星云精准测试Horn版本	适用简介	主要支持JAVA语言类项目		
语言	JAVA1.6、JAVA1.7、JAVA1.8、JAVA1.9				
支持被测系统平台	windows系列	linux系列	UNIX（AIX）	Android	
应用场景	后台系统、网站、PC桌面应用、android应用等				
产品	星云精准测试Shell版本	适用简介	主要支持C/C++语言类项目		
语言	C89/C99/C++11/C++14/C++17等				
支持被测系统平台	windows系列 (VisualStudio2015-2019)	linux系列(GCC/QT)	UNIX（AIX）	Android底层	Vxworks(6.0以上)
支持体系架构 (*32-64Bit)	x86系列/Atom*	ARM*	MIPS	powerPC	
应用场景	后台系统、PC桌面应用、嵌入式系统应用、Android底层通讯/驱动等				

为响应广大用户的需求，目前正在进一步扩展适应的语言和平台覆盖面。

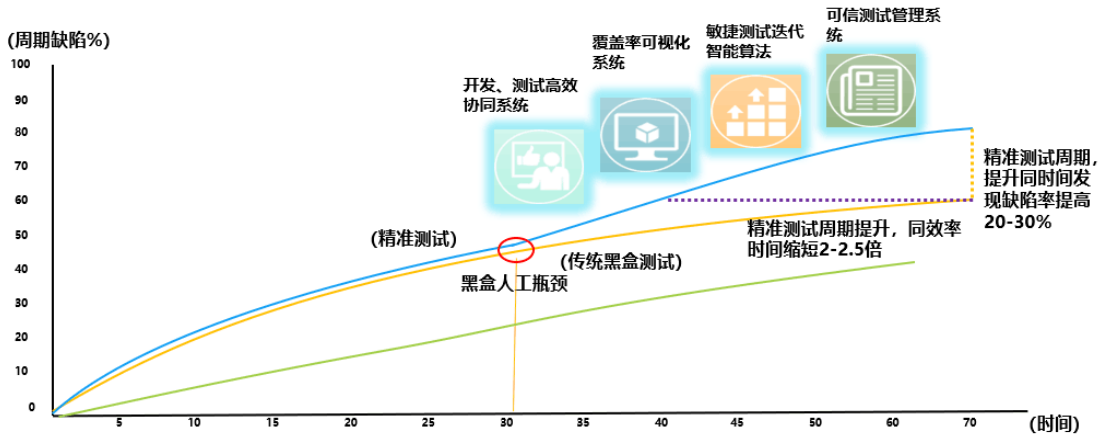


图 1-1 精准测试在大型系统的效率运行分析

星云精准测试，既保证了传统功能测试前期的高效率运行区间，又能在后期通过系统的数据，让开发、测试充分协同，完成全程高效的自动化精准测试。

第二章 精准测试技术体系对软件测试技术的升级解析

精准测试技术体系经过近 6 年的商业项目实践日臻成熟，大量商业应用案例的正面积极反馈，表明它已经成为新的软件测试技术方向体系。星云精准测试灵巧详细的功能设计和高度可靠的产品内核得到了市场的广泛认可，对于国际上原有的白盒测试工具来讲，是一种质的飞跃。

2012 年，精准测试的商用工具正式进入设计和研发阶段。与引进国外技术不一样，精准测试的核心技术体系是星云测试团队在国内的原创设计和研发。产品经过 3 年的悉心研发、反复内测、不断优化后，2014 年精准测试在 CSTQB 会议上，发布了第一个商用版本 ThreadingTest，引起了很大反响。6 年前，行业上还全面沉浸在黑盒测试的大氛围中，从未见过精准测试的超前的运行方法和提供的强大智能的功能，由于理念与设计过于超前，早期有一些业内质疑的声音。但由于产品独到的优秀特点和巨大的潜在价值，诸多伯乐开始试用，给了产品在实际场景中不断优化、强化的宝贵机会。经过不同领域、不同客户的各种高复杂度大型系统的百般锤炼、验证后，自 2016 年开始，精准测试开

始赢得全行业的重视和响应。

优秀的商用落地性是精准测试技术流行的一个重要因素。由于精准测试把准了黑盒测试无法解决的难题和固有瓶颈，踏准了测试技术发展大的进程，并从设计一开始就特意注重了商业场景下使用要求复杂性，因此它从诞生之日起就不仅仅是一套理论，而是有着非常强的实用性基因和可扩展架构，多场景应用成果卓然。

精准测试最底层的核心技术：“一种基于用例与源码双向追溯的测试装置及方法”，类似在重建量子纠缠的场景和数据。它成功的将功能用例和对应的代码二者之间的追溯路线，实现了精准无误的可视化。这彻底解决了在黑盒的状态下，软件工程师们无法获知程序内部运行轨迹的难题。自此开始，计算机处理海量测试数据的强大分析能力开始展现。

我们知道，所有的黑盒测试都是人工进行的，它依赖的是人的算力，而软件正确性验证的工作量，主要是组合路径膨胀的问题，人的算力是远远跟不上的。那么精准测试如何从测试的角度对业务进行深度的测试辅助分析呢？我们用量子纠缠的形象类比来说明这个问题。量子纠缠理论我们可以简单理解为两个处在纠缠态的粒子一旦分开，不论分开多远，如果对其中一个粒子作用，另一个粒子会立即发生变化，且是瞬时变化。“一个粒子”会与“另一个粒子”产生相互作用，两个表面互相不关联的粒子，互相作用互相影响，他们对外是一个整体，无法单独描述“单个粒子”的性质，只能描述这“两个粒子”的整体性质，这就叫做“量子纠缠”。类比到软件测试上，我们的被测试软件中常见的软件界面和功能输入输出以及渲染它的复杂代码，他们之间就和两个纠缠在一起的粒子一样，具备强关联性。一个发生了变化，另一个必定发生变化。

精准测试的另一核心功能是：“回归测试用例的自动选取”，就是基于用例和代码的追溯（量子纠缠）关系在进行运算之后全自动得出的。用例和代码精准的追溯机制，使数据开始可以实施大量的智能测试算法。在软件测试理论界，大量论文算法的基础都是默认建立在用例和代码的关系上进行分

析的，但因其追溯机制工业实现难度极大，所以一直停留在学术理论阶段。星云测试引领的精准测试方法体系，落地在精确便捷的商用产品和大型解决方案中，让大量先进的测试分析方法和理论，得以实实在在的验证。

精准测试首次明确的提出了：如何采集和应用双向追溯数据，进行测试辅助分析系统的构建。不管是纯软件系统还是运行在设备中的软件系统，用例与代码精准的数据可追溯性，都是基础性的强需求。比如金融转账业务、用手机拍照、机器人控制器的动作控制指令等，每个操作都有与之对应的代码。星云精准测试产品的强大之处在于：所有的分析算法不需要和被测系统的业务功能做特定的适配，因此可以应用于任意软件系统的测试辅助分析。而传统的白盒在产品实现上以采集和分析总体覆盖率为主，没有能力将覆盖细化到用例层级，因此传统的白盒测试囿于覆盖率的概念中，无法实现更高层次的测试辅助分析需求。

星云测试产品的技术优越性及应用的易用性，使它成为新一代软件测试技术流的中坚力量。星云精准测试在企业应用场景中，为了方便客户还设计了完全静默的“傻瓜”运行模式，客户基本无需增加额外的学习成本。比如测试工程师打开测试用例的 excel 表格，当他准备执行某个用例的时候，只需点击这一项测试用例。随后通过 VBA 技术，直接调用星云精准测试的后台接口，再附加一整套深入应用后台执行线程的用户标签技术，就可以将用例和代码关联和分离出来（这里说的分离是指类似于 J2EE 服务端后台应用）。在对外提供多用户并发访问的情况下，可分离出每个用户执行的用例所对应的相应代码。

前面我们提到功能和代码这两个量子数据的重建，以前主要出现在研发/开发人员相对比较零散的执行单步调试功能的时候（即单元测试），无法存储。国外白盒工具因其设计基因的限制，无法支撑超大型、高复杂度的系统级测试需求。在星云精准测试体系中，只要测试工程师执行测试用例，几乎不需要额外付出，在建立测试用例与数据的对应关系的同时，瞬间即可将海量数据实时采集并存储

起来，用于后续测试大数据的分析和运算上。它的内核设计，使之可以轻松应用在数亿行的超大型应用上，实时采集、存储测试数据，而对原有系统的响应性能不产生干扰。

星云精准测试对软件测试体系是重大的技术革新，它从以下几个层面改变并提升了测试：

1. **对软件测试的深度进行了大幅度的拓宽，让计算机有能力直接对测试用例进行辅助分析，给出更深入的测试决策分析依据。**传统测试主要通过人工业务测试发现缺陷，定位缺陷必须由开发人员进行，二者信息没有精确的数据交互。精准测试主要基于测试人员标识的用例状态，以及自动记录的用例对应的代码路径进行频谱分析。它可以自动计算缺陷产生的代码位置，在发现缺陷的同时，同步产生缺陷定位的结果数据。精准测试提供的测试用例的聚类分析，是基于测试用例的路径空间距离进行聚类，聚类的结果可以直接对用例执行正确性进行审核，对用例进行等价类划分、分析缺陷密集区以及对用例执行分布进行评估等。精准测试提供的回归用例选取结果，经由海量的计算精准推荐而来，总体算法思路与人工分析的思路是类似的，采用人工智能算法模式把大量的计算完全交由计算机去总结分析，精准、稳定、可靠、高速。这个功能相当于解放了测试和开发团队大量的人力资源，彻底解决因团队成员变更而引入的未知风险。

2. **精准测试开辟的测试方法新体系，支撑了黑盒测试快速提升测试效能比的刚需。**精准测试对于企业来讲，相对于比需要人工编写、维护庞大自动化脚本的自动化测试，工作量是低很多的。精准测试和自动化测试是并行的技术方向，企业应用可以根据各自团队的特点来选择，并没有绝对的应用时序依赖关系。自动化测试近些年的应用遇到一个广泛的批评就是：让测试团队疲于应付用例的编写而不是测试用例设计等测试核心技术。精准测试属于测试分析系统，着眼点和立意相对较高，在测试理论、方法和商业应用落地验证与成果上，形成较好的整体闭环。它把测试需要关注的核心，引领到一个正确方向上来。

3. **精准测试在技术特性上解决了测试结果可信性的问题。**在精准测试之前，我们所见到的所有测试数据都是易伪造、易篡改的。例如我们通常使用的测试管理系统，一个用例是否被有效执行，绝大部分是在测试管理系统中被人工录入的。精准测试是在用例动态执行过程中，由计算机内部算法真实记录对应的程序运行逻辑，然后形成后面对测试数据进行精确分析的数据源基础。所以其底层程序运行的数据是没有办法也绝对不允许进行伪造和篡改的。精准测试果断去除了阻碍测试行业发展的重要障碍，让测试的结果精准化、可量化、可衡量化、可信化。此举，将大幅减低测试行业本身的管理成本，有效推动测试行业的快速、健康发展。

4. **精准测试大幅提升了开发和测试部门的协作效能。**前面提到全景调试器，测试工程师执行完用例，即时产生代码层级的全景调试数据；测试完成后一旦用例存在缺陷，那么基于这个全景调试图以及高度可视化的数据路径追溯，开发人员可以直接对缺陷进行定位，不需要在开发环境下重现缺陷，再单步调试以解决问题。实施精准测试之前，开发人员苦于很难有效参与对测试用例进行审核；实施精准测试后，用例的执行结果都映射到了代码层，开发人员通过代码的覆盖视图很容易就可以协助测试人员补充和确认用例，开发和测试的协作变得非常简单和直接有效。

不可否认，在精准测试之前，开发和测试存在沟通不畅的问题。通过精准测试在项目中的有效实施我们会发现，精准测试可以帮助开发提供非常有价值的数据：例如代码和用例的反向追溯可以帮助开发人员修改代码做参考，精准测试自动计算回归范围减轻了开发人员必须协助测试进行分析的工作量，因此，精准测试对开发和测试来讲，是一举两得、相得益彰的。

5. **星云精准测试发明了多个全新的、适应敏捷迭代开发模型下的覆盖率计算方法。**覆盖率是白盒测试最基本的技术，但是随着敏捷迭代的开发模式，即使最高端的白盒测试工具已经几乎没有用武之地，原因就在于由于迭代很快，在某个版本上通常只能采集少量的覆盖率后新版本就发布了。本应在一个版本上分析的覆盖数据，分布到了数个代码结构不一样的程序版本上，因此原有的统计方法和

参考意义基本上都失效了。精准测试提供了累计覆盖率，能够将多个版本的覆盖率以最新版本的代码结构进行投影，在控制流上进行累加。这样测试团队无需关注每个版本的覆盖情况，即可关注一个完整测试周期内所有版本的累积覆盖率。另外，考虑到敏捷迭代每个版本的测试需求，即：针对特定模块和功能范围进行测试（并不是全量测试），精准测试提供了相关覆盖率的计算方法。它可以自动圈定某个用例和用例集有关的代码范围（这些也是基于用例和代码的关系分析基础上动态计算的），在运行少量用例或者某个功能模块的情况下，它可以自动确定相关代码，把不相关的代码从覆盖率的分母中过滤掉。这样相关覆盖率在仅仅运行部分少量用例的情况下，依然具有很强的统计和参照意义。

第三章 精准测试的定义

精准测试定义：由中国团队发起并原创完成测试理念设计和商用产品研发的全新测试技术。旨在测试用例执行过程中，全自动建立任意运行模式的软件系统的功能点与源代码之间高度的可视化追溯机制，获取功能点相关的代码覆盖率并进行精准回归等测试用例深度辅助分析算法的应用。精准测试是测试理论界首次同时使用测试用例及其相关代码两个关键因子，进行质量综合考量和分析的创新测试方法体系。

3.1 基本定义

星云精准测试在测试用例执行过程中，可以同步全自动建立任意运行模式的软件系统功能点与源代码之间的高度可视化追溯机制，即通过内部算法能够将缺陷对应的代码出错位置直接定位出来；能够获取功能点相关的代码覆盖率并进行精准回归等，使测试用例深度辅助分析算法得以强化应用。这是软件测试首次同时使用测试用例及其相关代码两个关键因子，进行质量综合考量和分析的创新测试理论方法体系。这种有效的数据追溯与“量子联动”的突破性技术特性，大大增强了测试的深度与广度，打破了测试部门的成长天花板，为测试过程本身的价值挖掘和测试数据资产的增值，提供了必要而充分的条件。

3.2 关键技术

精准测试体系贯穿整个软件测试生命周期。精准测试主要侧重于系统级测试，在单元测试、集成测试、系统测试、验收测试、回归测试中，均能赋予测试数据强大的追溯能力。关键核心技术有：测试用例和代码逻辑的双向追溯，测试示波器、覆盖率可视化、回归测试用例的自动选取、缺

陷最后执行时序分析、智能缺陷定位、敏捷环境下多版本白盒测试数据的聚合、聚类分析、结合代码结构与动态数据的测试漏洞检出等。

精准测试在测试用例执行过程中，为用户从底层自动建立任意运行模式的软件系统功能点与源代码之间的可视化追溯机制，使用户获取用例级的代码覆盖率等多种高级测试数据。这一突破性的创新智能算法，有力的打破了软件开发、测试、维护及管理人员等之间的数据孤岛状态，实现了软件测试过程和结果的高度精准可视化，在软件高可靠性和高可信度方面，提供了全面而有力的数据支撑。精准测试运行模式是灰盒模式，即：可以在黑盒功能测试的时候，同步完成数据采集和分析计算，让“点测”团队也可轻易切入到精准测试模式，无需改变现有测试形式与流程，不增加额外的技术学习与转换成本。

3.3 应用范围

精准测试适用于任何形态的软件系统。星云精准测试可应用于各种软件包括嵌入式系统、分布式系统、web 应用系统、单机软件系统等各类软件的测试，并且完全不受限于被测试软件本身的业务需求。

3.4 用户收益

1. **实现从发现缺陷到预防缺陷的转型。**减少因为缺陷而对整个研发运维流程造成的返工成本。使发现缺陷前移，减少因项目后期的严重缺陷而导致产品交付延期、成本增高，影响交付质量。
2. **建立跨需求、开发、测试等部门的测试协同平台。**实现业务数据可追溯化和路径可视化。
3. **优化技术与业务方案。**根据不同的质量要求，优先级顺序、技术实现手段等，实现智能灵活的技术方案推荐、交付和应变机制。减少无数据支撑造成的内耗。

4. **企业“精准众测”平台，支撑千人团队同时在线。**测试数据可以根据管理需要做分析，为企业内的成本管控提供选型参考。实现跨地区，跨部门的业务人员，开发人员和测试人员的测试协同，即使不在同一办公地点的人员，也可针对测试需求，测试任务，测试用例和测试缺陷等方面进行远程沟通和实时协同作用，最终完成整个测试过程的实施。

5. **支持敏捷开发与测试模式。**关注新功能的增量测试，支持自动化回归测试，实现高效的 IT 交付。

第四章 精准测试的基础架构介绍

4.1 精准测试的技术架构

精准测试从某些层面来讲，赋予了测试用例真正的生命力。测试用例是测试数据的一种，当测试数据实现路线可追溯可视化的时候，一些高级算法的强大能力就可以显示出来。我们从精准测试的整体架构图中可以做一具体分析。

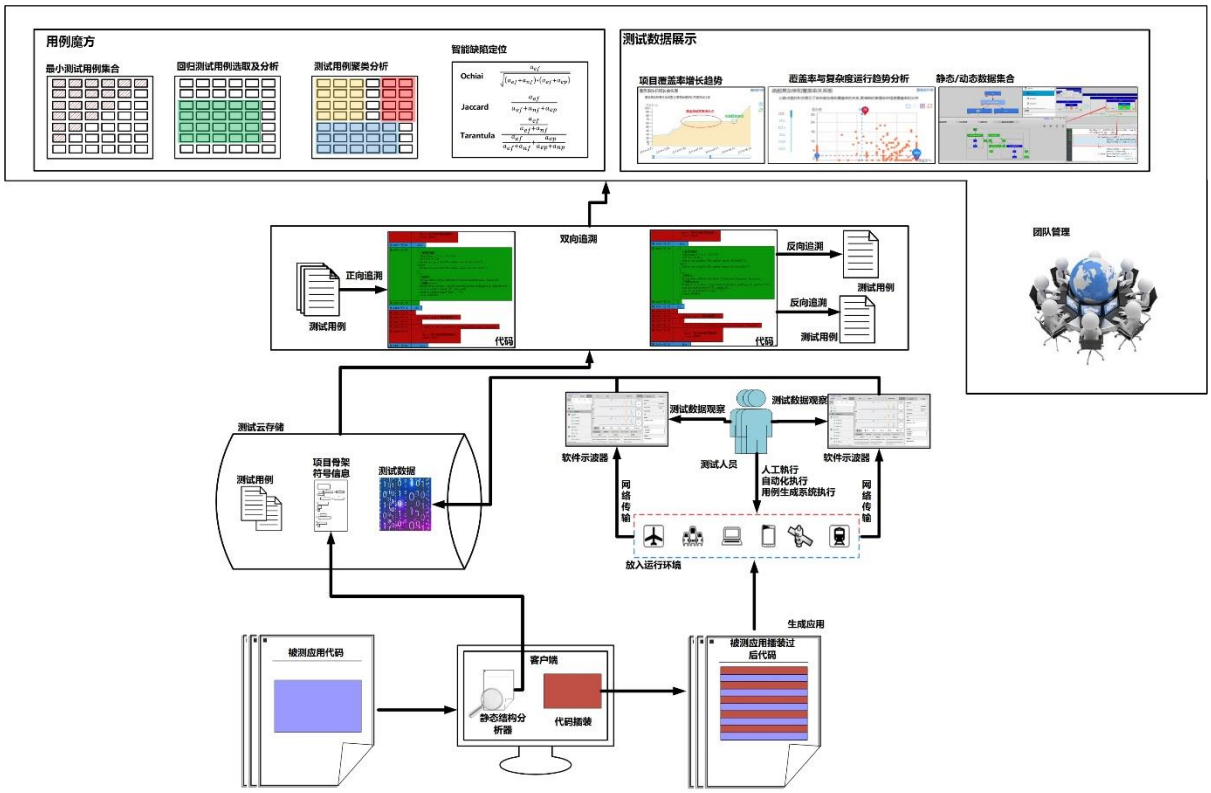


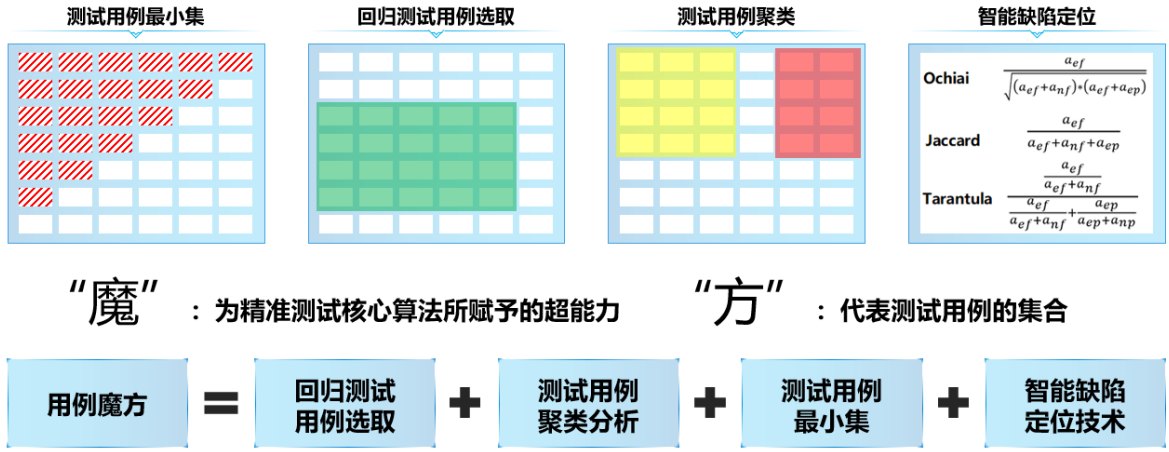
图 4.1-1 精准测试的总体架构图

第一层：利用先进的前置编译器，为客户做源码静态结构分析（在客户的实际环境中，根据客户的需求进行相关的技术配合）；

第二层：将处理好的系统程序放入测试环境运行，测试工程师通过人工或程序自动化的形式，开始执行用例（人工执行用例可以和测试管理平台或者 Excel 表格方式进行对接），精准测试的“软件示波器”采集运行数据并进行高速智能运算，获取精确的测试数据；

第三层：根据采集的代码与对应的测试用例，在星云精准测试平台中实现用例与源码的互相追溯；

第四层：通过精准测试的分析平台，可以对测试数据进行缺陷定位、用例聚类分析、回归测试用例和最小测试用例集等功能的计算，用户还可以根据需求，批量生成相应的测试报告，或进行测试数据高级分析。



精准测试中的回归测试通过内部算法，自动选取新版本修改后可能影响到的测试用例。

图 4.1-2 精准测试的用例魔方

在总体业务架构图的左上角区域，我们可以看到“用例魔方”几个字。大家可能会比较好奇“用例魔方”的涵义，它是精准测试中非常核心的高级回归功能之一。所谓“魔”是精准测试核心算法所赋予的超能力，所谓“方”实际上是代表测试用例的集合，每个测试用例用一个小方块标

识，所有测试用例的集合用一个大方块。当我们把精准测试的和用例分析相关的功能画成架构图形表示的时候，它自然而然地看起来就像魔方。

精准测试体系中，测试用例对应的代码逻辑精确而完整的实现了全自动化的追溯和存储，因此赋予了测试用例深入分析的基础能力。在精准测试的用例魔方中，目前存在三个面（随着后续功能的增加，将增加分析的面）：即回归测试用例选取、测试用例聚类分析、测试用例最小化，同时辅之以智能缺陷定位技术。当测试用例与代码的追溯关系建立的时候，测试魔方的核心功能区即同步构建出来。为数据的多角度分析，提供了丰富的资源素材库。

4.2 软件示波器

软件示波器是星云精准测试独有的功能，它如同软件的质量运行情况“追溯穿行器”一样，在测试工程师按下启动键的同时，即时开始建立用例与代码的自动关联。示波器把采集到的测试数据通过可视化的窗口界面进行实时展示，内容涵盖采集到的块、条件和函数信息。蓝色波形代表写入的数值，黄色波形表示读取的数值，用户可以清晰的看到被测系统的数据变化。它如同心跳监控仪一样，如果被测试程序发生了崩溃，软件示波器就像人的心脏停止跳动一样，一根横线拉直向用户报警。如果正常采集到数据，会有持续的波形展示出来，高效而精准地监控程序细微的运行状况。示波器精密捕获每个软件单元任何微小的运行波动和行为改变，支持多次运行数据的比对。它可以根据需要记录崩溃前的至少 50 个块，使“崩溃重现”变得轻松简单。

软件示波器中的测试用例可以从现有的测试管理系统导入进来，先选中用例点击开始，驱动被测试系统运行后，软件示波器就会采集到程序内部运行逻辑对应的波形信息。用例执行结束后，点击停止。这个用例运行阶段的数据，通过开始和结束的点击，边界就记录下来了。

图 4.2-1 软件示波器

面板的正下方可以展示函数的各种调用信息。包括类、函数，参数类型等。清晰的列示出参数列表、类运行情况、内存检测数据、数据库拦截等多角度的数据分析追溯情况信息。

示波器观测的维度较多，用例与代码的追溯是精准测试的基础功能，后面的高级算法都在这个基础上展开。用例和代码的追溯就像一个全景的调试器，只要功能由测试人员运行，所有的内部代码执行逻辑瞬间就可以展示出来。通过测试数据的反向追溯分析，开发人员可进行一致性修改，避免修改引入新的缺陷，通过正向追溯结果，开发可对用例的执行进行全面掌握，可用于快速修复缺陷和详细实现确认。

同时软件示波器也提供一个辅助的等价类划分的功能，它将一个用例从开始到结束所执行的路径信息终值，完整记录下来。如果两个用例终值不一样，就可以确定为不是等价类。对于很多从功能表面很难界定是否等价类的测试用例，软件示波器可以给出精确结果。

因此软件示波器的价值与意义在于：

（1）只要测试开始执行，即可以透明方式高速采集功能运行过程中对应程序的运行逻辑。

（2）在系统高速运转下采集，可保证对原有应用无干扰，超过 1500w/s 的采集速率。

（3）可采集程序的条件，执行路径，执行参数，内存使用等动态运行数据。

为了方便客户在运行项目测试时，一边对被测系统做实时数据监测，一边同步观察到示波器里面数据写入和读取的波形，星云做出了实时数据监测的悬浮窗缩略图，减少了切换带来的工作量。它不影响原有被测系统的界面，以半透明悬浮的方式展示在被测试应用界面的前面。

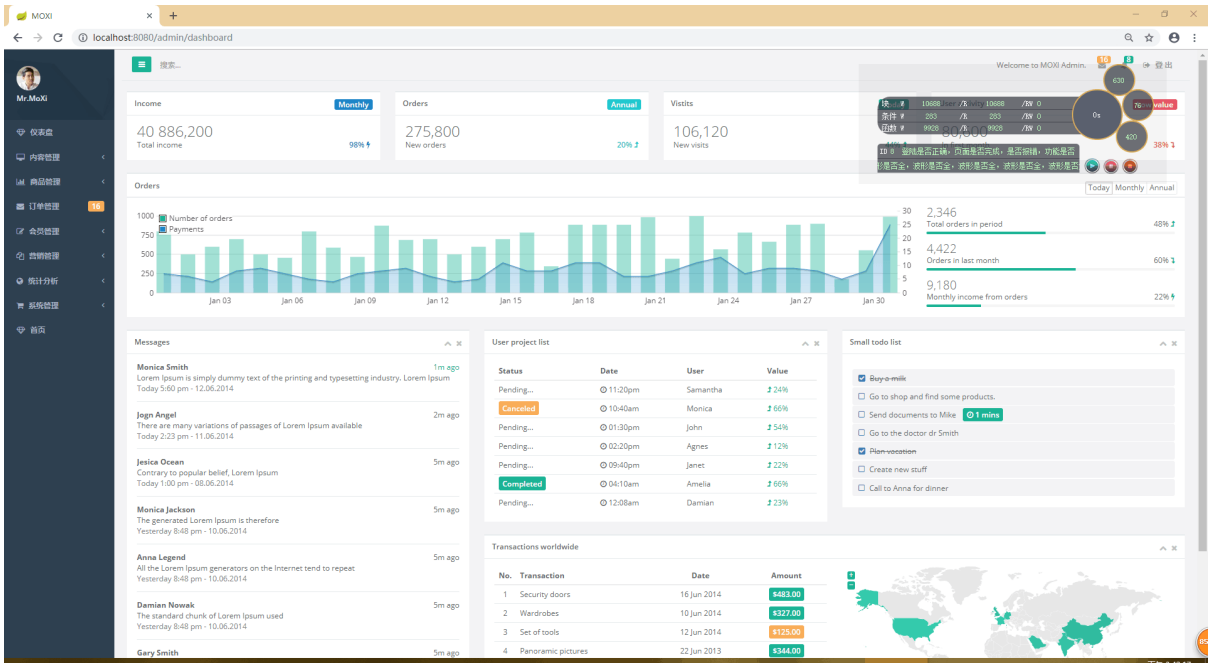


图 4.2-2 软件示波器悬浮窗

4.3 精准测试的双向追溯

精准测试的测试用例和代码的双向追溯功能，是精准测试核心技术之一。如同前文的“量子纠缠”，即运行测试用例的同时，精准测试可以通过程序自动的记录并追溯到这个测试用例相应执行的代码。如果测试人员关注某一些代码行，它可以追溯出哪些测试用例在运行过程中运行过这段代码。

通过这个技术特性，测试工程师的每个测试用例都可以进行量化分析和统计，提供了开发人员和测试人员之间精准的数字化交流依据， 增加测试和开发的交流效率。

双向追溯技术记录了每个测试用例对应的程序内部的执行细节，细致到每个条件、分支、语句块的执行情况。开发人员亦可通过双向追溯的结果去理解程序逻辑，进行软件维护以及进行可一致性的修改。

4.3.1 双向追溯技术正向追溯

1) 将测试用例和代码执行信息自动关联，可追溯到函数级别及代码块级别；

2) 通过正向追溯可直接把 BUG 定位到故障和缺陷逻辑相应的代码，并提供最后运行的时序数据；

3) 通过正向追溯自动记录产生功能对应的详细设计实现，辅助软件解耦和架构分析。

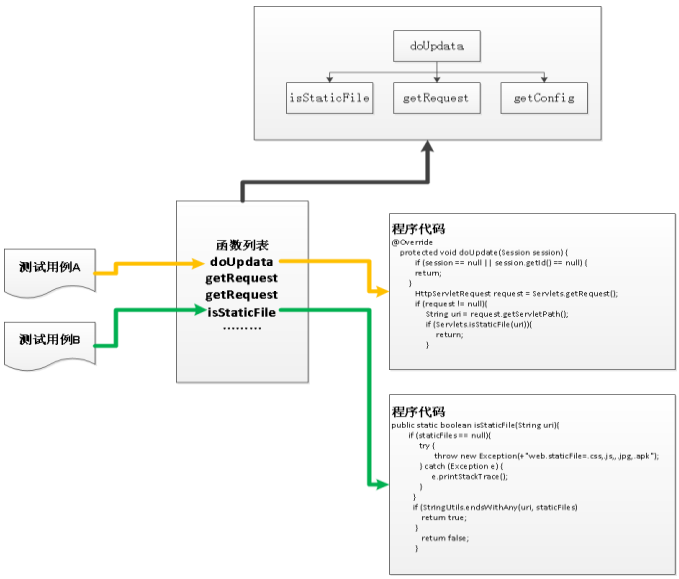


图 4.3.1-1 双向追溯(正向)-测试用例追溯到代码

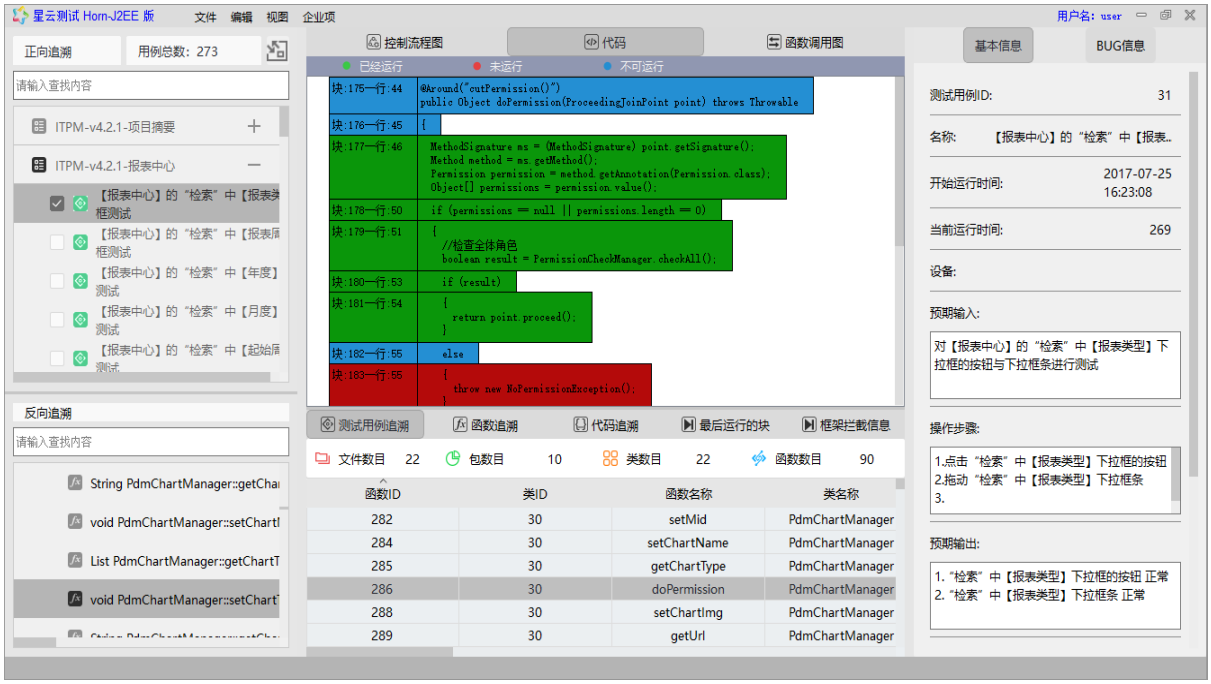


图 4.3.1-2　双向追溯(正向)-测试用例追溯到代码

4.3.2 双向追溯技术反向追溯

1)　将代码执行、函数、代码块级别和测试用例执行信息自动关联；

2)　通过反向追溯可直接观察代码变动所影响的测试范围；

3)　协助开发，进行代码修改后影响功能的范围评估；

4)　协助测试人员对代码修改部分所影响的测试用例进行评估。

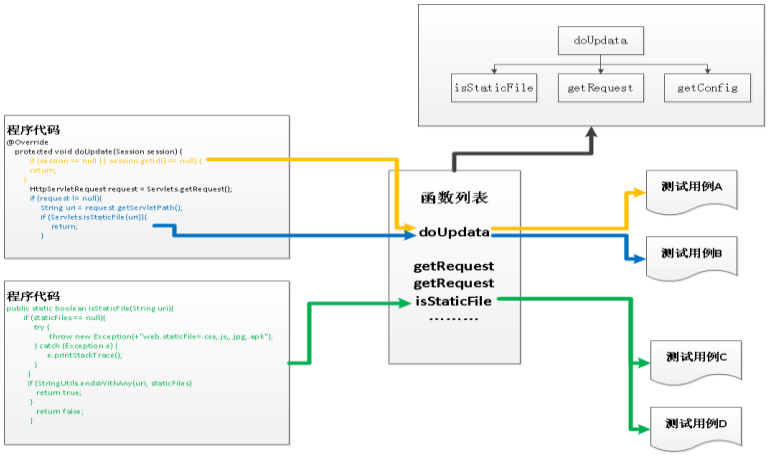


图 4.3.2-1　双向追溯(反向)-代码追溯到测试用例

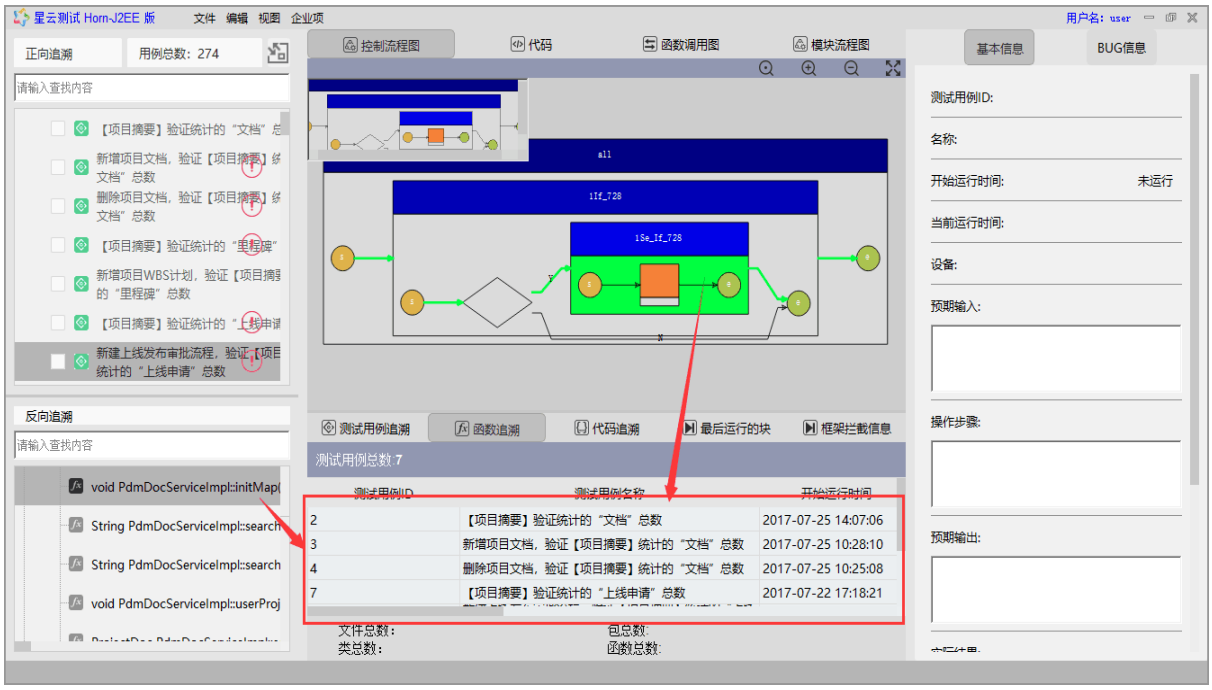


图 4.3.2-2 双向追溯(反向)-代码追溯到测试用例

4.3.3 数据追溯技术-追溯测试用例的全景调用

精准测试通过正向追溯把测试用例运行的代码执行进行了全景绘制，在全景图中，测试人员可以有效的观察到函数之间的整体的调用与走向，观察出被测模块与上层之间的调用关系。

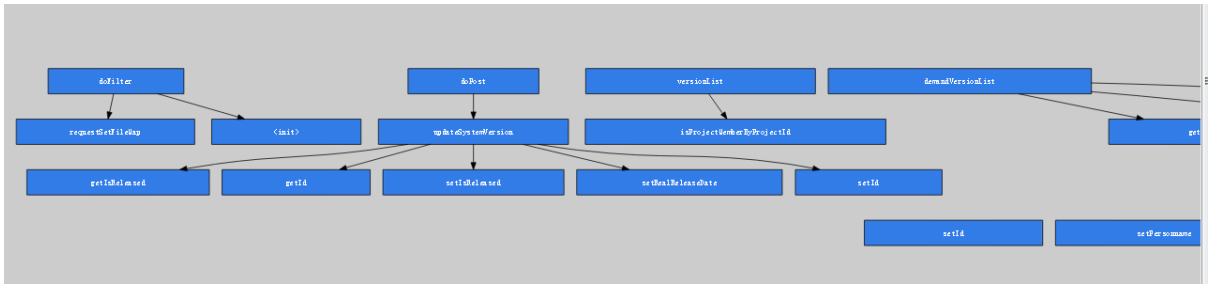


图 4.3.3 测试用例运行的代码整体调用

第五章 精准测试的核心组件与功能

精准测试的核心组件与功能：软件测试示波器、用例和代码的双向追溯、智能回归测试用例选取、覆盖率分析、缺陷定位、测试用例聚类分析、测试用例自动生成系统等，完整的构成了精准测试技术体系。

精准测试系统本质是一套强大的计算机开发与测试系统，实现了数据的可视化联动，以及开发辅助分析。它的关键技术赋予了测试用例和代码强大的相互追溯能力，随后衍生实现了很多高级测试功能与算法。精准测试系统将用例深入到代码层分析后，可以大幅改进人工测试所产生的各种问题。

接下来将从风险控制、工作协同、敏捷迭代方面详细解析精准测试的核心功能和实际收益。

5.1 风险控制

5.1.1 七种测试覆盖率

星云精准测试提供 7 种测试覆盖率：分别为：SC0 语句块覆盖率、True 覆盖率、Both 覆盖率、CDC 覆盖率、Branch 覆盖率、MC/DC 覆盖率。

用户首先选择分析覆盖率的维度，例如选择了“SCO 语句块”维度，那么系统就会将被测试程序的所有语句块结构展示出来，并且用颜色表达覆盖情况，绿色代表覆盖，深蓝色代表未覆盖。同时告知覆盖率的分子和分母都是哪些，非常清晰的展示覆盖率可视化结果。精准测试在前期已经对程序的静态结构进行了深度的分析，因此用户在覆盖率可视化界面，根据选择的维度在代码层面上把需要展示的结构单元都结构化的展示出来。例如图 5.1.1-1 七种测试覆盖率中的第二张图

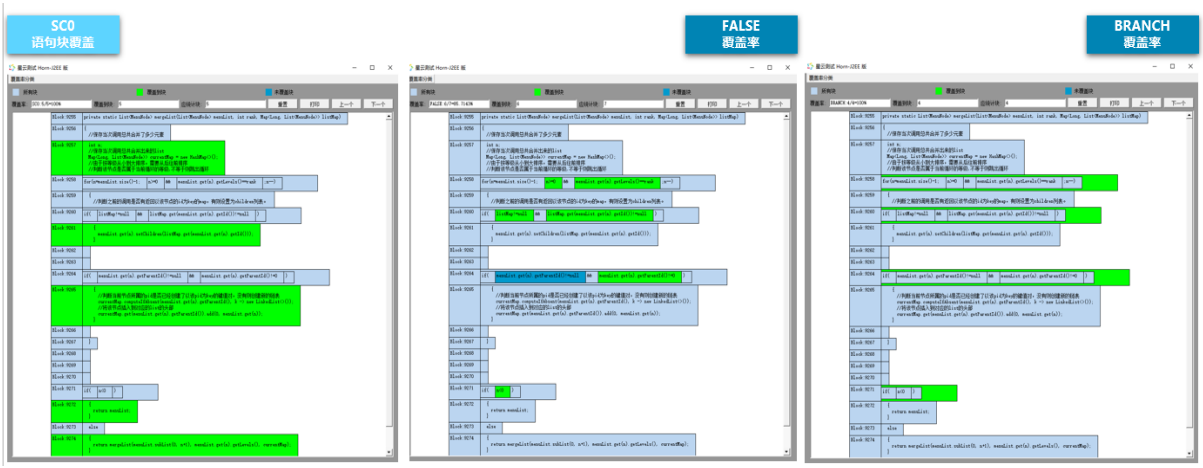


图 5.1.1-1 七种测试覆盖率

MC/DC 覆盖率可视化

MC/DC 覆盖率，即修正判定条件覆盖，该覆盖率数据 MC/DC 是 DO-178B/C Level A 认证标准中规定的、欧美民用航空器强制要求遵守的覆盖率标准。MC/DC 覆盖率可以基本保证被测试软件无缺陷，对于大型系统的可靠性要求很高的一些关键模块，建议采用这个覆盖率标准。MC/DC 覆盖简单来说，就是追踪复合条件中每个子条件的真假翻转，在其子条件不变前提下，是否都独立的影响了整个条件的真假值。星云精准测试系统会自动的把各种组合都列好，通过颜色表达哪些子条件已经满足，以及对应满足情况时其他独立子条件的组织情况。

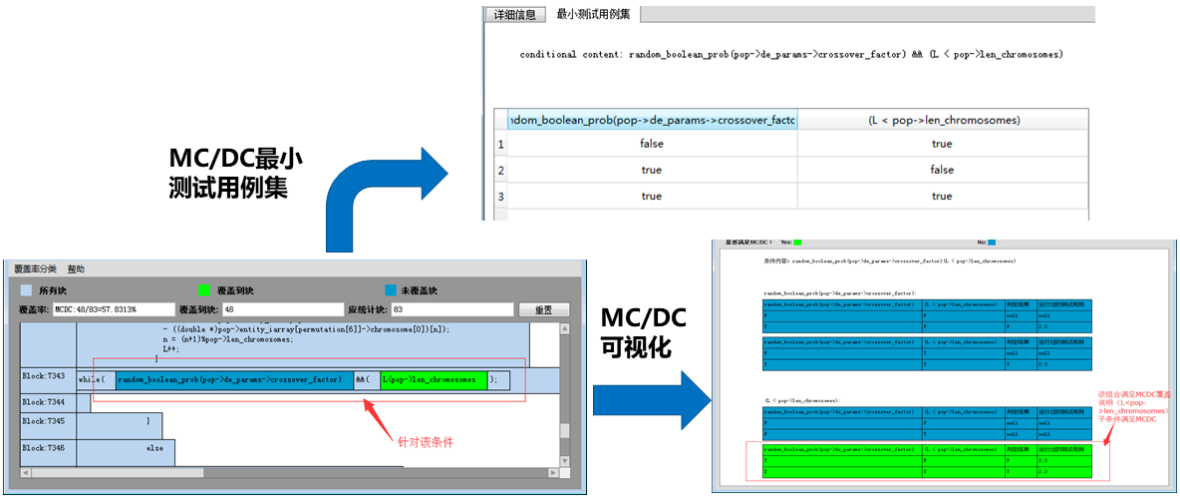


图 5.1.1-2 MC/DC 覆盖率可视化

条件组合可视化展示

精准测试对于多条件组合的代码，采用了最新研发的条件组合可视化视图进行展示。视图中，用户可以观察到每个条件的真假运行情况，以及条件与条件的组合运行情况。对于测试人员来说，当全部条件组合之间的 T 与 F 都完全满足时，即可实现代码全路径覆盖。

图 5.1.1-3 条件组合可视化展示

5.1.2 新增代码覆盖率

敏捷模式下，因迭代频繁其存量的代码量很大，通常更关注增量覆盖度量。精准测试可以在程序新版本发布后，自动计算新增（变更）代码的范围，给出新增代码的覆盖率。覆盖率的分母中的函数都是变更和新增的函数。与此同时，基于反向追溯的功能，我们还可以给出新增代码对应的测试用例名称。当某个新增函数没有达到很高覆盖率的时候，我们通过反向追溯的用例，可以判定因为哪些功能范围的用例设计不充分，导致了新增代码覆盖率不高。

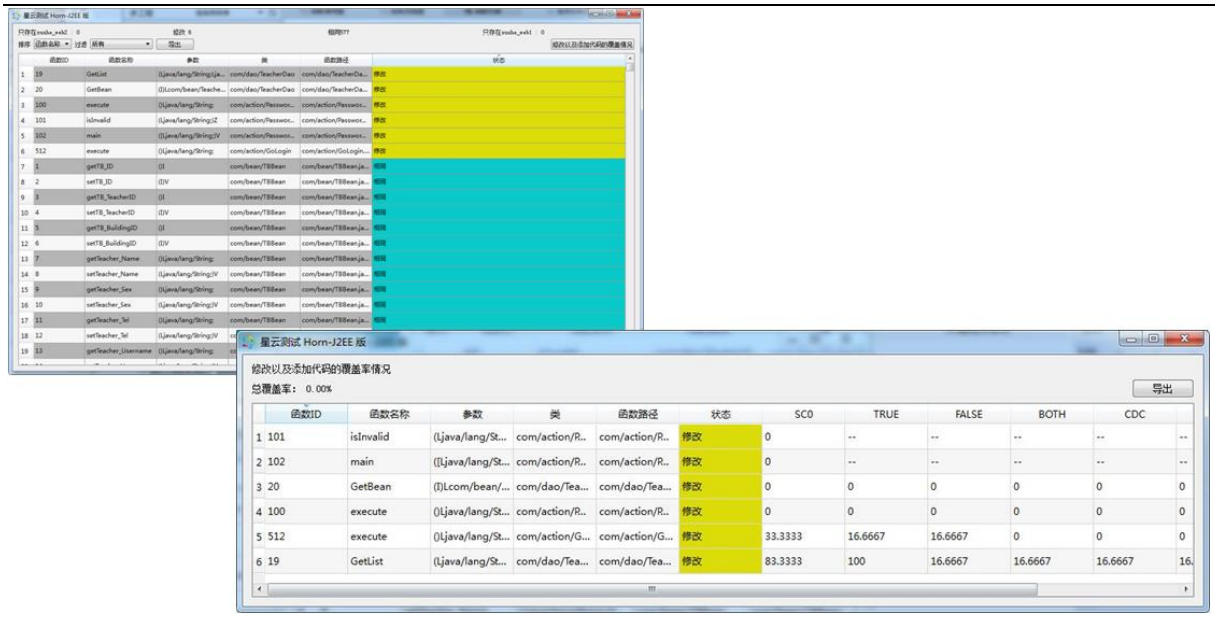


图 5.1.2　新增代码覆盖率

5.1.3 测试覆盖率范围筛选与再统计

在做精准测试或统计覆盖率时，往往测试管理者、开发人员、测试人员为了保证测试覆盖率的正确性，会对某个方法、类进行查看或在统计中把代码中一些废弃的函数、某些特殊情况下无法测试到的代码进行移除（至少是做相应备注），从而让测试代码覆盖统计率达到更加准确。星云精准测试在设计中，通过多种搜索、方法、类、模块过滤等功能，把需要统计的范围进行缩小、不需要统计的去除。根据用户的选择，进行覆盖率再统计展示。

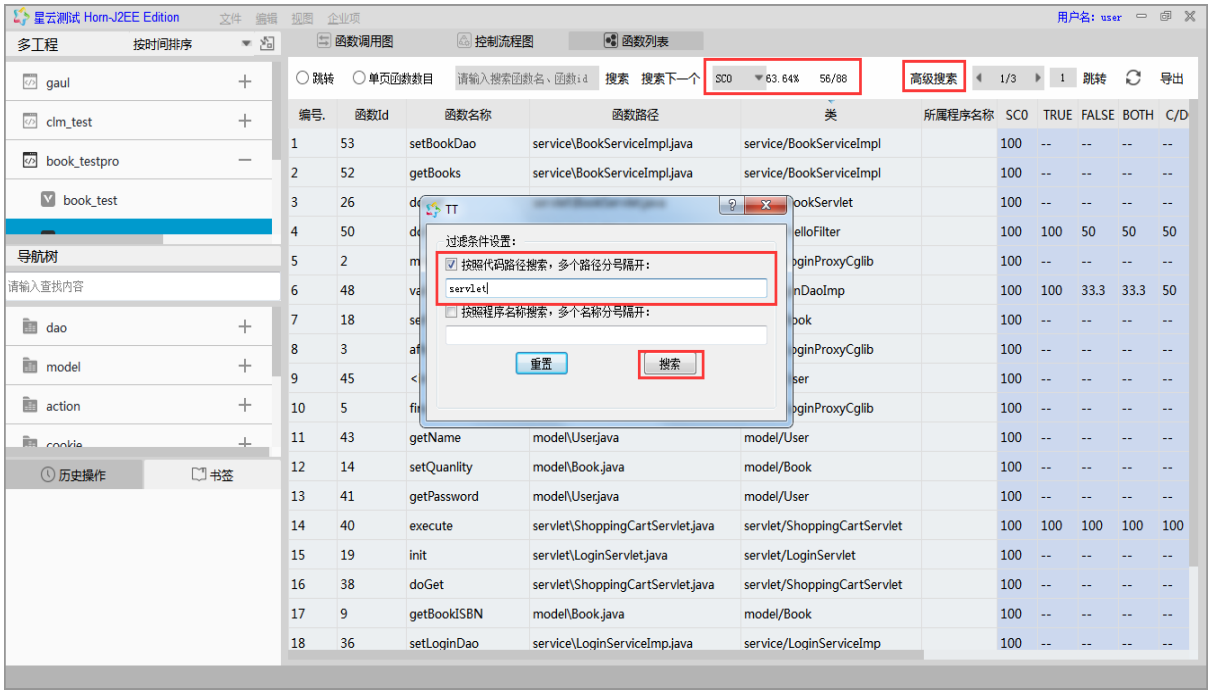


图 5.1.3　测试覆盖率范围筛选与再统计

5.2 工作协同

5.2.1 打通开发与测试的隔阂

精准测试打通开发与测试的协同工作通道，使得开发与测试能够更好的沟通，提高工作效率。传统模式下，开发人员关注的是代码，测试人员关注的是业务角度的测试用例，彼此的直接关联相对较弱。开发和测试的沟通，基本就是采用自然语言、Excel 表格、内部系统等，存在交流信息不够严谨的问题。例如测试工程师发现一个缺陷，提交到缺陷系统，开发需要花费大量时间再行理解、准备数据、复现、调试，直到最后的修正。因为业务上的功能执行和代码并没有明确的关系，通常测试工程师执行完功能测试用例后，让开发人员帮助评审也非常困难。

若测试工程师提供的测试结果都是比较模糊的功能逻辑描述，重现缺陷需要花费大量的时间。开发人员修改代码后，对于变更描述，以及变更引起的关联问题描述通常也都很模糊，导致测试又出现新问题。

企业采用精准测试技术后，通过执行用例可以直接追溯到对应执行的程序代码块，这样的数据化沟通，将使开发人员和测试人员之间的协同工作效率大大提高。

图 5.2.1 协同模式

5.2.2 源码动静态数据的统一

星云精准测试通过插装得到的项目静态结构信息，结合测试后采集到的测试数据，能够精准记录测试的过程，通过这些静态数据和动态数据视图，便于开发人员基于图形化结果进行快速分析。

对于不懂开发的测试工程师，通过程序控制流程图的图形以及通过颜色表示的覆盖信息，可以直接看到程序内部漏测的逻辑是什么，也可以通过这些结果直接与开发沟通，进行辅助用例和逻辑的补充。

因为内部逻辑的强追溯性并且能够图形化的打开，可以有力保证黑盒测试后期开发快速理解并解决瓶颈问题，保持全程测试的高效执行。

图 5.2.2-1 源码静态结构与动态测试数据统一图（函数调用图）

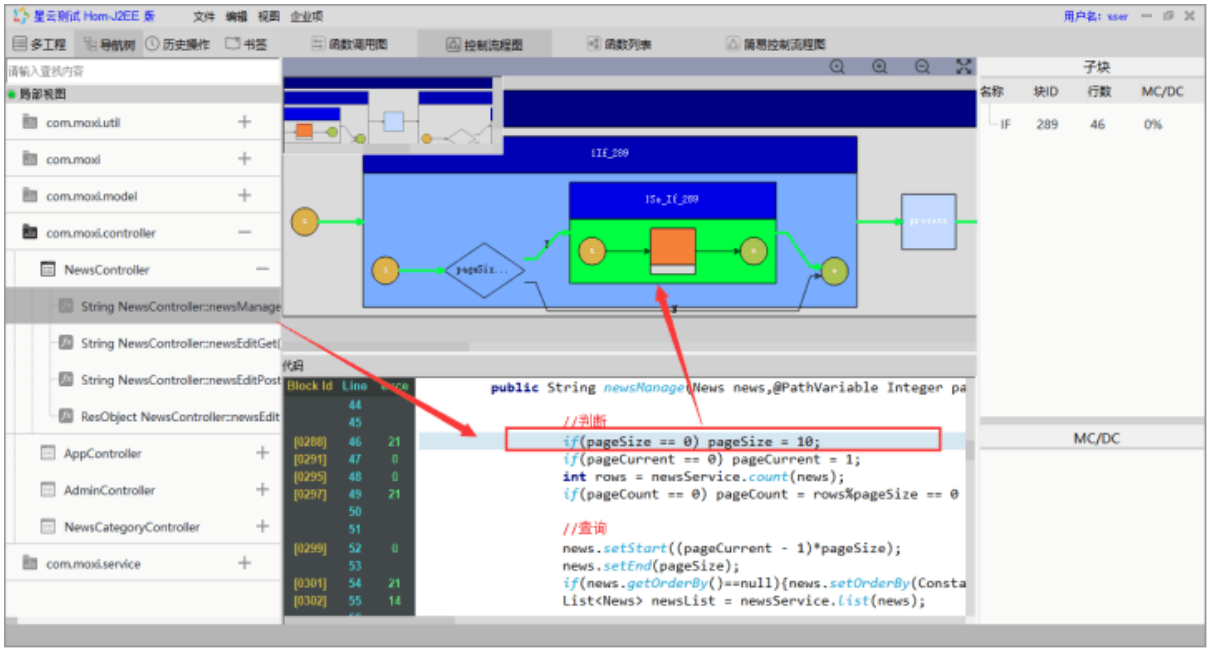


图 5.2.2-2 源码静态结构与动态测试数据统一图（控制流程图）

精准测试在程序静态分析的基础上，可以对程序绘制可视化的图形，同时将动态执行的覆盖信息染色到这些视图上。对于不懂开发的测试人员也可以很清晰的看到程序的哪些结构节点没有被覆盖到。例如图形中蓝色的节点是未覆盖的节点，绿色的是覆盖的，因为控制流程图本身有分支，嵌套等各种关系，测试工程师可以很容易判断出覆盖的范围大致是哪些。而如果有开发人员介入，可以更清晰地

分析测试工程师执行的用例所遗漏的程序逻辑。

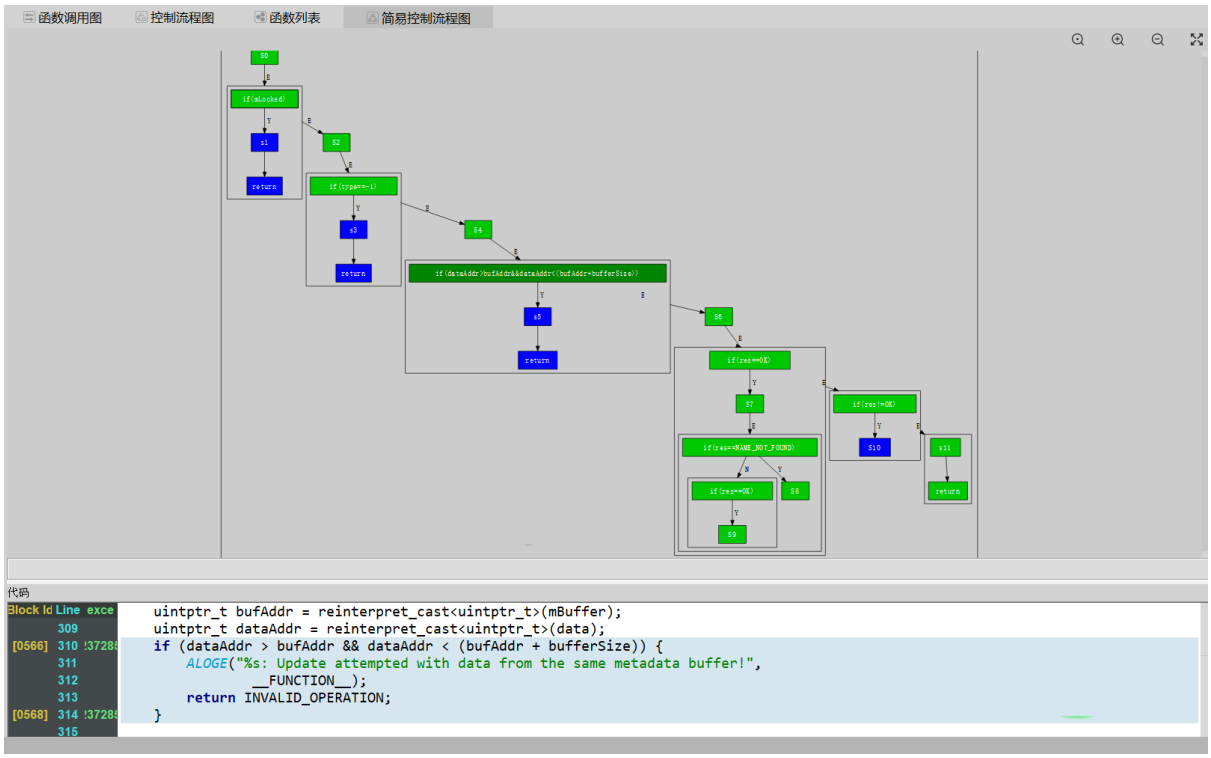


图 5.2.2-3 源码静态结构与动态测试数据统一视图

5.2.3 缺陷最后执行时序分析

星云测试可自动捕获缺陷或崩溃发生时，程序最后执行的详细路径信息。缺陷发生后，开发人员能够直接看到缺陷出现时，代码执行的时序和路径信息，直接定位缺陷并排查问题，节省大量的沟通、复现和调试的时间成本。

当功能执行发现缺陷后，在软件示波器上可以立即按下“停止”键，那么最后执行的代码序列就可以被抓取到，开发人员可快速定位缺陷最后执行的 50 个代码块、条件、判断的各种执行信息。

在下面视图中，我们可以根据标号（从 1 到 50），看到代码最后的运行时序，在图形里面的每一个绿色小方块为一个代码语句块或者一个条件、判定，在一个大方框下的绿色方块代表一个函数内部的代码块。经过布局算法后产生下述图形。

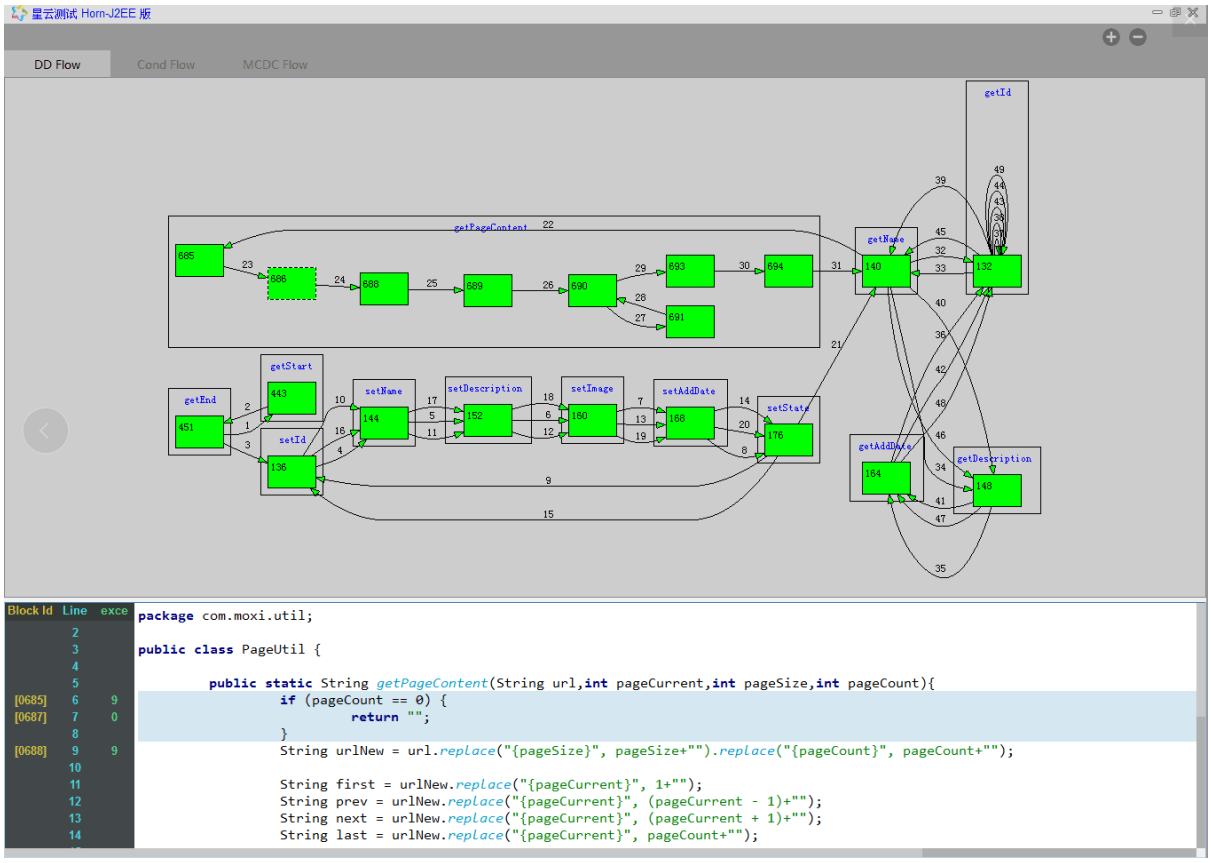


图 5.2.3 缺陷最后执行时序分析

5.2.4 智能缺陷定位

通常测试工程师只是负责发现缺陷，缺陷的具体定位只能交由开发人员来执行。精准测试打破了测试部门的天花板，即通过内部算法能够将缺陷对应的代码出错位置直接定位出来。这相当于增加了测试深度，同时体现了测试数据和测试过程本身的价值。

对于测试工程师来说，只要发现用例相似而程序在输出上有对错区分，就可以使用精准测试的智能缺陷定位功能。精准测试平台通过测试人员在功能测试阶段标记的用例执行状态，以及软件示波器自动记录的程序运行频谱，自动分析缺陷的出现的代码块。因精准测试平台可以获取每个用例执行时详细的路径追溯信息，测试工程师只要告知系统用例的状态，例如是否通过是否正确，那么精准测试内部算法就会去根据正确和失败的路径差异进行计算，给出缺陷出现具体位置的可疑度排名。

这个功能可以大大增强测试在整个开发流程的参与度和测试数据价值，也让开发人员减少了自

己去模拟场景的时间，快速提高开发和测试的协同和配合的效率。

1) 对于同类测试用例，经过多组测试可给出非常有效的结果。

2) 列出的可疑代码，可直接通过测试过程给出，提升测试的价值及产出。

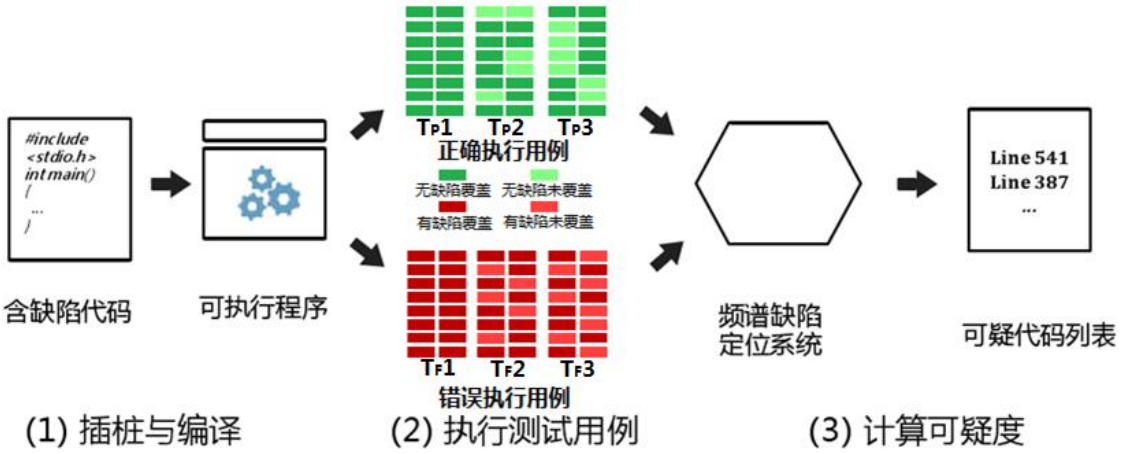


图 5.2.4-1 通过功能测试频谱法分析进行智能缺陷定位

选择可疑度算法、得到可疑度高的代码块，关联源码后，可根据代码可视化查看具体位置。可疑度计算有一个公式，并不复杂，通常每个代码块有 2 个变量，四种状态值。分别是：是否执行、是否通过，这样每代码块都有一个可疑度值。

星云精准测试提供 3 种常用计算公式，供大家参考。

Ochiai

$$\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})*(a_{ef}+a_{ep})}}$$

Jaccard

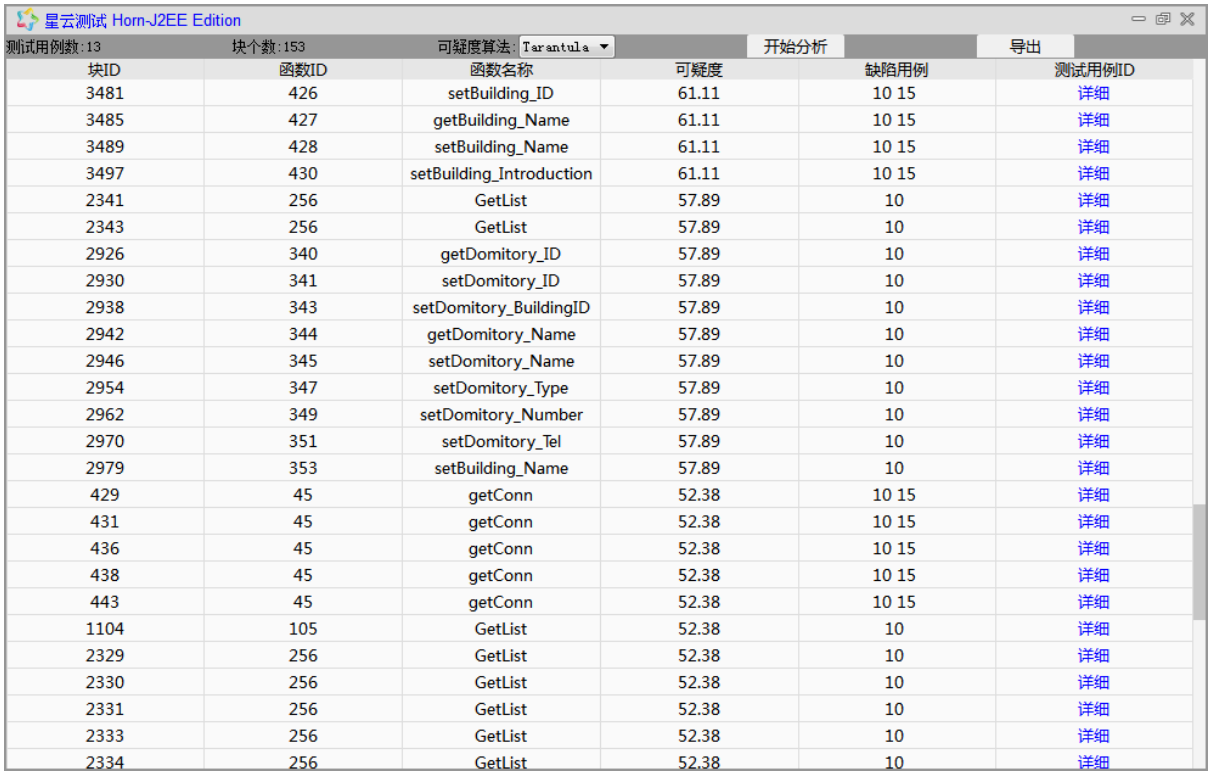
$$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$$

Tarantula

$$\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$$

aep 表示通过且覆盖到该块的测试用例的个数、anp 表示通过且未覆盖到该块的测试用例的个

数、aef 表示未通过且覆盖到该块的测试用例的个数、anf 表示未通过且覆盖到该块的测试用例的个数。结果表示该块的可疑度。



块ID	函数ID	函数名称	可疑度	缺陷用例	测试用例ID
3481	426	setBuilding_ID	61.11	10 15	详细
3485	427	getBuilding_Name	61.11	10 15	详细
3489	428	setBuilding_Name	61.11	10 15	详细
3497	430	setBuilding_Introduction	61.11	10 15	详细
2341	256	GetList	57.89	10	详细
2343	256	GetList	57.89	10	详细
2926	340	getDomitory_ID	57.89	10	详细
2930	341	setDomitory_ID	57.89	10	详细
2938	343	setDomitory_BuildingID	57.89	10	详细
2942	344	getDomitory_Name	57.89	10	详细
2946	345	setDomitory_Name	57.89	10	详细
2954	347	setDomitory_Type	57.89	10	详细
2962	349	setDomitory_Number	57.89	10	详细
2970	351	setDomitory_Tel	57.89	10	详细
2979	353	setBuilding_Name	57.89	10	详细
429	45	getConn	52.38	10 15	详细
431	45	getConn	52.38	10 15	详细
436	45	getConn	52.38	10 15	详细
438	45	getConn	52.38	10 15	详细
443	45	getConn	52.38	10 15	详细
1104	105	GetList	52.38	10	详细
2329	256	GetList	52.38	10	详细
2330	256	GetList	52.38	10	详细
2331	256	GetList	52.38	10	详细
2333	256	GetList	52.38	10	详细
2334	256	GetList	52.38	10	详细

图 5.2.4-2　智能缺陷定位展示

5.3 精准测试对敏捷迭代的支持

5.3.1 敏捷迭代下多版本白盒测试数据的聚合

在敏捷环境下由于版本迭代速度很快，每个不用的代码版本上通常只能采集到少量的覆盖率。一旦发布新版本，就意味着代码发生了变化，覆盖率数据就需要重新采集，但是每个版本采集的少量覆盖率，从分析层面上并没有多大的意义。

针对以上问题，精准测试给出了“**累计覆盖率**”的计算方法。它将一系列迭代版本的覆盖率，在最新的程序版本上进行投影累加。用户可将一个阶段各个版本的覆盖率累加起来进行分析。算法的思路是以最新版本代码为基础，以某个函数为单位，一直往前累加。直到这个函数在之前的某个版本代

码发生了变化，就停止累加。例如下图函数 A 的 4 个版本的覆盖都可以累加，是因为这个函数在 4 个版本中都没有发生变化。而红色的函数 B 只累加了 3 个版本，是因为从版本 v2.0-2 后这个函数发生了代码变更。之前的代码覆盖就不能累加了。

星云精准测试-敏捷环境下多版本白盒测试数据的聚合如图所示。

图 5.3.1-1 敏捷环境下多版本白盒测试数据的聚合

这种累加是在确保函数代码没有发生变化的情况下，在控制流上对相应节点的覆盖率进行累加。例如下图中 3 个版本，覆盖率不同的分支和控制流，数据累加后可以看到所有分支都覆盖到了。

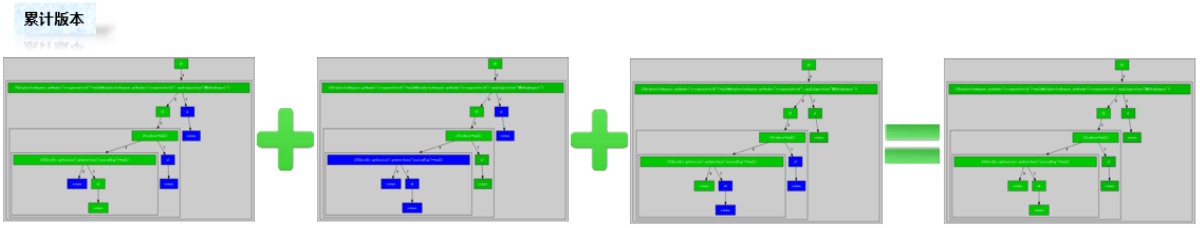


图 5.3.1-2 敏捷环境下多版本白盒测试数据的合并分析

5.3.2 聚类分析

星云精准测试提供的聚类分析功能，根据测试用例的函数执行剖面的向量化信息，对测试用例进行精确的空间距离计算后执行聚类分析。聚类结果可以分析被错误执行的用例，例如不相关的功能点聚类到一起，则说明其测试执行可能存在错误。

精准测试提供的聚类分析功能，也可以辅助找到缺陷分布的密集区域。大部分情况下，缺陷分布会呈现 2/8 的聚集特性。在时间紧张的情况下，我们可以通过聚类结果，每个类选取中心点以及周边几个用例。如果没有问题，就可以去测试其他聚类，如果发现一个类缺陷概率高，那么这个类就需要进行重点测试。通过聚类结果可以分析测试用例的分布密度等信息，辅助进行测试决策。

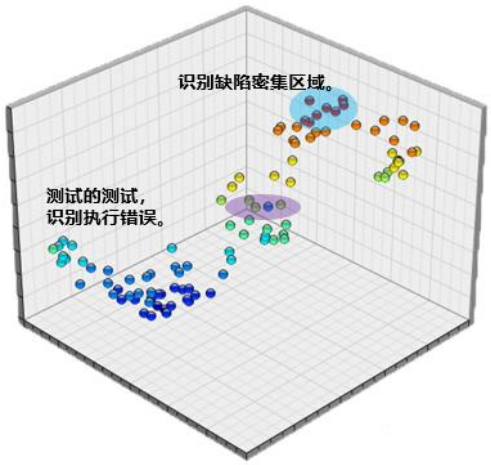


图 5.3.2-1　测试密度

下图中测试用例分类都有一个名字，这个名字是聚类结果中每个类中心点的测试用例名字，它基本标定了这个类的功能点范围。

聚类的大小代表了某个功能范围测试是否足够充分。例如一些应该重点测试的功能，聚类结果中的用例数应该很多；一些小众的功能，聚类中用例数应该比较少。一个类中用例数越多，这个类圆圈比例越大，反之则越小。在聚类的基础上，我们也可以对一个类中的等价类用例进行分析。是等价类的用例，会分成一组专门展示。

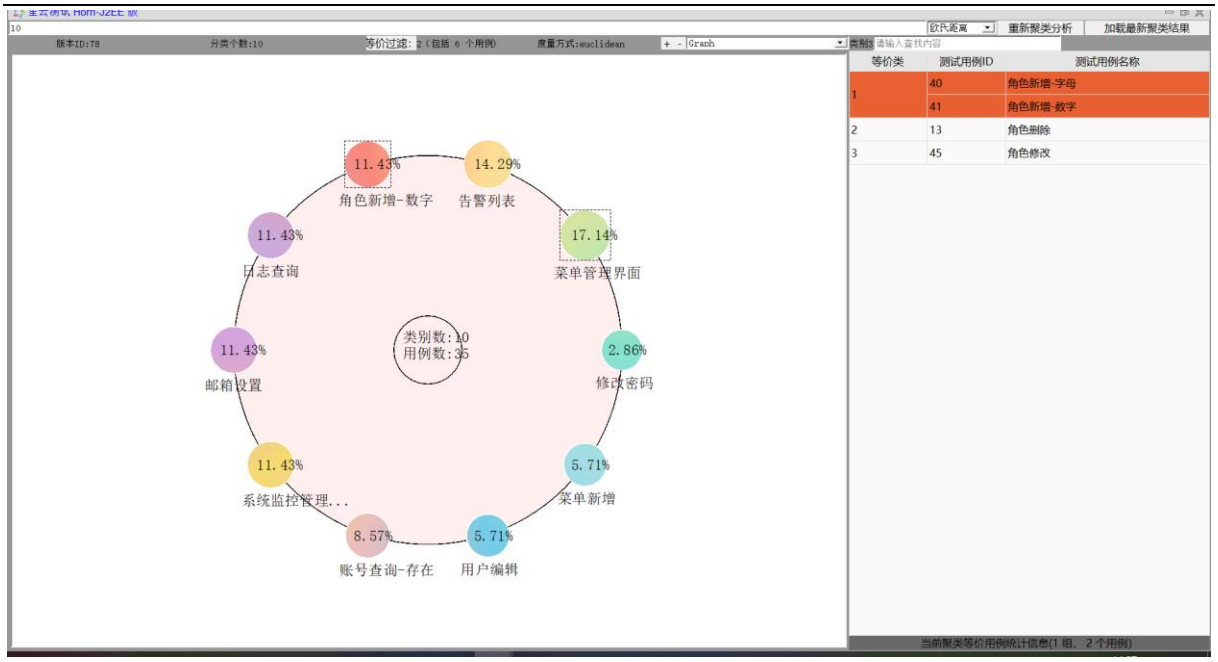


图 5.3.2-2 聚类分析

5.3.3 覆盖率漏洞检出

在敏捷迭代过程中，通常没有充分的时间将所有函数的覆盖率都达到一个很高的层级（Level）。精准测试结合代码结构和动态数据综合分析，通过计算直接筛选出潜在的高危测试漏洞，可以在短期内确定高危漏测模块并针对性的解决，帮助用户快速找到严重缺陷。

1) 当测试时间不充分的时候，执行完黑盒测试以后，先看测试漏洞列表，里面显示了通过静态信息和动态信息计算，得到的最高风险的漏测点模块。

2) 我们可以通过复杂度进行计算，因为复杂度高的模块一般来讲，它是相对重要的模块并且逻辑复杂，如果动态覆盖比较低，我们会优先筛选出来进行排序。

3) 处于调用和被调用中间的模块，因为属于中间关键模块，我们也会计算它的扇入扇出，和动态覆盖率信息。如果它的比率很高，将被认为是高风险模块，被筛选出来进行排序。回归时，应优先测试风险指数高的高危模块，补充他们的覆盖率。

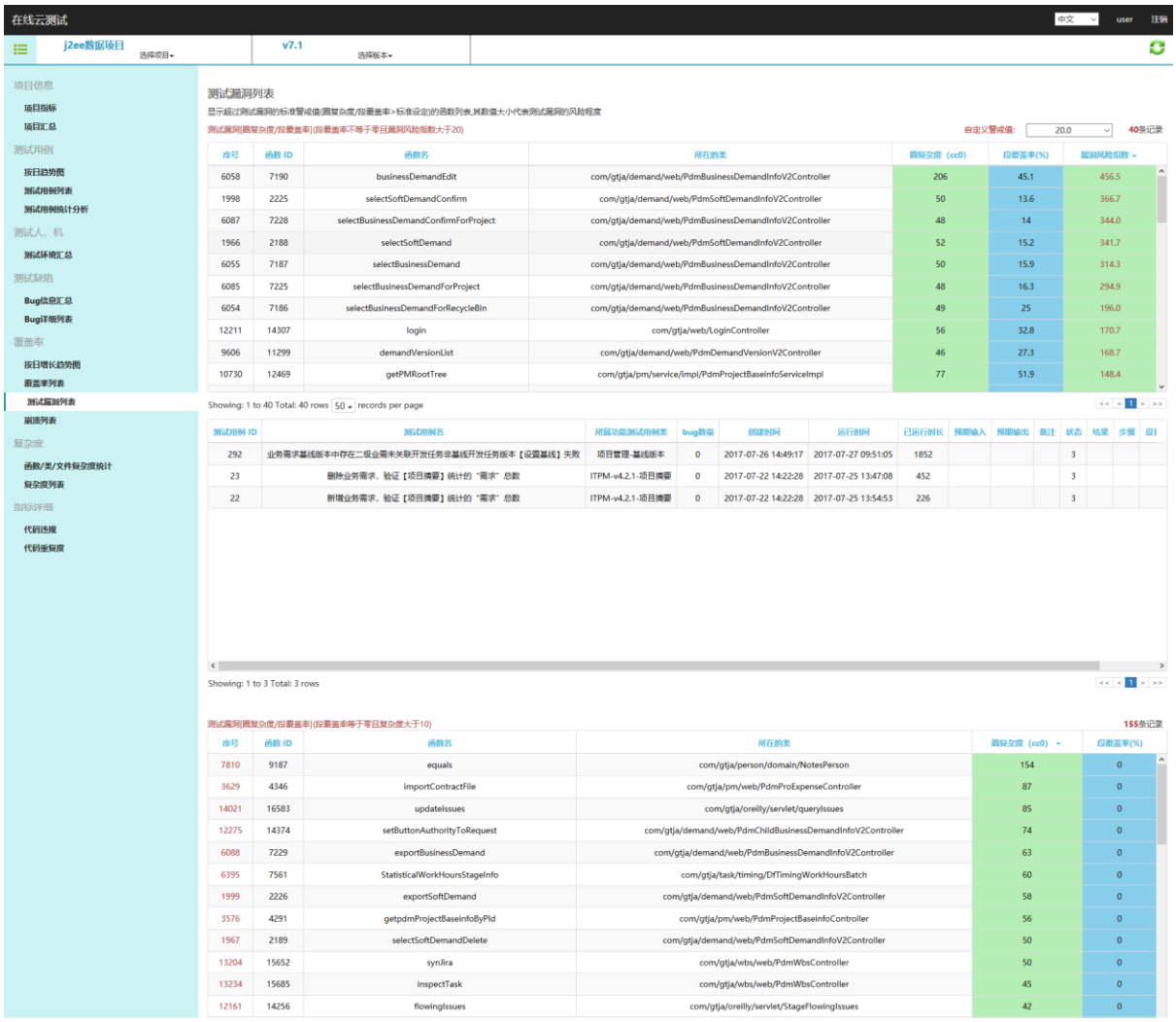


图 5.3.3 漏洞检测列表

5.3.4 最小测试用例集

精准测试也可以对用例集进行优化。比如用户有大量用例的情况下，尤其是自动化用例集含有长期维护的冗余用例。精准测试平台可以对很多重复用例的逻辑进行筛选和过滤，优化出满足当前总体覆盖的最小用例集。

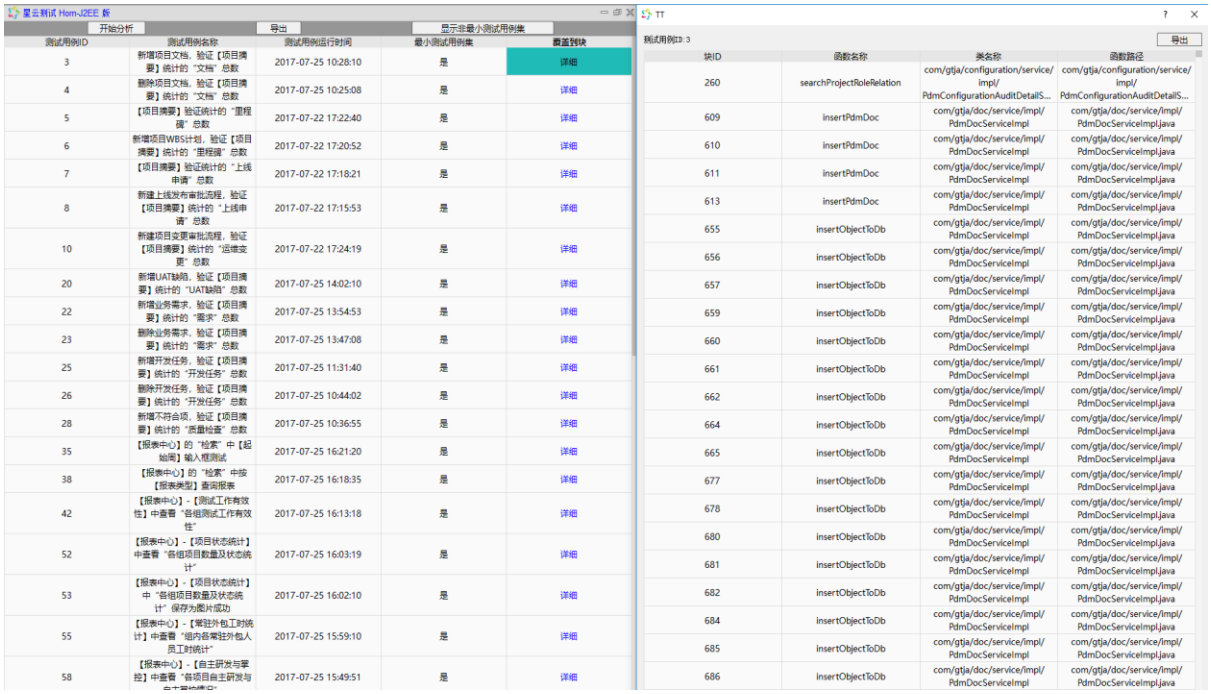


图 5.3.4 星云测试最小测试用例集

第六章 精准测试支持不同剖面的分析报告

在时间有限，经费有限，资源有限的情况下，我们既要考虑测试是否充分，也要顾及时间、人员、设备配置等限制条件。精准测试可以支持企业不同剖面的软件质量分析需求。

6.1 详细的测试总结报告内容

1) 测试资源分析：多少人，多长时间、整体测试有效率及执行率

2) 测试结果分析：描述需求的测试结果，系统实现了哪些功能点，哪些还没有实现

3) 缺陷情况分析：缺陷复现、缺陷处理、缺陷数量、属性、状态分布，缺陷预防、缺陷收敛度

4) 度量指标分析：测试覆盖率、测试执行率、测试执行通过率、测试缺陷解决率

5) 效率指标分析：进度偏离度、缺陷发现率、用例执行效率和质量等

6) 高风险识别与排序：注明当前项目中面临的最严重、最优先的问题

7) 整体评估：哪些功能已经实现，哪些功能还未实现，还遗留哪些问题，遗留缺陷分析

8) 优化建议：测试过程优化，从测试组的角度为测试工作提出建议

图 6.1 星云测试的差异报告分析

6.2 高级回归测试报告

对前期测试执行阶段发现的问题、缺陷集中的功能，业务比较重要且使用频繁的功能进行再次测试，确保系统上线后，已被修复的问题不会重新出现，重要的、高优先级的业务不会发生错误。

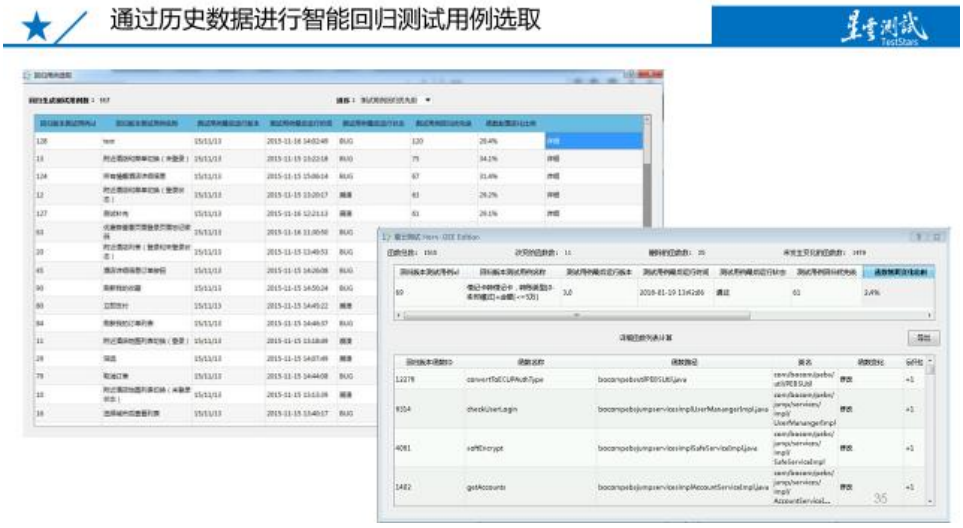


图 6.2 智能回归测试用例选取

6.3 测试用例库评估报告

软件测试的主要工作，是把软件需求映射为软件测试。测试用例是软件测试全过程的核心，也是测试执行环节的基本依据。精准测试充分满足软件测试执行过程中的各种指标：

1) 测试用例执行进度

2) 测试用例通过率

3) 测试用例颗粒度分析

4) 识别无用的测试用例

5) 识别冗余的测试用例

6) 从侧面提供增添新测试用例的依据

7) 从侧面提供调整测试用例库结构的依据

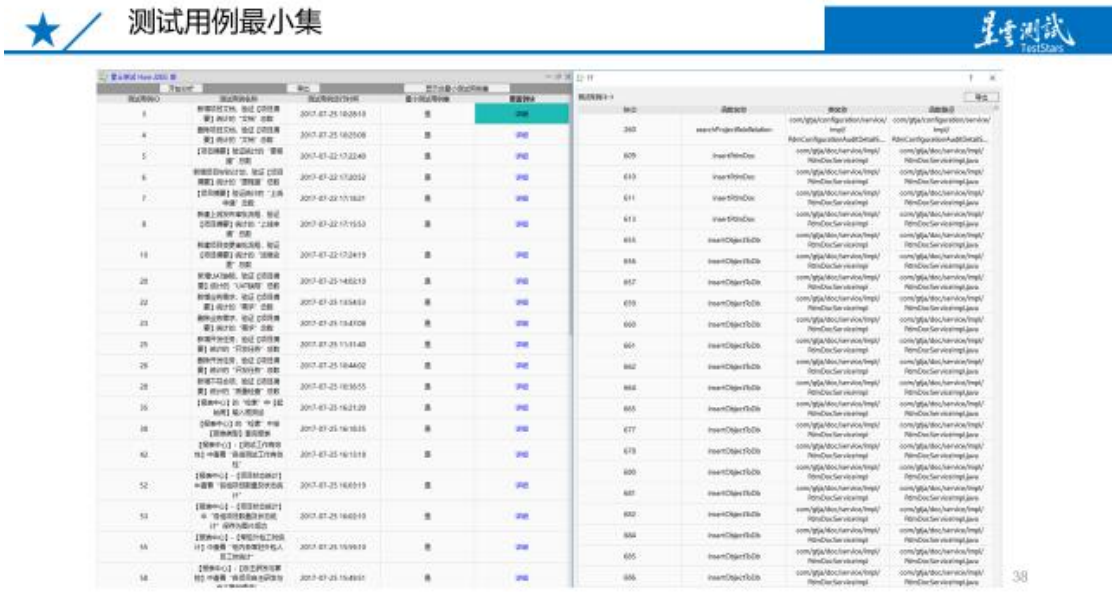


图 6.3 测试用例最小集

6.4 精准测试的 VIP 企业内网私有云可信化报表

星云精准测试提供多个剖面的高可靠性测试质量进度追踪报表。当客户端录入测试用例并采集

数据后，用户内网 web 端将产生实时、详实的测试数据分析报表。

该报表与普通的测试管理系统不同：普通的测试管理系统人为录入数据的情况比较多，使得数据状态的真实性没办法确切保证。精准测试提供的报表，底层数据来自于执行测试用例时候代码数据的采集，通过专用底层接口上传，完全无法进行数据调整或者篡改和伪造。

1) 通过浏览器登录测试系统，选择需要跟踪的项目，就可以实时对整个测试的质量、进度、人员进行精准的分析和管理。

2) 企业内网云端管理系统展示的数据基于精准测试数据的分析，所有数据原生精确，支持移动测试+本地测试。

3) 测试团队、开发团队、甲方负责人等多种角色都可以登录系统，从各个层面对测试、软件质量进行分析。

图 6.4 项目汇总展示

6.4.1 精准测试的 VIP 企业内网私有云-测试效率的直观展示

精准测试报告可直观分析每天的测试效率，通过代码模块和复杂度关系图，看到函数群落测试情况分布及趋势，可直观精准识别系统测试所处阶段。

每日增长覆盖率报表：管理者可以清晰的看到整个团队的效率趋势变化，比如刚开始测试的时候覆盖率增长快，到了黑盒测试瓶颈点上升就很慢了。这时候精准测试技术就开始发力，可以清晰地看到它在弥补效率损失方面的优势。

覆盖率和复杂度报表：可以很直观地看到测试的质量深度，例如在测试不充分的时候，复杂度高

的模块通常覆盖率都比较低，统计点分布自一个左上角的区域（表示高复杂度+低覆盖率），而当测试深入进行，这些点就会向右侧移动。管理者可以非常直观的看到系统测试的充分程度和上线的质量把握。

图 6.4.1-1 覆盖率每日增长趋势图与黑盒测试瓶颈

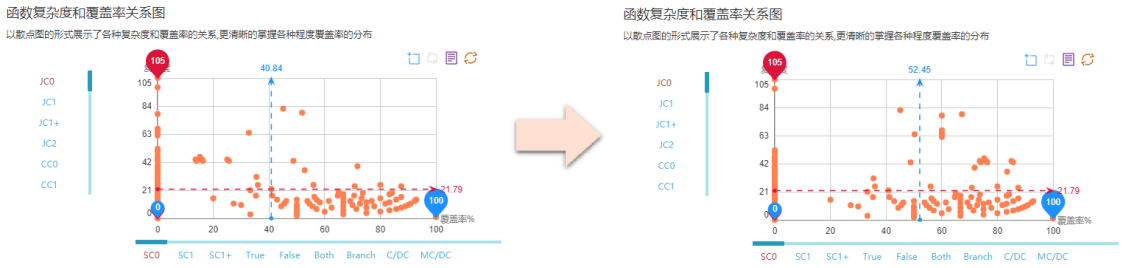
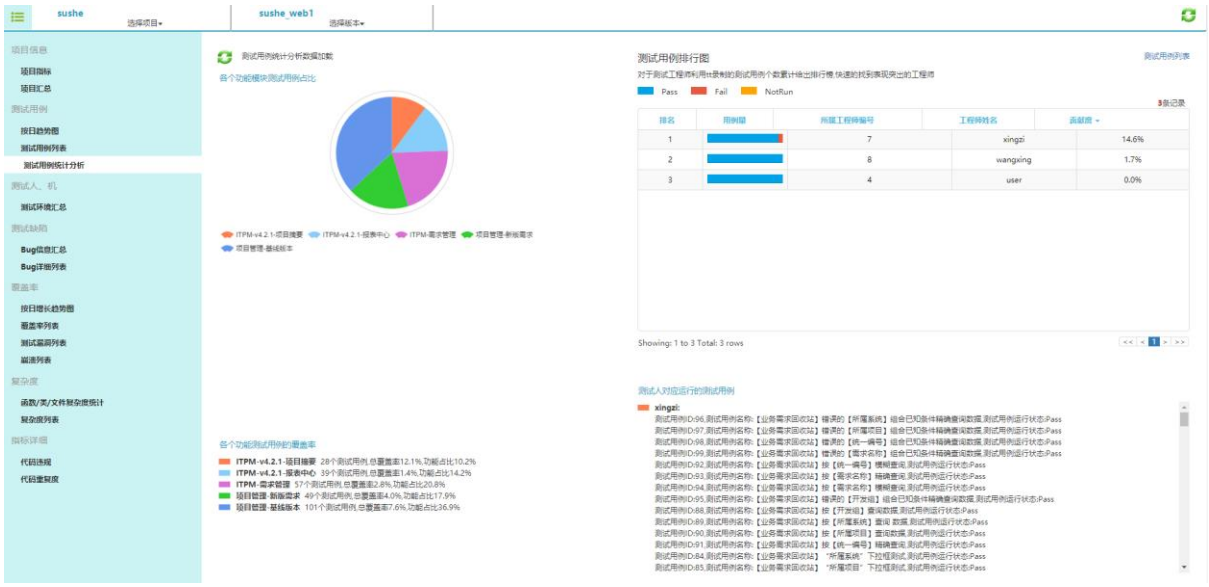


图 6.4.1-2 测试效率换档点与测试深度趋势观察表

6.4.2 精准测试的 VIP 企业内网私有云-测试用例排行图

测试用例排行分析报表：可直观展示参与测试工程师所执行的用例数、通过率和缺陷率，真实记录并分析每个测试用例的实效性。星云精准测试-测试工程师实效精准分析系统，将参与的测试工程师所执行的用例从逻辑覆盖映射到代码覆盖，真实记录并分析每个测试参与者的工作实效。以逻辑覆

盖为基准的而不是用例数量为考核标准。



测试用例排行图 测试用例列表

对于测试工程师利用tt录制的测试用例个数累计给出排行榜,快速的找到表现突出的工程师

Pass Fail NotRun

共15条记录

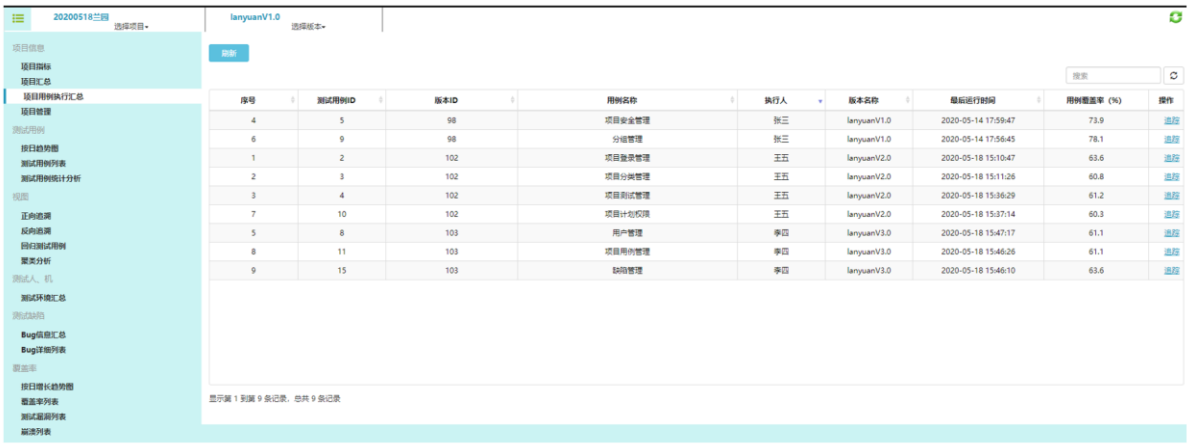
排名	用例量	所属工程师编号	工程师姓名	贡献度
1		79	测试a部014	38.6%
2		15	测试a部006	30.5%
3		10	测试a部001	26.6%
4		76	测试a部011	24.5%
5		14	测试a部005	23.7%
6		11	测试a部002	22.4%
7		77	测试a部012	16.5%
8		13	测试a部004	12.3%
9		12	测试a部003	10.9%
10		16	测试a部007	10.6%

Showing: 1 to 10 Total: 15 rows 10 records per page << < 1 2 > >>

图 6.4.2 测试用例排行图

6.4.3 精准测试的 VIP 企业私有云-测试用例双向追溯与覆盖率可视化

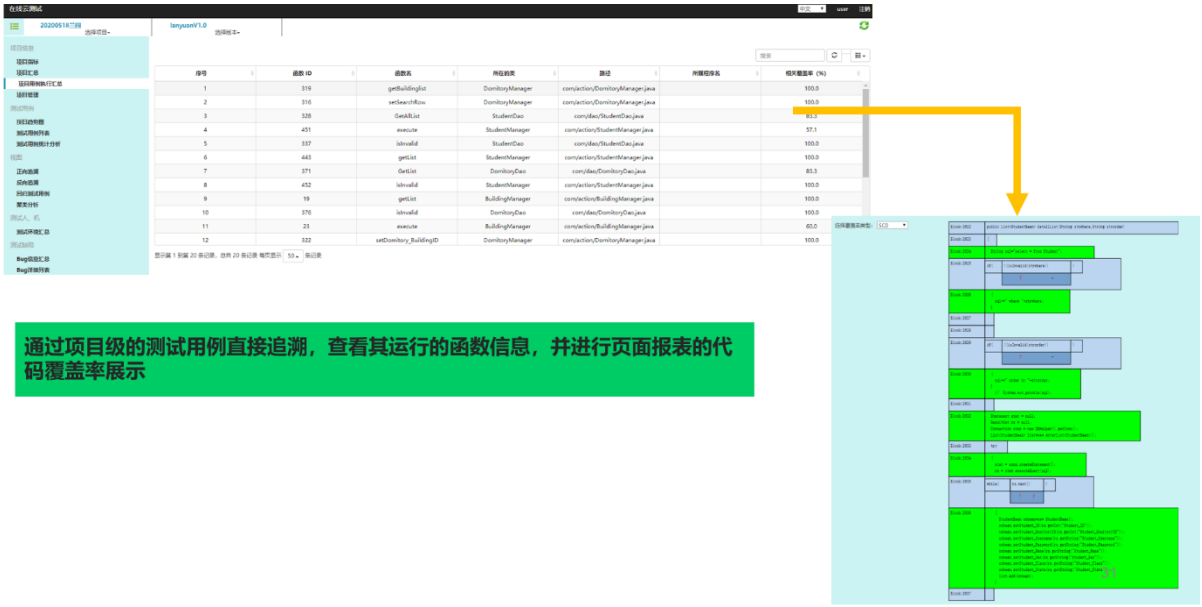
通过用户的内网精准测试 web 报表，可以追踪每个测试用例执行的覆盖率和执行的函数路径信息，那些没有真正执行的用例，将无法伪造其对应的覆盖率信息。



精准测试记录了每条测试用例在其执行的版本中的用例追溯信息，通过项目用例执行汇总对每条测试最后所执行的测试数据与所对应的版本进行追溯展示

图 6.4.3-1 测试用例双向追溯

追踪每个用例执行的函数信息以及具体的代码覆盖信息，在 web 端展示代码覆盖率视图，更具体的分析用例的执行情况。



通过项目级的测试用例直接追溯，查看其运行的函数信息，并进行页面报表的代码覆盖率展示

图 6.4.3-1 覆盖率可视化

6.5 精准测试在数字化转型中的作用

1、**完成数字化转型。**提高软件交付质效、实现快速迭代持续交付，有效呈现测试价值。

2、**培养业务和测试的“两栖专家”。**从不同角度提供有价值的数据依据，使测试团队既能熟悉业务

知识、业务场景，又具备较强的业务分析评估和整合创新的能力。

3、**数据化交付。**实现企业内部云平台建设、明确各工程活动环节的交付物和交付标准，并将质量验证标准、验证手段和监控工具嵌入流水线，保证各环节的有效性，对质量趋势进行提示和预警，及早发现缺陷，实现质量可视、过程可追溯、可审计。

4、**建立测试数据资源池，整合测试资产。**利用大数据、人工智能等技术建立质量智能分析模型等，为“产品质量智能分析平台”提供有效数据。

5、**减少因人员变动而产生的成本影响。**一般外包人员的流失率普遍为 20-40%左右，重新招聘和培养，将对项目进度及成本进度，造成很大影响。

第七章 精准测试企业级方案

7.1 DevOps 微服务架构下具有代码级穿透能力的精准测试

微服务是 DevOps 场景下热门的开发框架，在大型项目中被广泛采用。它把一个大型的单个应用程序和服务拆分为数十个的支持微服务，独立部署、互相隔离，通过扩展组件来处理功能瓶颈问题，比传统的应用程序更能有效利用计算资源。微服务之间无需关心对方的模型，它通过事先约定好的接口进行数据流转，使业务可以高效响应市场变化。

微服务一个明显的表象就是随着服务的增多，传统测试模式受到很大制约，无法有效进行下去，威胁到整体系统质量。所有 J2EE 代码层白盒采集工具都无法区分覆盖和具体功能的对应关系，只能以后台模式“笼统”的采集一个阶段的总的覆盖，无法满足对于 DevOps 下对于故障定位、深度测试分析以及敏捷发布算法的要求。星云测试的分布式微服务精准测试解决方案，是目前

市场上唯一可达到在复杂分布式系统中，跨多个服务器进行代码白盒级分析、实现请求分布式追踪的测试平台。

精准测试通过分布式追踪系统支持分布式系统的用例和代码的关联，首先在浏览器的请求端注入一个用户标签，通常这个用户标签和精准测试客户端的登录用户是一致的。这个用户标签会一直带到实际请求的执行线程里面，然后这个执行线程里面配合插装的代码，发出来的信息就知道是哪个用户对应的哪个用例执行的代码了。同时分布式系统的话，如果有后续节点的调用这个标签会继续根据支持的协议附加用户标签信息往后传递。

我们目前支持的协议包括 httpclient、dubbo、spring cloud、web service 以及消息队列也支持线程池内部不同线程的传递等等，对于我们附加的标签，可以在这些协议上继续往后传递，通常是在协议上附加我们的用户标签信息来实现。对于用户自定义的协议或者小众的开发协议，用户可以基于我们的接口自定义。

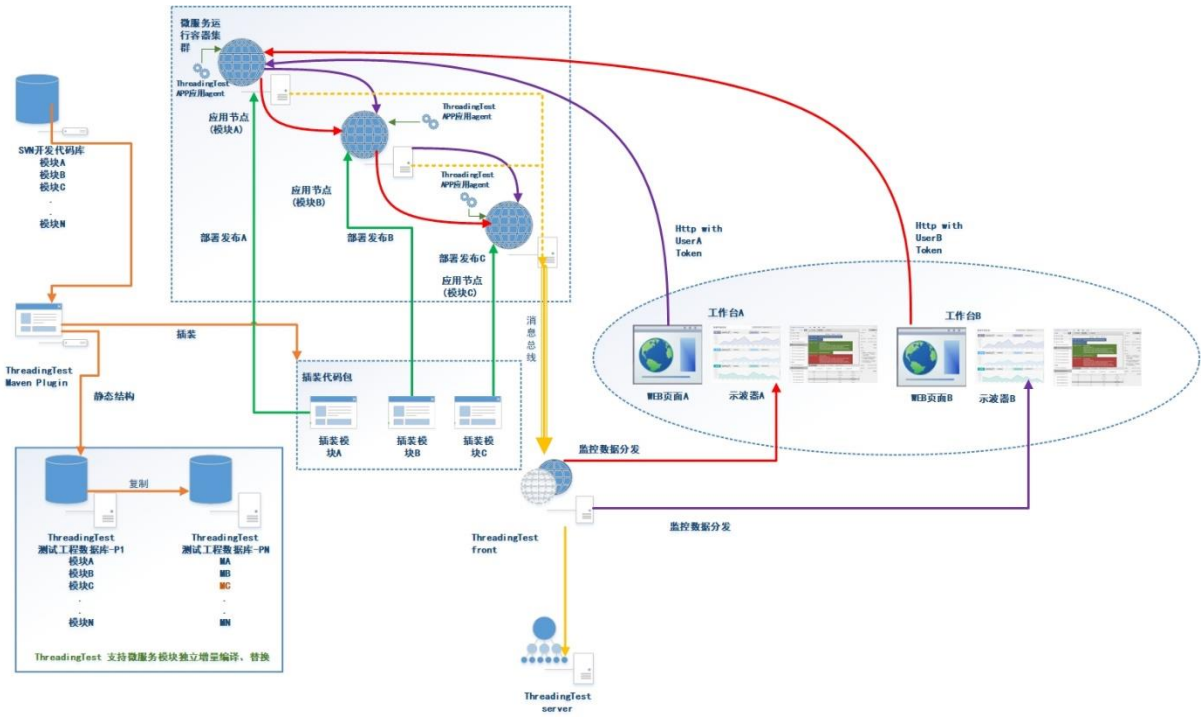


图 7.1-1 微服务穿透方案

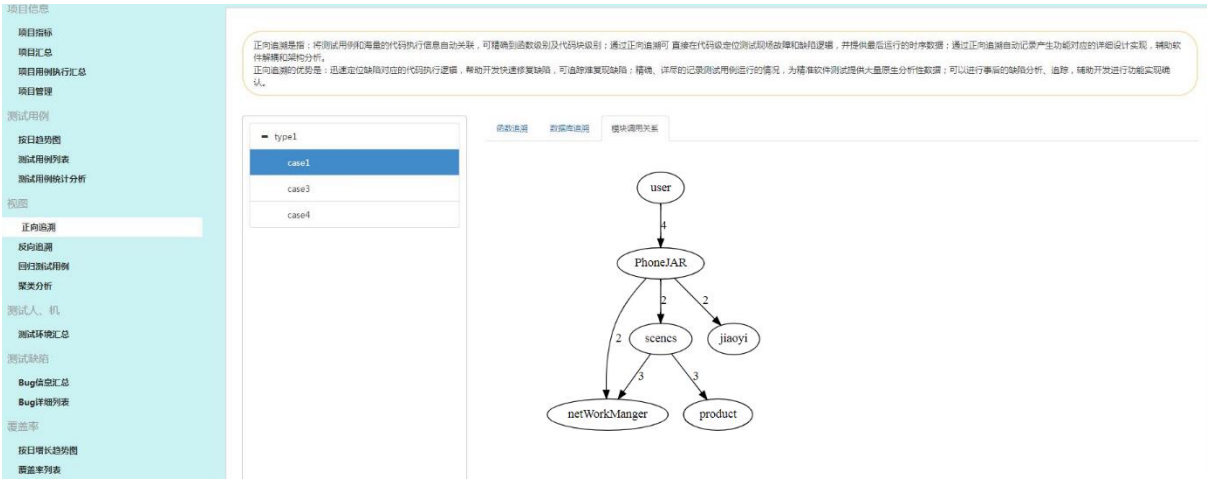


图 7.1-2 微服务模块调用关系图

7.2 “静默式”精准测试，让企业无缝完成黑盒测试的升级对接

精准测试最核心的技术关键就是：用例和相关执行代码之间有很强的对应和追溯关系。这个强追溯关系的建立，通过精准测试专属客户端上的“软件示波器“，用人工点击开始和结束按钮来标记测试用例的执行，进而确定对应代码执行路径的边界。

目前很多公司内部都有开发测试管理系统或者类似于 JIRA 这样的通用产品来管理和执行用例，如果同步使用精准测试客户端，则有指令重复之嫌。因此，星云精准测试做了具有深远意义的客户化改进-“静默式”精准测试。

大部分的测试管理系统，测试过程中会点击开始和结束按钮用于界定测试用例的边界，这和精准测试中的示波器操作是一致的。我们做了一个接口，使测试人员在“用例管理系统”中选择某个用例执行到切换到另外一个用例的过程，和精准测试示波器进行无感对接，测试工程师不用登录到精准测试客户端，即可实现“静默式”精准测试的数据采集。“测试管理系统”的用例会通过这个内部接口，自动的和精准测试平台建立连接。

图 7.2 中是一个基于 excel 用例管理的对接的方案，测试工程师只要在用例的表格上用鼠标点一下当前要执行的用例条目，通过 VBA 接口就可以通知示波器当前某个用例即将开始测试，使用起来非常简单。

精准测试也可以为 JIRA 这种企业级的管理平台提供插件进行对接。如果是用户自己开发的测试管理系统，也可以根据精准测试提供的接口文档做相应的平滑对接。

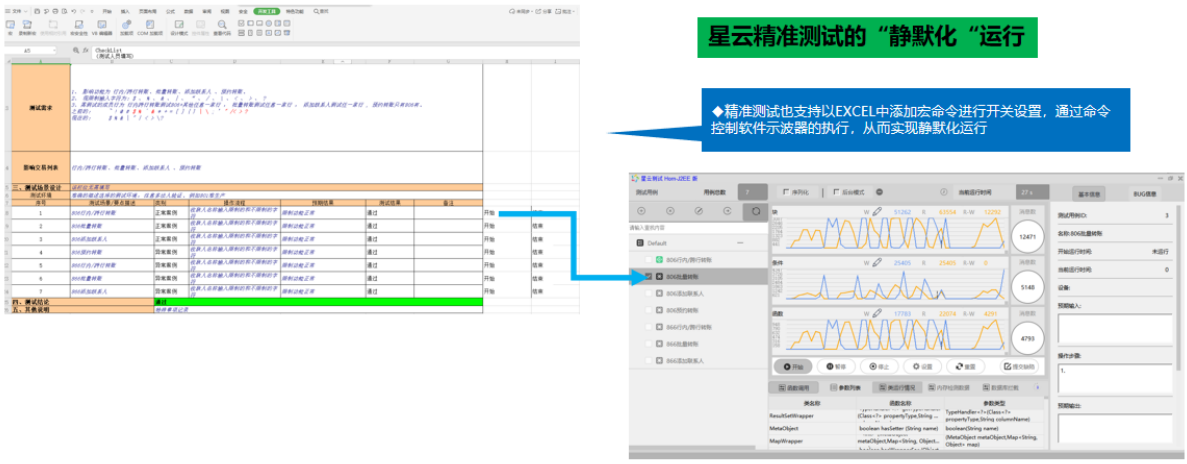


图 7.2 与 excel 表格对接

7.3 为自动化测试装上精准测试的“翅膀”

现代的专业软件测试中心，随着项目迭代，通常针对每个系统构建了大量的自动化测试用例集，而启动一次全量的自动化测试以 CI 级触发，使之大比率通过，非常困难。测试工程师们常常需要投入很高的成本，把大量精力花在自动化用例失败排查上面，然而有效发现 BUG 的概率依然很低。在反复排查无果、心神俱疲的情况下，很多单位几乎对自动化产生绝望之心，视之为鸡肋，用之无用，弃之可惜，让测试中心极为头疼。

如何让自动化用例发挥它们应有的效用，让 QA 工作不那么沉重呢？星云测试针对这一难题，进行了精准测试与自动化测试无缝对接的技术方案研发。经过大量企业实施与验证，精准测试的数据流最终可以“无感”对接到自动化测试中，极大扩展了自动化测试的优势，彻底改进了自动化测

试变更管理难的短板。

这一技术方案的推出，就像给自动化测试装上“精准测试”的眼睛和翅膀，瞬间就具备了多种飞跃性功能。比如：

1) 自动化测试用例与源码自动建立关联

2) 同步进行智能回归用例选取

3) 有效缩小自动化测试执行范围

4) 即时分析需要进行维护的测试用例集合

5) 全自动追踪每个测试用例的执行代码路径

6) 当自动化执行结束后可辅助直接定位自动化用例的代码出错点

7) 对自动化测试用例集进行分析，例如聚类分析，以及最小用例集合分析等

8) 对测试用例集的优化给出指导意见

9) 给出测试用例集运行的总体覆盖率信息

10) 协助有效的对用例集进行增补

11) 增量代码覆盖率分析等等。

精准测试系统提供标准接口，可以很容易的与企业原有的自动化平台进行对接，当执行完自动化测试后，可以得到每个自动化用例对应的代码逻辑信息、相应功能模块等，为自动化测试提供有效的价值叠加与放大。

7.3.1 精准测试提供 HTTP 请求中转台直连对接模式方案

精准测试提供 HTTP 请求接口给自动化或测试管理平台，通过 HTTP 请求将测试用例的创建、运行、结束与查看测试用例的覆盖情况命令，发给星云精准测试自动化平台的相应接口，进行测试

用例的创建录入与停止功能，并通过该接口进行覆盖率信息返回。

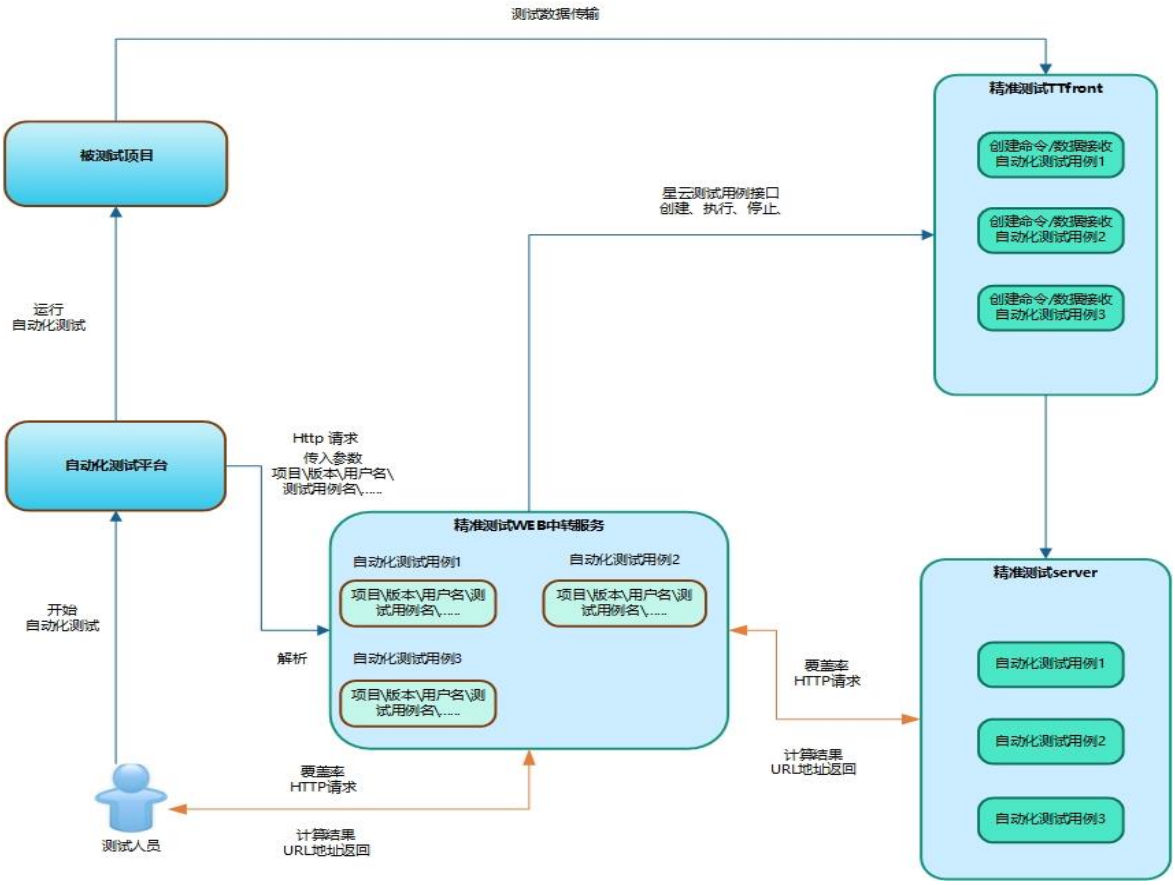


图 7.3.1-1 精准测试 HTTP 接口请求中转台直连对接模式

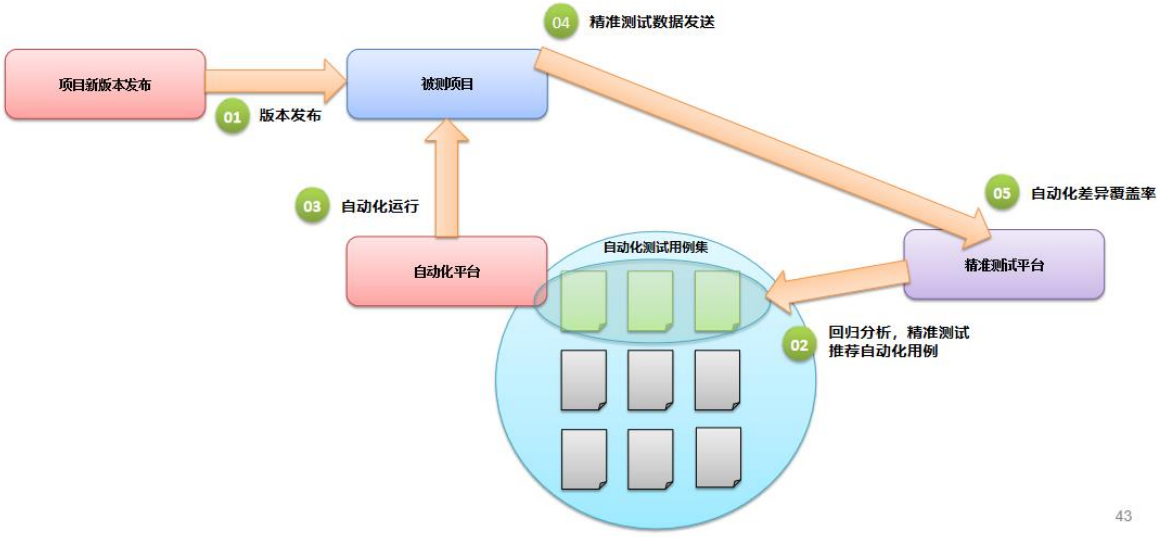


图 7.3.1-2 精准测试与自动化对接后推荐自动化用例场景

7.3.2 实现“手自一体”的高效解决方案

1) 精准测试将自动化测试脚本与“业务全景图”和测试用例库建立关联。

2) 在测试设计、测试执行、管理和质量监控等环节实现了“手自一体”的一体化管理理念。

3) 构建测试数据库，实现测试数据统筹分配和复用。

4) 组建“精准自动化测试执行云平台”，实现 24 小时测试目标。

7.4 星云测试插桩编译流程与 CI 集成

对于精准测试的插桩代码是直接静态植入了发布包，只能供测试使用，不能用户生产发布。通常精准测试与 CI 对接，每发布一个版本会产生两个分支一个是发布生产的分支，一个是发布给测试环境的分支，他们的代码基线是一致的，就是一个通过精准测试做了插桩，另一个没有做插桩，他们的功能和版本都是一致的。插桩后的代码运行后关联代码也是关联同样基线版本的代码就可以了。

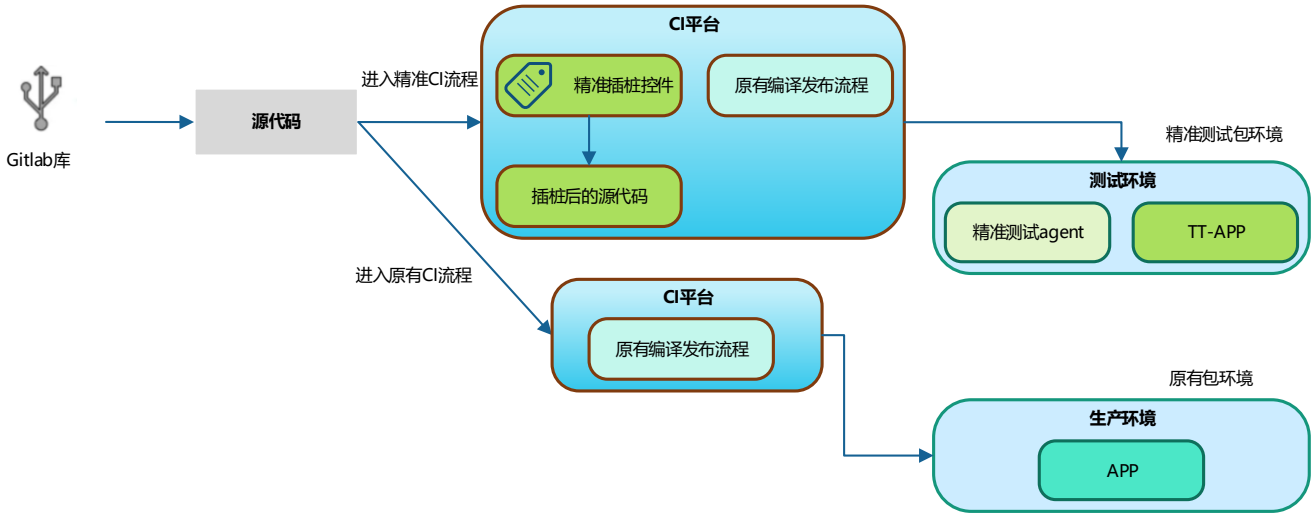


图 7.4 精准测试与 CI 对接方案

第八章 知识库累积

8.1 精准测试数据的价值

星云测试采集的测试数据和插装后分析到的静态结构信息，将作为大型企业系统大数据分析的基础数据。

星云精准测试-测试数据价值

（1）代码级的程序静态信息以及测试用例对应的海量动态测试的数据，这些多维度数据将作为大型企业系统大数据分析的基础数据。

（2）对本企业大量软件质量数据进行挖掘和分析，找到相关质量技术标准衡量的合理区间，避免常规错误。

（3）通过数据分析确定优异的开发方法和技术构件。

（4）通过质量大数据的分析结果，选择更加合理的技术方法，在设计阶段避免已知的缺陷。

8.2 精准测试智能回归测试用例智能选取

在大型软件的迭代过程中，测试最大的压力来自于回归测试，因为对于大型系统很难说得清楚新的变更可能会意外的影响那些其它功能。国际上有相关统计：每修改 6 行代码就会引入一个未知的缺陷。而要对回归范围进行分析通常必须是对系统很了解的架构师、高级开发人员或高级业务人员才能够分析，因为程序的功能和它的实现就像是一张蜘蛛网错综复杂，即使高级人员也很难分析的非常清楚。精准测试在记录了所有用例对应的代码逻辑的基础上，在新版本发布的时候通过分析用例执行的代码路径的变更范围，就可以自动计算出来回归用例的范围。它的算法逻辑和人的分析非常相似，但可以进一步提供非常高效率、海量数据的稳定分析输出，相当于把人的脑力算力转换成了计算机算

力。

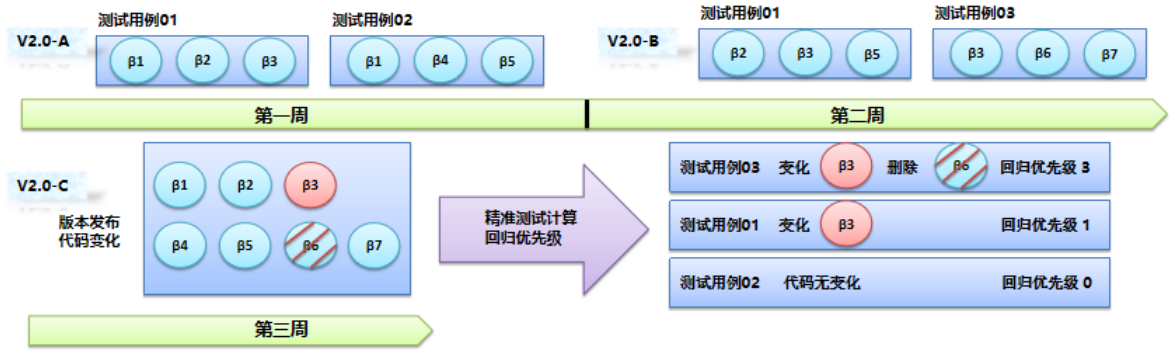


图 8.2-1 精准测试智能回归测试用例的选取

（1）适应快速的版本迭代周期，适应庞大的工程项目。

（2）在回归测试时，自动筛选测试用例，大大减少了回归测试的时间以及风险。

（3）降低了传统人工回归分析产生的测试盲点。

（4）精确计算回归用例的权重，测试人员在时间有限的情况下可以重点回归受改动影响最大的用例。

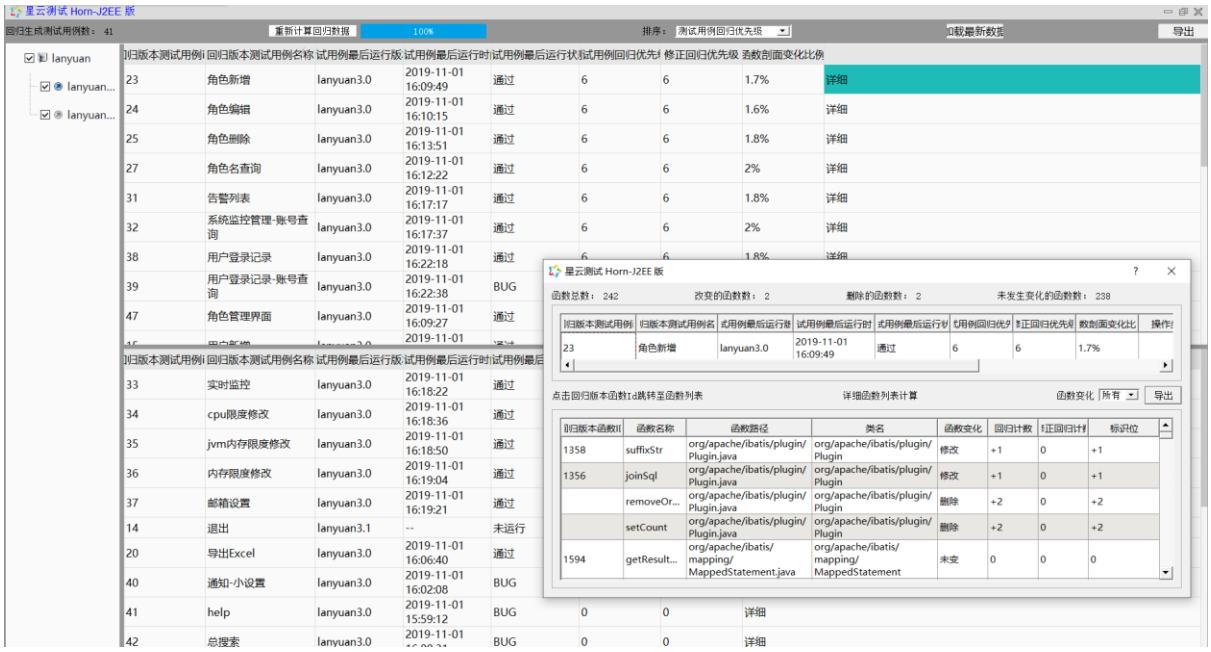


图 8.2-2 回归测试用例选取界面

8.3 精准测试在回归测试中的性能评估

回归性能，通常一般回归都是基于人对于系统的判断来做的。由人来进行回归用例集的判断，随着时间的延续，记忆将不可逆转地发生损耗并丢失，加之原团队人员的不断变更，老的系统维护越来越难，修改引入新的缺陷要越来越难风险。

通过星云精准测试企业离线平台，内置程序将持续、高效地全自动记录本企业自有的各系统测试用例和代码的关系数据，不用人工干预和记录。使用一段时间后，企业会得到越来越精确的数据，若加以有效利用，将发挥相关元数据及大数据的爆发性的价值。

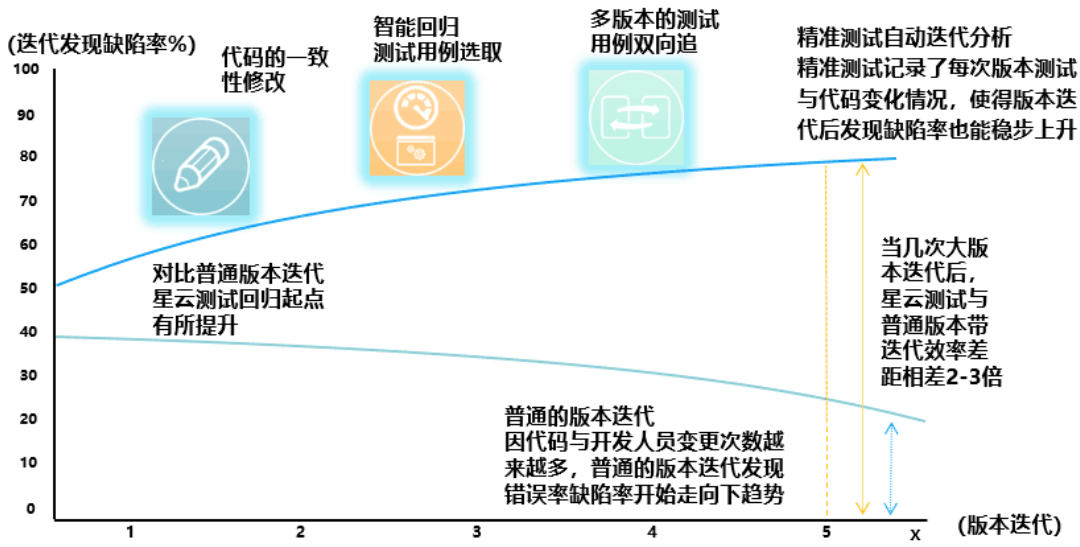


图 8.3 智能回归测试用例选取的性能评估

8.4 精准测试形成的测试资产

测试资产：即与测试相关的技术、文档、数据、脚本、管理、成本、进度、可用资源等。测试资产是测试工作过程中的产出，是项目的工作成果。建立测试资产库是测试资产复用的重要前提条件。

1、**经过实际验证，高效甄别出可复用的大量的测试用例和测试数据，可大幅缩短评审时间。**新项目

在测试过程中，如果需要测试用例或者测试数据，可直接在测试资产库获取，不必再花费大量的时间和人力重新编写。

2、测试资产库适用于维护型的项目。功能点经过修改后，通过测试资产的复用，只需要更新经过变动的功能点测试用例，其它的未经更改的功能点测试用例就可以直接使用。通过这样的测试资产的复用，可以大幅提高测试效率和测试质量，节约测试成本。

3、**测试资产库的建立有助于保护测试项目的劳动成果。**通过统一的途径收集和整理测试资产，既便于回溯本次项目的测试过程，又可以为其它项目提供参考和帮助，是测试资产复用的前提和保障。测试资产库在任何时候都处于最新和有效状态。

4、在测试可追溯的基础上，降低了跨业务、跨部门业务分析的复杂程度。可视化的动态业务全景图，为决策者、业务部门、技术人员的测试设计、执行和质量评估提供核心依据。

5、**建立有效测试质量指标评价体系，实现智能化测试方案设计。**积累自动分析用例集覆盖率、冗余度、回归权重等有效数据资产，实现“以最优用例，覆盖最全面功能”的目标。

6、**通过对历史数据的挖掘与分析，归纳出影响测试资源分配的关键因素。**通过对业务系统历史测试数据的挖掘和分析，归纳出被测系统的缺陷特征和识别趋势以及与之相关的关键因素，未来作为测试准入、准出判断的参考依据，为识别待测系统的投产风险提供依据，为实测系统上线决策和风险应对提供支持。

7、**具备成为“测试中台”的核心组件的条件。**星云测试旗下产品具有高度自研创新能力，星云可以根据用户的创新需求，做更为具体的行业整体解决方案。

下篇

Wings--企业级单元测试自动编码引擎

（2020 版白皮书）

第一章 Wings 企业级单元测试自动编码引擎诞生的背景

随着科技的飞速发展，软件系统越来越复杂，在系统测试阶段不断遇到的瓶颈，迫使行业逐步追根溯源到了单元测试阶段。软件缺陷发现得越晚，其处理费用就越呈几何激增，因此测试左移概念已经成为趋势。

单元测试面临的最大问题是单元测试用例编写工作量巨大，极端情况下与开发工作量比达到1：1，甚至更高，这使大部分开发团队要么主动忽视单元测试，要么象征性的走个流程。

如果可以让计算机先对被测试程序进行全局分析和深度理解，再由计算机进行全自动的完成单元测试编码，同时还能确保自动编写的代码无语法、语义错误的直接运行起来，这种用计算机智能算法全自动产生的测试编码去验证开发人员编写的源代码逻辑输入输出对错的高端测试模式，无疑是未来软件测试领域最为璀璨的“明珠”技术。

国外软件诸如 c++ test 完成了这个领域的初步技术探索，星云测试研发的 Wings（目前商用产品支持 c/c++程序）产品，则大踏步完成了整体技术跨越和多方商用落地验证。

Wings 可以对程序参数进行深度解析，比如 c++类、模板类、数组、结构体、指针、链表以及任意复杂结构的层级嵌套，同时对于面向对象的程序特性以及常用的容器库，能够完美识别和支持。对于一些 void*、函数指针、模板类等无法直接静态分析进行类型确定的特殊情况，均有基于人工智能的程序分析辅助进行类型确定。

Wings 在基于深度参数解析的基础上，对于全局范围的程序进行理解分析后，第一步 按照内置规则，自动化构建被测程序的输入用例代码；第二步 构建测试代码用于调用被测程序的源代码；第三步 构建被测程序输出断言，完成调用被测试程序单元的全部环境准备。这个构建速度非常快，可

以达到每分钟 100 万行左右的生成速度，编写的代码比程序开发人员手工编写的规范度高出一截，并确保 100% 的语法语义正确，免去大量的调试时间。

在驱动数据上，Wings 实现了驱动代码和数据的分离。Wings 基于深度参数解析基础上，可以根据参数的结构自动生成层级嵌套的测试数据结构，用图形界面可视化的展示给用户。用户只需要根据 Wings 提供的界面向导对测试数据进行填充即可，驱动程序会自动识别并读取这些数据，完成对被测试程序的调用。

Wings 还可以全自动生成参数捕获程序，并自动插装在被测试程序中。当被测试程序运行后，可以通过专用软件捕获程序中每个函数模块运行的具体参数值。Wings 的测试代码驱动自动生成和参数捕获，相当于完成了一种全智能的闭环测试验证体系。Wings 使测试数据不需要人工准备，只需要在前序轮次中通过参数捕获自动存储。若前序测试用例运行正常，那么这些数据都可以作为后续测试输入和进行校验的基础数据。

第二章 单元测试自动生成技术

2.1 测试左移后单元测试面临的问题

测试左移后，传统的单元测试一般面临很多问题，主要如下：

(1)传统程序级测试用例的编写会耗费开发人员大量的工时，比如 TDD 测试驱动开发里面的单元测试无法有效实施，导致所有测试几乎全部依赖于系统级黑盒测试。程序级测试用例的开发成本是功能实现代码本身时间至少为 1：1，绝大部分企业选择放弃开发程序级测试，而采用系统级测试方法。

(2) 需求发生变化导致程序实现发生变化后，程序集用例也需要发生变化，和自动化面临的问题一样，单元测试本身的可维护性问题导致投入是持续的而不是一次性的，会打消其企业应用的热情。

(3) 很多单元在未组装成系统的时候切入，如果需要进行测试需要进行大量的 mock 操作或者桩模拟，这个过程会造成单元测试的不精确性。

(4) 程序集测试数据量很大，全部需要用户来进行准备无法全自动从前序系统测试过程中获取。

针对以上问题，很多业内人士提出了很多办法，例如自动生成测试用例、自动构建测试驱动、模糊测试方法等诸多方式，但是实际开发中程序输入参数十分复杂，简单的输入已经不能满足，如何能够构建复杂的参数输入，是要解决的重要问题。因此，星云测试研发了 Wings 产品--单元级的自动编码引擎，它不仅能够自动构建测试驱动，还能处理十分复杂的大型程序参数，帮助企业能够更好的进行单元测试。

2.2 完成单元测试要素

构成单元测试的要素如下：

a. 测试数据

b. 测试驱动代码

例如针对以下程序，需要准备的测试输入以及测试代码

① 全局变量：c

② 参数：a、b

③ 预期输出：320 30

```
int c = 100;
int sum(int a, int b)
{
     return a + b + c;
}
int driver_sum()
{
     int a = 100, b = 20;
     c = 200;
     return sum(a, b);
}
TEST(test, driver_sum)
{
     EXPECT_EQ(320, driver_sum);
     EXPECT_EQ(30, driver_sum);
}
```

测试代码

2.3 构建测试数据和测试代码遇到的技术瓶颈

编写测试代码比较繁琐

开发编写驱动不难，但是浪费大量的时间，并且没有创造性。

编写代码是否有良好的代码规范

单元测试也需要一些规范支持，例如代码规范，注释规范等，有了这些规范能够为测试人员提供很好的功能测试用例设计的逻辑思考，也为其他开发熟悉代码提供极大的便利。

数据类型比较复杂

通常情况下，输入参数比较复杂，嵌套层析结构比较深入，例如类包含其他类等。

能否支持多组测试数据

一般代码中每个循环、分支判断以及输入输出都有可能产生缺陷。因此单元测试需要很多用例，满足不同的条件分支，达到满足覆盖率。

第三章 Wings 的基本架构介绍

3.1 Wings 测试用例驱动自动生成技术的特性

- Wings 是智能的全自动测试用例驱动构建系统，能够在很短时间内完成对大型复杂程序的自动解析、构建。每分钟达到 100 万行代码的生成速率。
- 可以将任意复杂类型逐步分解为基本数据类型，例如结构体嵌套，复杂类对象，c++标准容器，自定义模板类等。
- 支持多层次的可视化的数据表格来对变量类型进行赋值，无需关注驱动程序本身。数据表格可以表达任意深度和多层次的数据关系，用户只需要关注表格数据，完成对输入数据的校对。
- 能够区分系统数据类型和用户自定义类型，对于复杂的约定俗成系统类型可由用户自定义扁平式赋值模板，例如 std::vector 类型等，内部集成常用系统类型的模板。

3.2 Wings 的技术架构介绍

Wings 的技术架构：首先 Wings 利用代码静态分析技术，提取被测程序的主干信息并保存到 Program Structure Description（PSD）结构中，PSD 结构中主要存储函数的参数、全局变量以及返回值的信息，类的成员变量，成员函数，以及访问权限等信息。利用存储的信息，依据一定的编写规则，自动构建测试驱动、googletest 期望断言、测试输入、参数捕获等代码和值。

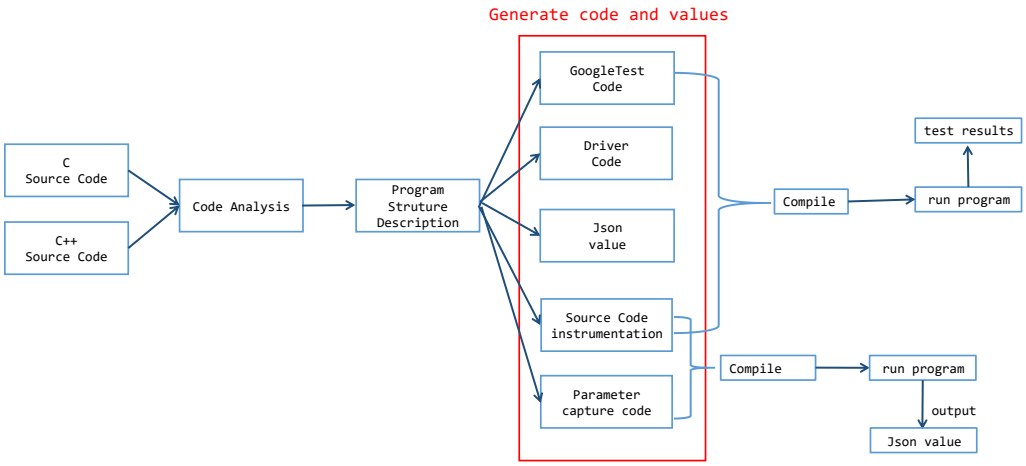


图 3.2 Wings 总体架构图

上述 Wings 的总体架构图，说明了 Wings 构建代码与测试输入的具体过程，整个核心技术是通过编译底层技术获取程序的信息，然后按照一定的规则构建需要的数据。

第四章 程序结构信息描述

程序结构信息（Program Structure Description）是指对源程序进行提取后的描述信息，主要包括类名、命名空间、类的成员变量信息、类的函数信息，以及各种类型信息等（程序结构信息，下文简称“PSD”）。描述信息的保存 C 语言是以一个文件为单元存储，C++提取所有的类信息存储在一个文件中。

Wings 的主要特性是能够对复杂的类型（结构体、类、模板类等）进行逐层展开

分解到最基本数据类型（char、int、string 等）。

PSD 结构存储在 XML 文件中，不同的 XML 文件存储不同的描述信息。Wings 提取的程序结果都存储在 Wingsprojects 文件下的 funxml 与 globalxml 文件夹中，其中 funxml 文件中主要存储的文件以及作用如下：

RecordDecl.xml：结构体，联合体与类描述信息。

ClassTemplateDecl.xml:模板类描述信息

EnumDecl.xml:枚举描述信息

funcPoint.txt： 存储函数参数为函数指针的分析结果。

funcCount.txt：存储分析到的全部函数，参数个数，参数类型信息。

void.txt：存储函数参数为 void*的分析结果。

filename.xml：存储 c 语言文件的信息。

注：具体文件的描述信息，参看附录 A。

针对复杂类型，例如结构体类型 location_s，成员变量中除了基本数据类型之外，还存在包含结构体类型的情况，如下所示的代码中，location_s 中包含 coordinate_s 结构体，以及 FILE 等类型的信息，针对不同的类型进行标记区分。

源代码如下：

```
typedef struct coordinate_s{
    void(*setx)(double, int);
    int mInt;
    char *mPoi;
    int mArr[2][3];
    void *vo;
} coordinate;

typedef struct location_s {
    int **mPoi;
    coordinate *coor;
    FILE *pf;
    struct location_s *next;
}location;

coordinate *coordinate_create(void);
int coordinate_destroy(location *loc,size_t length,char *ch);
void func_point(double param1, int param2);
```

生成对应的 PSD 结构信息如下图 4：

```
<coordinate_s StructOrUnionRecord="Struct" containFunctionPointType="TRUE" containVoidPointType="TRUE">
    <coordinate_s::setx baseType1="FunctionPointType" type="ZOA_FUNC" returnType="void" parmType0="double" parmType1="int" parmNum="2" />
    <coordinate_s::mInt baseType1="BuiltinType" type="ZOA_INT" />
    <coordinate_s::mPoi baseType1="PointerType" type="ZOA_CHAR_S" />
    <coordinate_s::mArr baseType1="ArrayType" RowSize="2" baseType2="ArrayType" ColumnSize="3" type="ZOA_INT" />
    <coordinate_s::vo baseType1="PointerType" type="ZOA_VOID" />
</coordinate_s>
<location_s StructOrUnionRecord="Struct">
    <location_s::mPoi baseType1="PointerType" baseType2="PointerType" type="ZOA_INT" />
    <location_s::coor baseType1="PointerType" type="StructureOrClassType" name="coordinate_s" NamespaceName="coordinate_s" />
    <location_s::pf baseType1="PointerType" type="StructureOrClassType" name="_iobuf" NamespaceName="_iobuf" SystemVar="_iobuf" />
    <location_s::next NodeType="LinkNode" baseType1="PointerType" type="StructureOrClassType" name="location_s" NamespaceName="location_s" />
</location_s>
<coordinate_create FunctionAttribute="SC_None" parmNum="0">
    <returnType returnType="coordinate *" baseType1="PointerType" type="StructureOrClassType" name="coordinate_s" NamespaceName="coordinate_s" />
</coordinate_create>
<coordinate_destroy FunctionAttribute="SC_None" parmType0="location *loc" parmType1="size_t length" parmType2="char *ch" parmNum="3" globalType="int
coordinate_destroy(location *loc,size_t length,char *ch)">
    <loc baseType1="PointerType" type="StructureOrClassType" name="location_s" NamespaceName="location_s" />
    <length baseType1="BuiltinType" type="ZOA_UINT" />
    <ch baseType1="PointerType" type="ZOA_CHAR_S" />
    <returnType returnType="int" baseType1="BuiltinType" type="ZOA_INT" />
</coordinate_destroy>
<func_point FunctionAttribute="SC_None" parmType0="double param1" parmType1="int param2" parmNum="2">
    <param1 baseType1="BuiltinType" type="ZOA_DOUBLE" />
    <param2 baseType1="BuiltinType" type="ZOA_INT" />
    <returnType returnType="void" baseType1="BuiltinType" type="ZOA_VOID" />
</func_point>
```

图 4：PSD 描述信息

C++的主要表示类型是类，因此测试是 C++以一个类为单元做测试，类主要包括类的成员变量名以及类型信息，成员变量的访问权限信息。类的成员函数分为构造函数、内联函数、虚函数等，成员函数的参数信息以及类型信息等。具体描述信息可以登陆 www.codeWings.net 下载 Wings 试用版本进行学习。

第五章 Wings 构建程序描述

Wings 通过获取到的存储信息，依据一定的编码规则，构建不同的代码程序，如驱动程序、期望断言程序、参数捕获程序等。

5.1 驱动程序构建

驱动程序指单元测试代码，Wings 针对函数参数的类型，自动完成单元测试代码的编写。为了更详细的说明驱动程序，以一个 C++类作为具体的例子说明。

类的声明如下：

```
class BlockBuilder {
public:
    explicit BlockBuilder(const Options* options);
    BlockBuilder(const BlockBuilder&) = delete;
    BlockBuilder& operator=(const BlockBuilder&) = delete;
    void Reset();
    void Add(const Slice& key, std::string & value);
    Slice Finish();
    size_t CurrentSizeEstimate() const;
private:
    const Options* options_;
    std::string buffer_;
    int counter_;
    bool finished_;
};
```

针对如上 BlockBuilder 类，构建一个对应的驱动类 DriverBlockBuilder，驱动类中主要包含构造函数、析构函数、每个成员函数的驱动函数以及返回值函数。为了避免类中重名函数，对写入 PSD 需要对应生成驱动的函数进行顺序编号。

驱动类声明：

```
class DriverBlockBuilder {
public:
    DriverBlockBuilder(Json::Value Root, int times);
    ~DriverBlockBuilder();
    int DriverBlockBuilderReset1(int times);
    int Reset1Times;
    int DriverBlockBuilderAdd2(int times);
    int Add2Times;
    int DriverBlockBuilderFinish3(int times);
    void ReturnDriver_Finish3(class Wings::Slice returnType);
    int Finish3Times;
    int DriverBlockBuilderCurrentSizeEstimate4(int times);
    void ReturnDriver_CurrentSizeEstimate4(size_t returnType);
    int CurrentSizeEstimate4Times;
private:
    Wings::BlockBuilder* _BlockBuilder;
};
```

在上述驱动类中，构造函数的作用是构造 BlockBuilder 的对象，用构造的对象，调用测试 BlockBuilder 中的成员函数。析构函数的作用是释放构建的对象。

构造函数与析构函数：

```
DriverBlockBuilder::DriverBlockBuilder(Json::Value Root, int times){
  Json::Value _BlockBuilder_Root = Root["BlockBuilder" + std::to_string(times)];
  /* options_ */
  Json::Value _options__Root = _BlockBuilder_Root["options_"];
  int _options__len = _options__Root.size();
  Wings::Options* _options_ = DriverstructOptionsPoint(_options__Root, _options__len);
  /* buffer_ */
  std::string _buffer_= _BlockBuilder_Root["buffer_"].asString();
  /* counter_ */
  int _counter_ = _BlockBuilder_Root["counter_"].asInt();
  /* finished_ */
  bool _finished_;
  int _finished__value_ = _BlockBuilder_Root["finished_"].asInt();
  if (_finished__value_ == 0) {
    _finished_ = true;
   }
  else {
    _finished_ = false;
  }
  _BlockBuilder = new Wings::BlockBuilder(_options_, _buffer_, _counter_, _finished_, false);
}
DriverBlockBuilder::~DriverBlockBuilder()
{
    if (_BlockBuilder != nullptr) {
        delete _BlockBuilder;
}
}
```

每个成员函数对应生成自己的驱动函数。类中的 Add 函数对应的驱动函数如下。

Add 驱动函数：

```
int DriverBlockBuilder::DriverBlockBuilderAdd2(int times)
{
    Add2Times = times;
    const char* jsonFilePath = "drivervalue/BlockBuilder/Add2.json";
    Json::Value Root;
    Json::Reader _reader;
    std::ifstream _ifs(jsonFilePath);
    _reader.parse(_ifs, Root);
    Json::Value _Add2_Root = Root["Add2" + std::to_string(times)];
    /*It is the 1 global variable: count    Add */
    int _count = _Add2_Root["count"].asInt();
    count = _count;
    /*It is the 1 parameter: key Add2
     * Parameters of the prototype:const Wings::Slice &key */
    Json::Value _keykey_Root = _Add2_Root["key"];
    /* data_ */
    char* _keydata_;
    {
        std::string _keydata__str = _keykey_Root["data_"].asString();
        _keydata_ = new char[_keydata__str.size()];
        memcpy(_keydata_, _keydata__str.c_str(), _keydata__str.size());
    }
    /* size_ */
    unsigned int _keysize_ = _keykey_Root["size_"].asUInt();
    Wings::Slice _key(_keydata_, _keysize_, false);
    /*It is the 2 parameter: value    Add2
     * Parameters of the prototype:const std::string &value  */
    string _value = _Add2_Root["value"].asString();

    //The Function of Class    Call
    _BlockBuilder->Add(_key, _value);
    return 0;
}
```

构成以上驱动函数的主要包括全局变量、参数、调用被测函数的构建。

以上是针对 BlockBuilder 类的驱动类的主要信息，而构建的程序遵守 google 的编码规范。一些命名规则如下：

Wings 生成的驱动代码，存储在 drivercode 文件夹中。

（1）driver.cc 与 driver.h，针对程序中使用到的一些公共函数以及头文件。

（2）同一个结构体或者联合体，可能会作为多个函数的参数使用，为了避免代码的重复，Wings 针对所有的结构体和联合体的不同类型，封装成不同的驱动函数或者参数捕获函数。

driver_structorunion.cc 存储结构体驱动函数的代 driver_structorunion.h 对应的头文件。

结构体实现函数的命名规则为：DriverStruct+结构体名字+类型，其中 Point 代表一级指针或者一维数组，PointPoint 代表二级指针或者二维数组使用。

源文件的命名规则为：driver_+源文件名/类名+.cc

例如：driver_nginx.cc 或 driverBlockBuilder.cc

驱动函数的命名规则：Driver_+函数名+（编号）

例如：Driver_ngx_show_version_info(void);

DriverBlockBuilderAdd2(int times)

（3）返回值的打印输出

返回值的打印输出函数命名规则：Driver+Return+Print_+函数名。

例如：DriverReturnPrint_ngx_show_version_info();

（4）用户源代码中的 main 函数自动进行注释，重新生成一个 main 函数文件，来进行测试。Wings 会生成驱动 main 的主函数文件为:gtest_auto_main.cc

Wings 主要针对参数进行逐层展开，解析到最底层为基本类型进行处理。驱动的赋值部分，主要就是针对基本类型进行处理。（注：特殊类型，比如 FILE 等，后面会详细讲解如何赋值）

以 int 类型举例，一般程序构成 int 的主要组成大概包括以下情况：

```
int p; int *p; int **p; int ***p;
int p[1]; int p[2][3]; int p[1][2][3];
int(*p)[]; int(*p)[][3]; int *(*p)[]; int (**p)[];
int *a[]; int **a[]; int *a[][3]; int (*a[])[];
```

Wings 会针对基本类型的以上 15 种类型，进行不同的赋值。

构建完驱动程序之后，要对驱动程序进行运行，Wings 构建 googletest 的框架，来进行测试。下面我们将针对期望断言的 googletest 程序进行构建。

5.2 googletest 程序的构建

在构建完单元测试驱动程序之后，Wings 将调用 googletest 的框架，完成对返回值的期望值验证代码，并且输出测试结果。

Wings 运行单元测试过程中，会构建返回值的保存代码，将程序的返回值结果进行存储，然后读取用户输入的预期值，进行结果对比，判断是否通过。

针对具体的类，对应生成 gtest 类，每个 gtest 类中对每个函数构建期望代码。例如针对 BlockBuilder 类，生成的 gtest 类为 GtestBlockBuilder。

GtestBlockBuilder 类的声明：

```
class GtestBlockBuilder : public testing::Test {
protected:
    virtual void SetUp()
    {
        const char* jsonFilePath = "../drivervalue/RecordDecl.json";
        Json::Value Root;
        Json::Reader _reader;
        std::ifstream _ifs(jsonFilePath);
        _reader.parse(_ifs, Root);
        driverBlockBuilder = new DriverBlockBuilder(Root, 0);
    }
    virtual void TearDown()
    {
        delete driverBlockBuilder;
    }

    DriverBlockBuilder* driverBlockBuilder;
};
```

BlockBuilder 类中的每个函数对应一个 gtest 函数，每个函数负责调用 DriverBlockBuilder 编写的驱动函数，对于包含返回值信息的函数，则对应生成具体的期望值对比。

期望对比函数 CurrentSizeEstimate：

```
TEST_F(GtestBlockBuilder, DriverBlockBuilderCurrentSizeEstimate4)
{
    const char* jsonFilePath = "drivervalue/BlockBuilder/CurrentSizeEstimate4.json";
    Json::Value Root;
    Json::Reader _reader;
    std::ifstream _ifs(jsonFilePath);
    _reader.parse(_ifs, Root);
    for (int i = 0; i < BLOCKBUILDER_CURRENTSIZEESTIMATE4_TIMES; i++) {
        driverBlockBuilder->DriverBlockBuilderCurrentSizeEstimate4(i);
        Json::Value _CurrentSizeEstimate4_Root = Root["CurrentSizeEstimate4" + std::to_string(i)];
        /* return */
        unsigned int _return_actual = _CurrentSizeEstimate4_Root["return"].asUInt();
        /* return */
        unsigned int _return_expected = _CurrentSizeEstimate4_Root["return"].asUInt();
        /* return_expected */
        EXPECT_EQ(_return_expected, _return_actual);
    }
}
```

最后调用自动构建的 main 函数运行整个单元测试过程。

5.3 参数捕获程序构建

参数捕获是指在程序运行过程中获取程序的变量信息，主要包括函数的参数、全局变量、返回值等。Wings 自动构建获取参数的程序，利用插装技术，将构建的捕获函数插入源代码中对应的位

置，将获取的具体信息，写入值文件，可以将获取的数据作为单元测试的输入，在代码发生变更后，利用相同的输入判断是否得到相同的输出，进行回归测试。

Wings 的参数捕获代码存储在 paramcaputrecode 文件夹中。其中命名规则同驱动格式一样，将所有的 driver 替换为 param 即可。Wings 针对每个类生成一个对应的参数捕获类，而参数捕获类中针对每个函数生成对应的捕获参数、全局变量以及返回值的函数。c++ 中类的成员变量是私有，无法从外部获取，Wings 利用插桩技术，对每个类插入一个捕获函数，来获取类的私有成员变量。

参数捕获类 ParamCaptureBlockBuilder：

```
class ParamCaptureBlockBuilder
{
public:
  ParamCaptureBlockBuilder();
  ~ParamCaptureBlockBuilder();
  void ParamCapture_Reset1();
  void GlobalCapture_Reset1();
  void ReturnCapture_Reset1();
  void ParamCapture_Add2(const Wings::Slice &key, const std::string &value);
  void GlobalCapture_Add2();
  void ReturnCapture_Add2();
  void ParamCapture_Finish3();
  void GlobalCapture_Finish3();
  void ReturnCapture_Finish3(class Wings::Slice returnType);
  void ParamCapture_CurrentSizeEstimate4();
  void GlobalCapture_CurrentSizeEstimate4();
  void ReturnCapture_CurrentSizeEstimate4(size_t returnType);
};
```

具体的捕获函数不再详细说明，具体信息可以在 Wings 官网下载试用版本查看。

第六章 Wings 类型以及面向对象语法特性的支持

Wings 能够支持任意的类型以及面向对象的语法特性。

类型支持：

- 基本类型，int、double、float、std::string 等
- 任意复杂的结构类型，结构体、联合体、枚举、链表、多级指针、数组、树、图等
- 任意复杂的类对象，标准库容器、自定义模板类

特殊模板：

- 区分用户自定义的类型与系统变量类型（标准库头文件中包含的类型）
- 类的运算符重载函数
- void *与函数指针
- 识别类中包含 delete 与 default 关键字的函数进行特殊处理

语法支持：

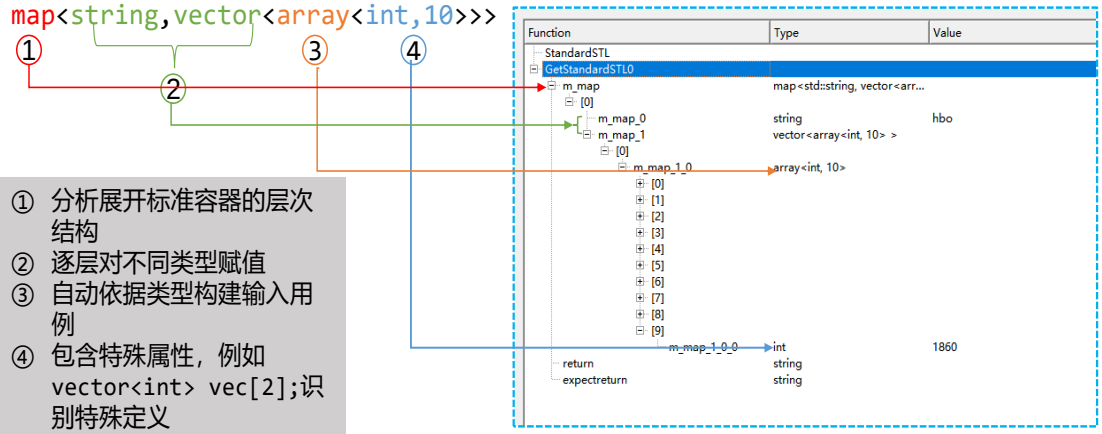
- 处理 static 函数、保护和私有函数
- 类中定义私有结构体
- 类中包含 delete 与 default 关键字的函数进行特殊处理
- 多态与复杂的类继承

6.1 链表

针对链表类型，采用比较灵活的赋值方式，考虑到实际应用中的一些因素，针对链表类型，默认赋值两层结构，在实际测试过程中，用户可依据需要自动添加节点。

6.2 标准库容器

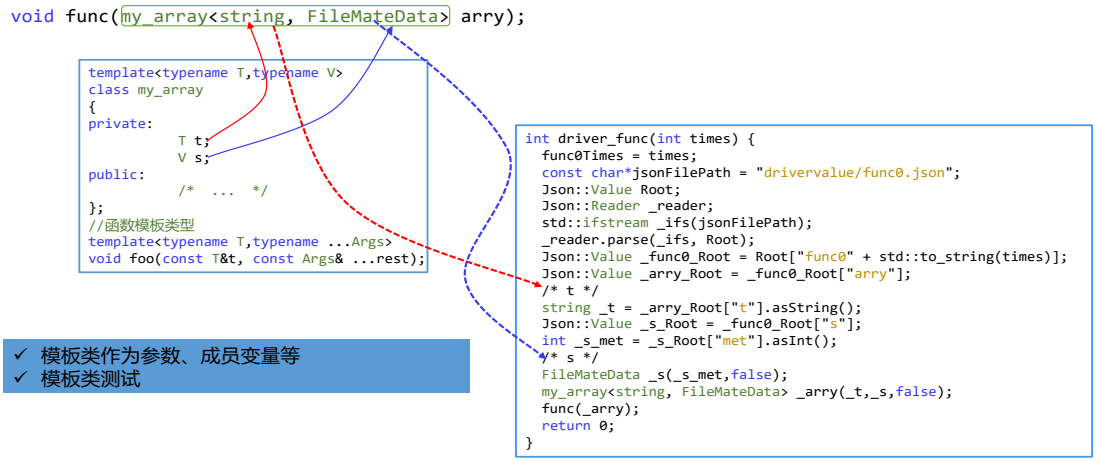
Wings 能够支持 c++的标准库容器，能够对容器进行逐层展开，利用不同容器的标准赋值函数进行赋值以取值。



其他类似的容器例如 QT 中的容器以及 boost 库中的相关容器，我们在继续支持。

6.3 自定义模板类

一些用户自定义的模板类类型，Wings 能够识别是用户自定义的模板类，Wings 依据实际程序中的赋值类型，进行处理。



6.4 void*与函数指针

Void*与函数指针在实际程序中可以作为函数参数或者结构体与类的成员变量，针对不确定的赋值类型，Wings 提供了具体的解决办法：

① 利用编译底层技术，对源程序静态分析，获取真实类型，对真实类型进行赋值

② 由用户在数据表格界面配置实际类型

6.5 特殊模板

在实际的代码程序中，会存在一些类型无法使用通用的模式全部展开,如一些系统变量（FILE、iostream）、第三方库以及一些用户需要特殊赋值。

Wings 是如何针对以上特殊类型进行赋值，举例如下：

```
struct sockaddr_in {
    short   sin_family;
    u_short sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```

步骤如下：

a. 识别 sockaddr_in 为系统变量类型，特殊标记

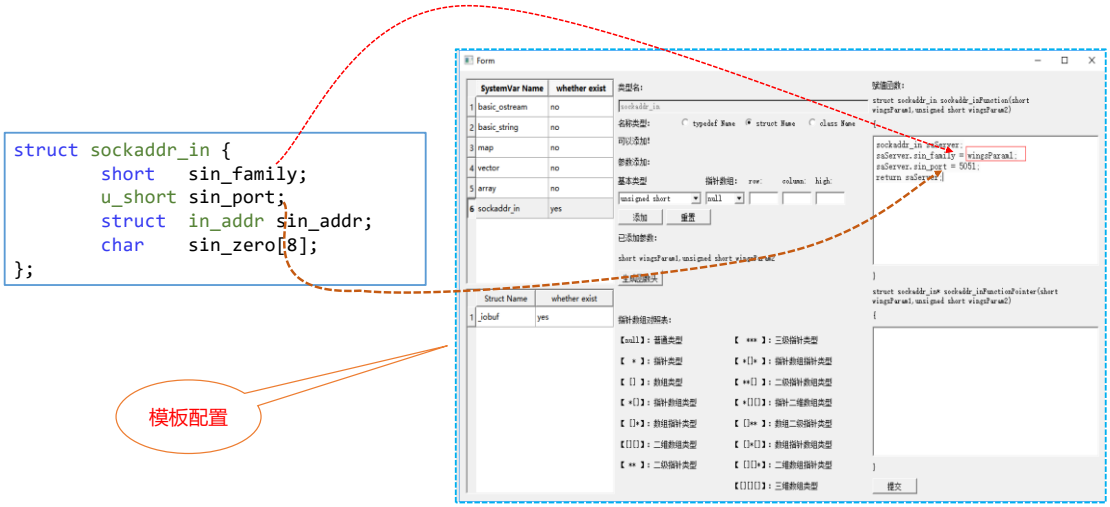
b. 检测程序中的所有系统变量类型，显示在模板界面

c. 用户配置特殊变量，如 sin_family

d. 构建 sockaddr_in 对象

e. 调用模板，生成驱动

模板配置如下：



图：6.5 模板配置

第七章 数据表格

Wings 目前测试用例数据采用随机生成的方式，支持 int、char、double、float、bool、char*类型。数据表格可以任意编辑以上类型的数值。

(1)Wings 数据表格将会针对参数进行展开，假如参数类型为结构类型，数据表格将分层展开结构的类型，到基本类型。

(2) 针对基本类型的指针类型，例如 int *p;Wings 处理为不定长度的一维数组类型，int **p;处理为不定长度的二维数组类型，默认长度为 3，数据表格界面可以点击进行添加和删除数据。

(3) 针对不定长度的数组作为函数参数，例如 int p[];Wings 默认长度为 1，用户依据需求，在数据表格界面进行任意添加和修改即可。

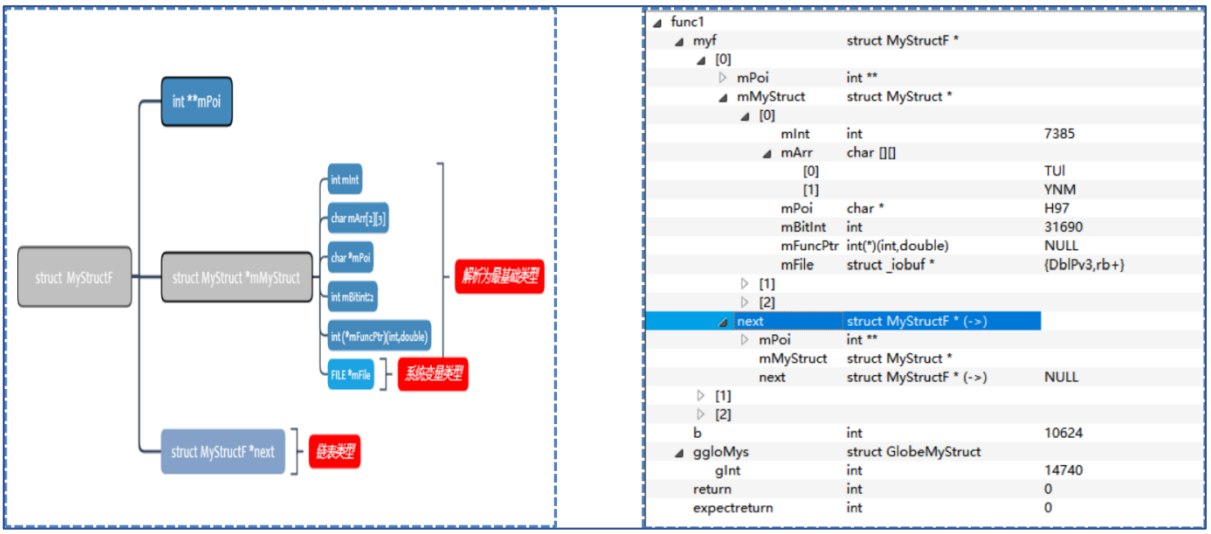


图 7-1 数据表格展示

附录 A

表一：type 属性

ZOA_CHAR_S/ZOA_UCHAR/ZOA_INT/ZOA_UINT/ZOA_LONG/ZOA_ULONG/ZOA_FLOAT/ZOA_UFLOAT/ZOA_SHOTR/ZOA_USHORT/ZOA_DOUBLE/ZOA_UDOUBLE	基本类型
StructureOrClassType	结构体类型
ZOA_FUNC	函数指针类型
ZOA_UNION	联合体类型
ZOA_ENUM	枚举类型
ClassType	类类型

表二：basetype 属性

BuiltinType	基本类型
ArrayType	数组类型
PointerType	指针类型
StructureOrClassType	结构体类型
UnionType	联合体类型
EnumType	枚举类型
FunctionPointType	函数指针类型

表三：其他属性

Name	代表结构体、类、联合体名字
NodeType	代表链表类型
parmType	代表函数参数类型
parNum	代表函数参数个数
SystemVar	代表此类型为系统头文件类型
value	代表枚举类型的值
bitfield	代表位域类型所占字节
returnType	代表返回值类型
Field	类成员变量
Method	类构造函数

paramName	类构造函数参数名
paramType	类构造函数参数类型
TemplateArgumentType	STL 结构参数类型
WingsTemplateArgument	STL 结构嵌套参数名字
TemplateArgumentValue	STL 结构中参数为具体值
FunctionModifiers	函数访问权限
FunctionAttribute	函数是 extern 或者 static 函数
FuncClassName	函数所属类
OperatorFundecl	重载运算符函数
Operator	重载运算符类型

星云测试系列产品

星云精准测试平台-JAVA 版

支持语言：Java 语言

支持平台：Android、J2EE 平台

企业级单元测试自动编码引擎

支持语言：C/C++语言

支持平台：Windows、Linux

星云精准测试平台-C/C++版

支持语言：c/c++语言

支持平台：Linux、Unix、嵌入式平台

支持语言：objective-c 语言

支持平台：iOS、OSX 环境

星雲測試
TestStars

大客户销售：18017932990

商务合作：13916614336

星云客服 QQ：1953892121

客服邮箱：contact@threadingtest.com

星云测试微信公众号

星云测试官方网站