# （一）立项依据与研究内容（4000-8000字）：

## **1．项目的立项依据**（研究意义、国内外研究现状及发展动态分析，需结合科学研究发展趋势来论述科学意义；或结合国民经济和社会发展中迫切需要解决的关键科技问题来论述其应用前景。附主要参考文献目录）；

### 1.1 研究意义

#### 1.1.1智能化时代三驾“马车”

我们正处在一个智能化时代（互联网+及工业4.0时代），大数据、云计算和人工智能已经成为这个时代进步的三驾马车(图1-1所示)，它们分别为智能化时代提供**数据**、**算力**和**算法**层面的支持，从而成为各行各业技术革新和社会发展的重要引擎。



**图1-1 拉动智能化时代的三架马车**

世界各国为推动智能化进程，特别是美、英、日以及欧盟等发达国家，制定了相应的发展战略以及行动计划（图1-2所示），2016年10月，美国前总统奥巴马在白宫前沿峰会上发布报告《国家人工智能研究和发展战略计划》，12月20日，美国白宫又跟进发布了一份关于人工智能的报告——《人工智能、自动化与经济》，日本政府也先后发布《机器人新战略》和《人工智能技术战略》，世界各国已经把发展智能化软件的政策提升到国家战略的高度。近年来，为促进大数据、云计算和人工智能的发展，我国同样密集出台了一系列发展战略、行动计划和支持政策：2015年，国务院颁布《促进大数据发展行动纲要》，强调数据已成为国家基础性战略资源。李克强总理在两会的政府报告中，提出要“制定互联网+行动计划 ”的要求，推动移动互联网、云计算、大数据、物联网与现代制造业结合，促进电子商务、工业互联网和互联网金融健康发展，引导互联网企业拓展国际市场；2016年，国家发改委、科技部、工信部、中央网信办联合发布了《“互联网＋”人工智能三年行动实施方案》，提出了三大方向共九大工程，系统地提出了我国在2016-2018年间推动人工智能发展的具体思路和内容；2017年，国务院颁布《新一代人工智能发展规划》，指出要抢抓人工智能发展的重大战略机遇，构筑我国人工智能发展的先发优势，加快建设创新型国家和世界科技强国；同年，工业化信息部发布了《促进新一代人工智能产业发展的三年行动计划（2018-2020年）》，目的在于深入实施“中国制造2025”，加快人工智能产业发展，推动人工智能和实体经济深度融合，力争于2020年在一系列人工智能标志性产业取得重要突破，在若干重点领域形成国际竞争优势。



**图1-2 世界各国人工智能政策与执行机构**

#### 1.1.2智能化软件系统的广泛应用

在当前全面推进战略性新兴产业及高技术制造业建设的形势下，智能化软件系统为提升现代企事业单位生产力水平提供重要支撑，为国民经济的飞速增长和社会的持续稳定发展提供有力保障。智能化软件系统是指能够产生人类智能行为的软件系统，通常通过学习或者自适应等方式获得处理问题的逻辑，具有强大的认知和问题解决能力，正在推动经济社会从数字化、网络化向智能化加速跃进。在智能化时代背景下，大数据提供的海量数据、云计算带来的超强计算能力以及人工智能算法的不断演进为智能化软件系统的飞速发展插上了腾飞的翅膀。

目前，智能化软件系统在各个领域表现良好甚至达到了人类的水平，谷歌公司的AlphaGo围棋智能机器人[5,6]依靠深度学习技术战胜了排名世界第一的围棋冠军柯洁及职业九段棋手李世石；IBM 公司的深蓝智能计算系统[6]战胜了国际象棋特级大师加里·卡斯帕罗夫。智能化软件系统渗透到我们生活的每个角落，正在逐渐改变我们生活的方方面面，主要包括以下领域：

1. **智能家居**：人工智能技术打造智能家电。通过人工智能技术丰富家用电器的功能，对家电进行智能化，并为各种音乐类智能辅助设备提供智能服务和类型的人工智能应用模式是目前最为智能家居市场所广泛接受的。如三星公司的智能冰箱Family Hub内嵌Tizen智能系统，整合了诸如音乐播放器、内置拍照监控、日历查看等功能，实现家电产品物联网化；美的冰箱携手阿里巴巴YunOS系统推出的“650升双屏新款概念互联网冰箱”使用了英特尔实感技术（Intel Real Sense）和英特尔 Haswell高性能处理器，通过图像识别技术记录食材种类和用户日常饮食数据，集合大数据云计算、深度学习技术，分析用户的饮食习惯，并通过对家庭饮食结构营养分析，结合时令、体质特征等多种维度，给出最全面最营养的健康膳食建议；亚马逊的Echo智能音箱，该音箱在语音识别上更加开放的策略，这款产品最大的亮点是将智能语音交互技术植入到传统音箱中，从而赋予了音箱人工智能的属性，这个被称为“Alexa”的语音助手可以像你的朋友一样与你交流，同时还能为你播放音乐、新闻、网购下单、Uber叫车、定外卖等等。

此外，人工智能技术助力家居智能控制平台，通过开发完整的智能家居控制系统或控制器，使得居住者能够智能控制室内的门、窗和各种家用电子设备，谷歌的Google Home家庭设备的控制中心、扎克伯格的“贾维斯”智能管家，苹果公司开发的HomeKit智能家居平台等都是智能家居控制平台的典型应用。

1. **智慧出行：**无人驾驶汽车是一种通过计算机系统来实现操纵行驶的智慧汽车，主要依靠智能路径规划技术、计算机视觉以及全球定位系统等技术协同合作，使计算机可以在没有人类操作下，安全驾驶汽车。无人驾驶汽车可以解决老年人残疾人出行困难的问题，也可以减少道路交通拥堵，同时可以一定程度上排除人为错误和不明智的判断，极大的降低道路交通事故安全隐患。近年来，无人驾驶汽车呈现出接近实用化的趋势，国外谷歌公司和国内百度公司在无人驾驶汽车领域处于领先水平。此外，智能交通机器人可以用于路口指挥交通，降低交通警察工作量；智能交通监控可应用于停车场、高速路口收费站、路口车辆抓拍等较为简单的监控设施；智能出行决策，可以通过对车流量、道路拥堵情况分析，提供智能路线规划、智能导航、出行信息提示和实时路况显示等服务，极大方便了人们的出行。
2. **智慧工作：**据埃森哲公司测算，到2035年，人工智能技术的应用将使制造业总增长值（GVA）增长近4万亿美元，年度增长率达到4.4%。作为新的“生产要素”，人工智能对于制造业的影响有几方面：（1）机器将部分取代人的工作，实现智能自动化。在中国、日本等国家，可以弥补由于老龄化、人力资源成本提升带来的劳动力短缺问题。（2）人工智能通过增强劳动力技能带来生产效率的提升，以提高人的效率，经过重新培训的员工可以执行更高级的设计、编程和维护任务或创造性的工作。（3）人工智能与制造业的深度融合不但将加速新产品的开发过程，还将彻底颠覆原有的生产流程，人工智能程序不仅可以自动完成任务，而且还可以实现全新的业务流程。比如，根据客户的个性化需求自定义产品配置。这将是人工智能在制造业领域的最终目标。目前，人工智能在制造及服务行业已经有了一些应用，京东的无人仓储依靠智能控制系统实现了6倍于人类的思考决策速度及 10倍于传统人工仓库的货物处理效率[7]；微软亚洲研究院利用智能化软件系统来优化现有的航运操作，改善航运业网络运营，提升经济效益[8]；微软亚洲研究院利用智能化软件系统来优化现有的航运操作，改善航运业网络运营，提升经济效益[8]。
3. **智慧医疗：**人工智能在医疗中的最早应用是诊断，医疗自动诊断促进了专家系统的发展，并且使专家系统在20世纪80年代后期达到顶峰。人工智能在医疗诊断上的应用能够和医生的专业知识得到相互的补充。计算机没有医生灵活多变的思维，但这也表明机器相比人类更能得到统一的结论。除了减轻医生的负担之外，人工智能的诊断结果可能比医生使用听诊器诊断的结果更准确。以诊断孕妇是否患有先兆子痫（Preeclampsia）为例，智能诊断可以比临床医生得到更早的结论。先兆子痫是导致孕妇死亡的主要原因，也是胎儿并发症的主要原因。 患有高血压的妇女在首次怀孕时几乎有15%最终患有先兆子痫。为了解决这个问题，孕妇必须提前进行分娩。[Velikova and Lucas (2014)](https://www.researchgate.net/publication/259133506_Exploiting_causal_functional_relationships_in_Bayesian_network_modelling_for_personalised_healthcare)提出一种统计模型能够比人类医生提前四周诊断孕妇是否患有先兆子痫，从而能够提前帮助患者进行抗高血压治疗。此外，智慧医疗还包括以下应用：**智能医疗机器人**可用于外科手术、功能康复及辅助护理等方面。谷歌母公司Alphabet和强生公司在2015年12月联合成立Verb Surgical公司，研发新一代辅助手术的机器人；**智能药物研发**已发展到虚拟药物筛选阶段，在计算机上模拟药物筛选的过程，对化合物可能的活性作出预测，对可能成为药物的化合物进行有效筛选，大幅降低药物开发成本，最著名的药物研发深度学习模型可能是IBM的Watson机器人，在慢性病管理、精准医疗、体外检测等九大医疗领域中实现了巨大突破；**智能诊疗**融合了知识图谱、自然语言处理、认知技术、自动推理、机器学习、信息检索等技术，通过假设任职和大规模的证据搜集、分析、评价，从而给出诊疗判断。百度在2016年成立了百度医疗大脑项目；**智能影像识别**指运用人工智能技术识别及分析医疗影像，帮助医生定位病症分析病情，辅助做出诊断；**智能健康管理**主要集中在风险识别、虚拟护士、精神健康、在线问诊、健康干预以及基于精准医学的健康管理。如Welltok的健康管理平台，运用人工智能技术分析用户体征数据，为每一个用户创建多渠道的个性化健康路线。
4. **智慧教育：**教育AI时代已经来临，不仅仅是对学生的用户行为进行分析，更重要在于，减轻老师工作量，解放家长的时间和精力。未来网络课程或将占据更大的比例，届时整个网络教育将会朝着个性化方向发展。这依赖于人工智能的大数据分析及智能语音，通过图像识别，可以进行机器批改试卷、识题答题等；通过[语音识别](http://www.wanwushuo.com/)可以纠正、改进发音；而人机交互可以进行在线答疑解惑等。人工智能加持下的在线教育可以实现对每个学生定制差异化教学方案，并跟进学习进程。除了主动灌输知识之外，随着AR/VR技术的发展，学生还可以与在线老师实现交互，手机、电脑两个端口互传，实现扫描翻译等工作，还可以实现兴趣指引等板块，比如唱歌，跳舞等都可以在线被指导。
5. **智能科研：**人工智能技术还可以提升研究人员发现和解决问题的能力，助力科学研究与发现，谷歌最新的人工智能AlphaFold，在一项极其困难的任务中击败了所有的对手，成功根据基因序列预测了生命基本分子-蛋白质的三维结构[9]；医学影像企业Enlitic开发了从X光照片及CT扫描图像中找出恶性肿瘤的图像识别软件，利用深度学习的方法对大量医疗图像数据进行机器学习，自动总结出病症的“特征”以及“模式”[10]。
6. **智能金融：**随着人工智能的兴起，越来越多的经济学家对人工智能在经济上的应用产生了浓厚的兴趣，这种趋势越来越明显，并且在许多现实世界的任务中，例如交易任务，人工智能积极地超越并取代金融工作者。AI对人类的主要优势是能够处理大量的互联网上产生的数据。例如，想要预测未来的股票价格，AI可以使用公司活动信息，评论，新闻，微博上涉及的信息以及许多其他来源的信息。但是对于金融工作者而言，几乎不可能为交易者实时处理并分析这些数据中所包含的信息。研究人员已经研究了很多方法将人工智能应用到金融方面，比如说递归神经网络在预测价格方面显示出令人满意的结果。相应地，其他深度学习方法在破产和欺诈预测，信誉度量和风险管理方面也取得了较好的结果。以欺诈预测为例，如果用户的网上虚拟账户被盗，恶意攻击者可能转移账户中的资金到其他陌生账户中，因为AI可以通过模式识别技术理解客户的历史行为以及判断此次交易是否是异常行为，进而预防攻击者的恶意行为。

**（8）智能零售：**技术进步正在改变公司与客户沟通和互动的方式。人工智能在21世纪的零售业中扮演着重要的角色以及是零售业的发展趋势之一。它被认为是零售数字化的新杠杆，也是向客户提供个性化体验的创新方式，同时使零售商能够提高生产力。研究人员Gartner表示，到2020年，零售业中85%的客户互动将由人工智能管理。凭借丰富的客户数据，零售商可以推出图像识别系统，定制产品推荐，购物助手，实时价格调整以及面向聊天机器人的客户服务。 所有这些致力于销售，在线或实体店的创新都基于人工智能技术。例如面部识别系统已经被零售店用来作为促进销售的有力工具，他们通过屏幕嵌入式摄像头，识别客户所穿的服装的性别和类型，然后显示个性化的推荐，最终提供令客户满意的产品；零售业中的库存管理一直是零售商的一个问题，为了确保零售商有足够的商品来应对高峰需求而又不产生库存积压，人工智能技术已经被用于仓库和商店之间的拣货、库存组织和订单分配等工作上。比如，微软的合作商推出Fellow Robots服务机器人，它通过人工智能识别产品并验证产品是否带有正确的标签，以及产品是否处于正确的位置或者是否被错误的移动等来提高工人拣货的效率。除此之外，Focal systems公司推出配备有侧摄像头的智能购物手推车，智能购物车能够在顾客购物时不断拍摄货架，然后分析图像并且用计算机视觉算法检测货架上物品的数量，从而准确地知道库存是否处于缺货的状态。如果货架上商品的数量低于设定的阈值，则会向员工发送警报以进行补货。

**（9）智能电商**：人工智能技术已被成功的用于电商平台的商品管理，创建个人推荐，预测商品价格，提高产品图像质量以及识别可疑广告和行为。机器学习技术的使用改善了用户体验，使电商平台的使用更加安全。关于商品管理主要是处理不符合法律要求的卖方广告。例如，涉及违禁药品、危险武器、色情信息的广告。智能系统通过检测广告中的文本或图像，将疑似包含上述信息的广告进行拦截或将其发送给员工进行人工复查。与直接让员工人工检查相比，人工智能系统可以降低时间开销，降低成本并提高质量。在价格预测方面，对于商品的市场价格有准确的预测是十分重要的。特别对于p2c市场上的卖家来说，设置最优价格以快速销售商品并同时获得最大利润非常重要。为了确定价格，卖家需要做手动市场研究，这需要花费很多时间。目前电商平台可以借助人工智能系统进行市场分析，并向卖家推荐商品的最优价格，还能够向卖家显示根据不同价格预测的买家兴趣水平。对于买家而言，系统能够提示卖家产品的价格是否最佳。人工智能在价格预测上的应用使卖方和买方的体验更加愉快和有效。关于商品个性化推荐，系统会分析用户行为以及用户历史点击的商品或者广告，利用机器学习算法构建预测模型，并及时向用户展示他可能感兴趣的商品列表。实施个性化建议还可以增加电商系统的市场收入，并提高买家和卖家的满意度。人工智能技术还可以提高产品的图像质量。产品的图像质量会影响市场的销售水平。 通常，大型制造商和经验丰富的销售商具有能够创建高质量图像内容的优势。 然而，较小的卖家通常局限于移动电话上的相机质量。 为了提高用户上传的照片质量，一些智能图像处理技术已经被用到电商平台，例如Ubcoin Market， 使用超分辨率技术将普通用户上传的商品照片自动发送到人工智能模块，经神经网络处理以提高图像质量。

#### 1.1.3智能化系统的安全隐患

智能化软件系统为人类带来了极大的便利了，然而智能化软件系统也并非是完美的，在性能和安全可靠性上也会遇到种种挑战，**智能化软件系统一旦发生故障，则损失巨大，甚至危及生命安全**[118]**。**

1. 流行的亚马逊Echo被认为是“聪明”的智能音箱之一，但一个德国人的Echo音箱在他不在家的时候被意外地激活了，在午夜之后开始播放音乐，吵醒了邻居，他们给警察打了电话，警察不得不从前门破门而去，把音箱关掉；
2. 亚马逊Alexa助手自动帮小孩下单购买玩具，一名六岁的女孩在跟Echo内置人工智能语音助手Alexa聊天时，意外订下了价值170美元的玩具和一盒重达四磅的饼干。虽然孩子的妈妈在收到一个不知打哪来的订单确认电话后立刻进行了取消操作，但该订单却已经被处理，且一个跟孩子身高几乎相同的玩偶就在隔天送来了。最后，他们无奈的决定将这一玩偶捐赠给当地儿童医院；
3. 国外网站出现一段视频，在视频中小孩向亚马逊Alexa下达指令：“Alexa，请播放‘Digger digger（一首儿童歌曲）’。”之后，亚马逊个人助手通过算法进行识别，竟然认为孩子想听情色内容。不一会，孩子的父母才意识到发生了什么事，可惜他们已经无法阻止Alexa继续播放声音。对此，亚马逊新闻发言人表示：“我们之前已经修正过该问题，但Alexa很难完全屏蔽成人内容。目前我们正在加强限制防止类似的事情再次发生，并已经与客户取得联系向他们致歉。”
4. 微软旗下的聊天机器人Tay变成种族歧视和屠杀支持者[13]。
5. 2016年，谷歌的无人驾驶汽车和一辆公共汽车相撞，原因是系统错误的预测公共汽车会在一系列罕见的条件下减速或停车让行，然而实际上公共汽车不可能停止[7]；
6. 2016年，特斯拉的一辆无人驾驶汽车和一辆拖车相撞，原因是拖车的外表颜色和天空相近并且底盘较高，导致系统将一辆白色卡车误检测为天空[8]。
7. 在2016年的深圳高交会上，一台名为“小胖”的机器人在没有指令的情况下突然自行打砸展台玻璃，最终导致部分展台破坏。与此同时，该机器人撞向展台玻璃致使玻璃倒地、摔碎，并划伤一名现场观众，致其脚踝被划破流血送医；
8. 2015年“达芬奇”医疗机器人在**心瓣修复手术中**把病人的心脏“放错位置”，并戳穿其大动脉，病人最终由于多器官衰竭逝世。在无法真实模拟病人的情况下，难以全面测试系统所有可能的情况。
9. 2017年日本一架无人机因操作失误在坠落的过程中砸伤一名工人，因为无人机飞行时受到无线电信号干扰，致使飞行器跟遥控器失联，在飞至70米高空时与一起重机相撞坠落进而砸向一名工人，致使其面部受伤。

......

类似的案例还有很多，这些案例很多都是智能化软件自身漏洞导致的，这些漏洞或许导致软件的失控，如Echo音箱莫名启动，但也有可能在一些安全关键的领域（如无人驾驶、航空航天等）带来灾难性影响，软件质量缺陷造成的损失往往是难以承受的。因此，在智能化软件系统充溢生活的今天，保障智能化软件的质量是一个意义深远的问题。**如何有效保障智能化软件系统正确、高效、可靠地实现其既定任务是一个需要解决的重要问题。**

随着软件规模的不断增大，以及软件复杂度的不断提高，软件系统的安全可靠性也面临严峻的考验，软件漏洞的存在往往是不可避免的。因此，软件安全漏洞检测应成为智能化软件测试的一个重要组成部分。软件安全漏洞是在软件设计与实现过程中存在的一些容易被恶意攻击者所利用、对软件安全构成威胁的缺陷或不足。常见的软件安全漏洞包括缓冲区溢出、内存泄漏、整数溢出等。一旦攻击者构造恶意输入，可能造成任意命令执行、任意文件读取等严重安全问题。因此，如何快速而准确地发现软件安全漏洞一直以来是软件安全领域的研究热点，也是智能化软件测试关注的一个难点。

代码和缺陷报告作为软件开发和维护的重要组成部分，能够为软件安全漏洞检测提供重要的信息来源。通过软件测试的方法发现出的软件缺陷通常是以缺陷报告的形式记录的，尽管缺陷报告在被提交时需要被标明该缺陷是否是安全漏洞，但是由于对信息安全知识的欠缺，报告提交者往往无法正确的区分安全漏洞与常规的软件缺陷，因此许多安全漏洞常常被错误地标记为与安全无关的缺陷，从而导致这些安全漏洞无法被及时地修复。若Security相关的漏洞无法被及时地识别和修复，则容易被恶意攻击者所利用，使得系统被入侵进而遭到破坏，例如人脸识别账号被盗刷、智能家居被入侵等，2017年央视315晚会上技术人员仅凭一张观众的自拍照，借助人脸关键点定位和自动化人脸动效技术，通过将自拍照由静态改为动态，可以完成刷脸登录需要的眨眼、微笑动作，成功破解了刷脸登录的人脸认证系统，说明像这种使用人脸识别的APP等智能软件也都存在被盗用的风险，使得个人的安全隐私受到威胁。若Safety相关的漏洞无法被及时地识别和修复，则有可能引发严重的安全事故。例如，2015年“达芬奇”医疗机器人在心瓣修复手术中把病人的心脏“放错位置”，并戳穿其大动脉，病人最终由于多器官衰竭逝世。可见，程序代码一旦存在安全缺陷，就会造成比普通缺陷更为严重的影响。因此需要尽早识别、定位和修复安全相关的漏洞，使自动驾驶、医学成像等智能软件系统的部署和运行更为安全可靠。因此，**软件安全漏洞检测对于保障智能软件这类安全关键（safety-critical）系统的安全可靠具有十分重要的意义。**

若要保障智能软件系统的安全，不仅要准确地识别出安全相关的软件缺陷，还要快速地定位缺陷所在的源文件，以便及时地修复缺陷，避免安全缺陷长时间暴露于公众之下给恶意攻击者以可乘之机，这就需要开发者能够根据缺陷报告中提供的相关信息，有效地定位到相应的包含此缺陷的源文件。 为了定位缺陷，开发人员不仅需要使用他们的领域知识分析缺陷报告，还需要从同行开发人员和用户收集信息。尤其当文件和报告的数量很大时，比如说有数百甚至数千个文件组成的大型项目，使用人工代码审查的方法定位缺陷所在的源码和分析缺陷产生的原因可能非常耗时。因此，如果能通过解析缺陷报告自动定位到缺陷源代码，则有助于提高软件维护的效率。

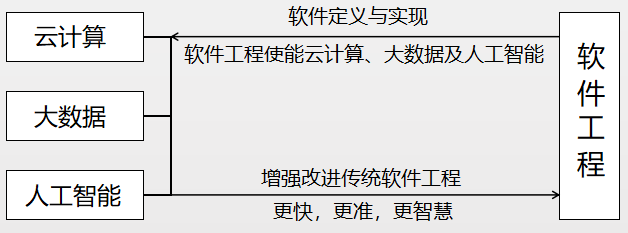
然而，大部分源文件可能是与缺陷无关的，开发者要想从数以千计的源文件中找到包含此缺陷的源文件如同大海捞针，而缺陷报告文本描述的多样性和缺陷信息的不完整性，以及缺陷报告中用自然语言描述的缺陷信息与软件代码之间的语义失配和词汇鸿沟问题，进一步加大了从报告自动定位到代码源文件的难度。因此，**如何从大量的软件缺陷报告中自动地识别和定位可能危及智能化软件系统安全的缺陷是一个急需解决的关键科学问题。**

### 1.2 国内外研究现状及发展动态分析

#### 1.2.1软件工程发展动态

软件工程是一门研究用工程化方法构建和维护有效的、实用的和高质量的软件的学科。它涉及[程序设计语言](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80/2317999)、[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728)、[软件开发工具](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%BC%80%E5%8F%91%E5%B7%A5%E5%85%B7/4605523)、系统平台、标准、[设计模式](https://baike.baidu.com/item/%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F/1212549)等方面。现代社会中，软件应用于多个方面。典型的软件有[电子邮件](https://baike.baidu.com/item/%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6/111106)、[嵌入式系统](https://baike.baidu.com/item/%E5%B5%8C%E5%85%A5%E5%BC%8F%E7%B3%BB%E7%BB%9F/186978)、人机界面、[办公](https://baike.baidu.com/item/%E5%8A%9E%E5%85%AC)套件、[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F/192)、[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8/8853067)、[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728)、[游戏](https://baike.baidu.com/item/%E6%B8%B8%E6%88%8F/33581)等。同时，各个行业几乎都有计算机软件的应用，如工业、农业、银行、航空、政府部门等。这些应用促进了经济和社会的发展，也提高了工作效率和生活效率 。

信息化时代下，由软件定义与实现的云计算、大数据和人工智能正蓬勃发展，悄然的改变着这个世界，极大的解放了生产力，进一步提升了工作和生活效率，同时云计算、大数据和人工智能的发展也深刻地改变着软件工程领域（图1-3所示），以下我们简单回顾一下这些改变，展望未来的软件工程发展趋势。



**图1-3 软件工程与“三驾马车”**

**云计算与软件工程**

社会需求推动了云计算的日益发展，云计算平台以及云计算环境下的软件可靠性和安全性越来越成为人们关注的焦点，并已经成为云计算发展和应用的瓶颈，这给传统的软件工程提出了一系列挑战，例如：（1）云计算环境的特殊性研究，研究云计算环境下资源的服务化、易接入性、可扩展性和可度量性等，研究这些特性给云计算环境下软件可靠性、安全性及软件测试带来的特殊性问题。（2）云计算环境下的软件可靠性理论与方法研究，研究开放的运行环境下软件运行剖面的变化规律及建模方法，云计算环境下软件可靠性建模理论等。（3）云计算环境下的软件安全性理论与方法研究，研究云计算环境下私密性、完整性、可追踪性和可用性等。

另一方面，云计算平台提供的几乎无限强大和灵活的计算能力，极大地推动了软件工程的理论和实践：例如，继基于组件的软件工程之后，面向服务的软件工程，人们提出在云平台上软件即服务（SAAS），一切皆服务（XAAS）的观念。并开始研究如何利用云平台基础设施所提供的资源来进行在线需求获取、软件构造、软件测试等的一般方法，并且研究如何将传统的过程、方法以及他们的基础设施向云端迁移。开源社区、众包空间和软件生态系统异常活跃。

**大数据与软件工程**

大数据是由软件定义实现的，同时大数据分析也可以有效增强软件工程实践，两者在现实世界中互相丰富和发展。

一方面，软件工程助力大数据软件系统开发。人们已经做了很多工作来研究如何通过软件工程来提高和改进大数据系统的能力，因为大数据系统给人们提出了很多挑战，例如，任何时候，任何需要，大数据系统需要能支持快速的、弹性的和跨多数据中心的扩展。因此，需要构建一个可灵活扩展，可自由合成的和无需人工干预的大数据系统是关键。主要体现在：

（1）大数据软件系统的系统架构。人们研究了大数据系统模块化和可扩展等需求，以及数据资源、数据存储和数据集成等数据流功能模块，提出了很多适合大数据系统的软件体系结构。

（2）大数据系统的测试和调试。研究大数据系统的测试标准，利用形式化方法对大数据系统的正确性进行验证。大数据系统运行过程中产生的日志等数据也具有大数据特性，这些数据也是我们对大数据系统进行测试和调试的重要依据。

（3）利用适当的软件过程开发大数据软件系统。人们已经开始研究利用CMMI和敏捷软件工程方法对大数据系统进行开发，提高开发效率和质量。

（4）大数据项目的管理。研究大数据项目开发的支持工具和技术，为大数据项目的需求、设计和实现等过程提供全程管理。

大数据软件质量保证需要采纳和扩展传统软件工程的质量保证技术，例如，测试大数据软件需要一些能够生成充分且有代表性测试用例的新方法，然而，由于数据规模巨大，穷尽测试不可能，因此需要采用形式化验证技术对大数据系统进行分析。数据源来自多个方面，并不是每种数据源都与大数据分析有关，且采集处理这些数据都需要相应的成本，大数据软件的成本效益需要在事前和事后进行评估。

大数据分析的需求给软件工程提出了很多挑战，例如为了处理大数据的大容量性、多样性、高速性、准确性等问题，不仅需要一些新的分析算法和工具，同时也需要持续发布及质量保证等非功能改进。

另一方面，大数据助力软件工程。人们一直研究如何利用大数据方法来改进和提升软件工程能力，具体包括以下几个方面：

（1）软件工程分析及其可视化。研究软件工程过程中的数据收集，整理和应用等问题。具体解决的问题包括：基于大数据的软件需求分析、结构、设计、编码、测试和维护等。

（2）软件社区数据挖掘。通过对一些大型组织或开源社区长期积累的数据进行挖掘，发掘一些对软件工程方法和技术有价值的认识。具体解决的问题如：源代码中哪些代码是新写的，哪些代码来自其他项目？从成百上千的项目观察中，我们能否跟开发者推荐最佳实践？

（3）自适应系统。通过对软件工程数据的应用，使得软件工程具有自适应性。具体解决的问题包括基于大数据的故障诊断、隐私保护、安全性和可靠性等问题。

软件生命周期中会产生大量的、各种类型的数据，例如开发过程中的源代码、需求文档、缺陷报告、测试用例；系统运行中的运行日志、性能度量、事件记录；用户交互中使用行为序列、调查问卷、社交媒体反馈等等。随着互联网软件服务的普及，数据的种类更加丰富，规模也越来越大。基于这些数据，软件从业者可以提取出关于软件质量和开发动态的重要信息。因此，数据对于现代软件开发的作用日益明显，并且至关重要。

数据驱动的软件工程运用机器学习、数据挖掘、信息可视化以及大规模数据处理等技术，旨在帮助软件从业者以数据驱动的方式进行软件的开发、运行和维护，有效处理、浏览和分析软件生命周期中生成的数据，从中提取有用的信息，做出正确决策。数据驱动的软件工程关注大数据分析技术在软件行业中的具体应用，是对现代软件工程方法的有效拓展，同时也与软件开发的实践密切相关。

**人工智能与软件工程**

“人工智能”这个词汇最早是出现在1956年，是在Dartmouth学会上美国的一个专家首先提出来的。从此以后，各个国家的相关研究者从中发展出了很多的人工智能理论和原理，人工智能的概念也逐渐的被人们所了解和认知了。简单来说，人工智能技术主要研究的目标是要能够使一台计算机或者一台机器可以完成一些通常需要我们人类亲自动手或动脑来完成的工作，也就是说只有依靠人类本身的智能才能完成的一些复杂工作。所以计算机本身就是人工智能的体现，人工智能技术的发展历史和计算机科学与技术的发展是密不可分的，人工智能技术也必须依靠计算机技术的发展才能逐渐发展成熟。

欧洲信息技术研究计划（ES-PRIT）之前就曾提出要把人工智能技术和软件工程技术紧密的结合在一起，从而可以开发出一套比较有效的工具，这个工具同时也可以支持软件系统的具体分析和设计工作。而且近年来越来越多的研究也表明，人工智能技术和软件工程技术的联合发展是非常有必要的，同时也必将引起软件开发方法和软件程序管理模式的改变，这样就可以形成一个新的开发和管理规范，而人工智能技术也可以使软件的开发更加容易、更便于修改和维护。

从目前发展情况来看，人工智能可以使得软件工程研究更加智能化与高效化，具体包括以下几个方面：

（1）可以利用人工智能中专家决策系统以及人工神经网络系统来对软件工程项目进行更加智能化的设计。

（2）把智能化的模块组装到某个大型的软件系统中，从而使得这个软件可以更好的适应快速改变的需求。

（3）应用人工智能技术来提高某个计算机程序的用户界面的友好性，以及把人工智能技术应用到对图形用户的接口以及面向对象的程序设计当中等。

（4）通过利用人工智能方法对代码、各种评审记录和日志等资料进行分析，可以有效预防和发现各种软件故障，特别是利用强化学习方法生成高质量的测试用例，使软件工程的能力得到大幅度的提升。

虽然取得了很多的进展，但是计算机当中应用人工智能技术在未来很长的一段发展时间中还是有很多问题需要解决的，比如说，如何基于人工智能系统来建立支持环境和软件开发的人工智能机制。

另一方面，人工智能，特别是机器学习与深度学习的发展，要求强大的系统软件支撑，软件工程可以为人工智能研究提供架构、测试的支持以及指导，具体包括：

（1）软件工程可以为人工智能算法提供很好的实践验证平台。人工智能算法的有效性与可靠性，不能只停留在理论推导，需要进行一定的实证，而软件工程正是这样一个连接人工智能理论与实际应用相结合的桥梁，在实践中检验真理。

（2）软件工程可以提供面向人工智能的产品架构，指导人工智能产品开发、调试、测试、维护、管理等。例如，人工智能算法需要大量处理数据，这就需要数据库技术的发展，人工智能算法的可靠性验证则需要测试技术的革新。

（3）软件工程作为一门成熟学科，可以对人工智能的发展提供一定指导。人工智能作为新兴学科，其在正确性、可靠性、应用性上还有很多困难需要解决，而软件工程经过数十年发展，已发展成一门涉及[程序设计语言](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80/2317999)、[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728)、[软件开发工具](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E5%BC%80%E5%8F%91%E5%B7%A5%E5%85%B7/4605523)、系统平台、标准、[设计模式](https://baike.baidu.com/item/%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F/1212549)等方面的成熟学科，很多研究技术和经验对人工智能发展有很大的指导作用。

智能化已成为软件工程发展的新趋势，很多软件工程的难题可以尝试用人工智能方法去解决，智能化软件工程主要特征包括：

（1）需求分析自动化。针对软件项目中需求模糊，模型抽象，功能杂乱等难题，利用人工智能分类算法，Topic Model和自然语言处理相关理论进行处理。

（2）代码分析自动化。可以利用人工智能中卷积神经网络等方法实现自动化检测代码缺陷，甚至可以自动写代码。

（3）软件测试自动化。利用深度学习方法可以智能化选择合理的测试技术与方法对软件进行全方位、多角度的测试。

（4）故障诊断自动化。针对日志数据，挖掘其中有效信息，利用深度学习方法和聚类技术，可以自动的诊断定位故障。

（5）故障修复自动化。对于诊断定位的故障，利用历史修复策略等信息自动修复故障，实现故障处理的智能化。

**云计算环境下的基于人工智能和大数据的软件工程**

互联网和云计算的发展，软件已无处不在，计算机软件正在重新定义和实现着整个世界，同时，智能化时代的到来，软件系统的智能化日益成为人们关注的焦点，所以，不久的将来，人工智能系统将替代各种软件进入人们生活的每个角落。人工智能和软件工程将空前的紧密融合在一起，一方面，软件工程通过人工智能的应用变得更加强有力，另一方面，软件工程为无处不在的各种人工智能系统开发和应用保驾护航，推动智能化时代的健康有序发展。

#### 1.2.2软件测试方法体系的研究现状

从1969年开始,人们就已经提出软件工程的概念,研究如何以最经济的方式，在最短的时间内开发用户最满意的软件产品。整个软件工程可以说是一个不断与软件错误和缺陷做斗争的过程，为了生产高质量的软件，软件工程为软件质量把好下图所示的“四道关”（图1-4所示）。第一道关是利用形式化方法[34-38]、高度集成的软件开发环境和各种支持工具等在软件需求分析、设计和编码阶段严防死守，预防可能出现的各种问题和错误，这一步虽然非常重要和有效，但一般不可能消灭所有错误，仍有很多错误防不胜防而隐藏在软件中[17]。第二关就是要通过软件测试方法来发现和纠正这些没有防得住而隐藏在软件中的错误。即使软件测试技术已非常成熟，我们也很难保证通过第二关软件测试之后，软件中就不可能再有错误，所以，我们需要第三关，通过容错计算技术，在软件系统中植入容错能力，使得即使在前两关中没有防得住的错误、没有检查出来的错误存在于软件系统，也不至于给系统造成重要损失。第四关是在前面三道关的基础上，对软件系统仍然存在错误风险的预测，即软件可靠性的研究。

第一关

预防错误

（形式化方法）

第二关

检测错误

（**软件测试**）

第三关

容忍错误

（容错计算）

第四关

预测错误

（软件可靠性）

**图 1-4 软件质量“四道关”**

软件测试是指以发现错误、度量和提高软件质量为目的而理解、分析和运行程序的过程。软件测试的目标是以最小的成本，及时准确地发现软件中隐藏的错误，从而提高软件质量，降低风险。人们根据软件开发的不同阶段、不同的测试对象、基于不同的理论和角度提出了一系列软件测试方法，形成了图1-5所示的软件测试方法体系，具体可以有如下几类：

（1）软件测试根据是否运行程序可分为静态测试[2]和动态测试[4]。静态测试包括桌面检查、代码审查和代码走查等方法。动态测试根据测试用例设计是否依据程序内部结构可以分为黑盒测试和白盒测试[2]，白盒测试包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖、线性代码序列及跳转测试等；黑盒测试包括等价类划分、边界值分析、因果图分析、错误猜测、状态转换测试等[3，40]。

（2）根据软件开发的不同阶段可以将软件测试划分为：单元测试[10]、集成测试、系统测试、验收测试、回归测试、验证测试、确认测试、Alpha测试、Beta测试和Gamma测试等[4]。

（3）根据被测试软件的开发方法和应用环境的不同可以分为[1,4,20]：面向对象软件测试[6]、面向方面软件测试、面向服务软件测试[15]、构件软件测试[5]、嵌入式软件测试[42]、Web 应用和网构软件测试[41]等，后面还要出现普适计算环境下的软件测试、云计算[7]和物联网环境下的软件测试等。

（4）根据软件不同特性和方面的测试可以分为[1,4]：负载测试、压力测试、性能测试[14]、安全性测试、安装测试、可用性测试、稳定性测试、授权测试、用户接受性测试、一致性测试、配置测试、文档测试、兼容性测试和Playtest（试玩测试）[37]等。

（5）根据不同特殊的测试技术可以有：组合测试[8,43]、蜕变测试[19]、变异测试[22]、演化测试[9，11]、FUZZ测试[29]、基于性质的测试[18]、基于故障的测试[1]、基于模型的测试[17]、反模型测试[28]、结对测试[31]、在线测试[32]、基于操作剖面的测试[20]、基于用例和／或用户陈述开发测试用例、基于规格说明的测试[23]、统计测试[21]、逻辑测试[39]、随机测试[16，25]、反随机测试[26，27]、自适应随机测试[12，24]、GUI测试[30]、冒烟测试和探索测试[33]等。



**图 1-5 软件测试方法体系**

#### 1.2.3软件测试两大关键问题

**软件测试是一种广泛采用的软件质量保证手段**[14]，通过运行有限的测试用例，比较测试用例的输出与预期输出是否一致来检测软件中潜藏的故障。软件测试包含两项关键的任务，即**测试用例的生成**和**测试结果的判定**。测试用例生成为待测软件产生用于检查程缺陷的软件输入及其对应的预期输出，直接影响着软件测试的有效性和效率。代表性的软件测试方面的研究工作包括[15]：

* **基于符号执行的测试用例生成：**使用符号值代替具体值分析程序所有可能的执行路径，并通过约束求解为执行路径生成测试用例[16]。符号执行技术由King等人[17]首次提出，其初衷是使用符号值执行程序并以此分析程序所有可能的行为。然而，该技术受制于困难的约束求解过程和庞大的程序分析开销。为解决约束求解难的问题，研究者开发了多种约束求解器，如z3[18]、STP[19]、choco[20]等；为解决程序分析开销大的问题，研究者提出了选择符号执行[21]、动态符号执行[22]等。此外，研究者通过符号执行生成高覆盖率的测试用例：Cadar等人[23]设计并实现了符号执行工具KLEE，可以为待测程序生成超过90%代码覆盖率的测试用例，Godefroid等人[25]在符号执行的约束表达式中引入了符号变量在执行中应满足条件的相关描述，提升了白盒测试中用例生成的有效性与效率。
* **基于模型的测试用例生成：**使用待测程序的形式化模型（有限状态机、UML模型等）自动为待测程序生成测试用例[25]。基于有限状态机的测试用例生成方法将待测程序所有可能的状态进行抽象，定义状态转换所涉及的输入及输出，构造待测程序的有限状态机模型，最后以状态覆盖、转移覆盖、路径覆盖等策略从状机模型中选择状态转移序列并生成测试用例[15][26]。例如， Kansomkeat和Rivepiboon [27]使用UML状态图为待测程序构建模型并自动生成满足一定覆盖条件的测试用例集。2018年Xue Han等人[110]分析三个大型开源项目中的300个错误报告,他们发现输入参数和配置的组合可以暴露性能错误，首次提出PerfLearner，根据错误报告自动生成测试帧，这是一种提取测试框架元素的自动化方法并为性能测试生成测试框架以指导实际性能测试用例生成。实验证明其能生成测试框架和检测实际性能错误。
* **基于组合测试的测试用例生成：**使用不同的组合策略将待测程序不同参数的值进行组合，为待测程序生成规模尽可能小而不显著损失故障检测能力的测试用例集（组合覆盖表）。目的是使用较少的测试用例分析软件中各参数之间的相互作用对整个系统产生的影响。目前，组合测试中主要有四类测试用例生成方法，包括贪心算法、启发式搜索算法、代数方法、随机方法。贪心算法以尽可能多地覆盖未覆盖组合为原则给待测程序逐一生成测试用例并形成测试集，典型的方法有one-row-at-a-time [28]及in parameter order 算法[29,30]。启发式规则对预先存在的测试集进行一系列转换，直至测试集覆盖所有的组合，典型的方法有Ghazi等人[31]提出的基于遗传算法的技术及Bryce和Colbourn[32]在贪心算法基础上改进的技术。代数方法通过数学函数或递归构造的方式推导测试用例，典型的方法有[33,34]。随机方法则以随机的方式从测试全集中选择测试用例，通常被作为实验对比的基准方法[35-37]。
* **适应性随机测试：**使新生成的测试用例与所有已生成测试用例具有最远的距离，使测试用例在软件输入域上更加均匀地分布。Chen等人[38]在此想法的基础上提出了适应性随机测试，改进了随机测试的故障检测能力。此后，研究者在最远距离测试用例选取方面提出了多种不同的策略，形成了适应性随机测试的多个变种。Chen等人[38]从已有的候选随机测试用例中选取与已生成测试用例差别最大的测试用例，并提出了基于固定规模候选集的适应性随机测试（FSCS-ART）。Chen等人[39]提出了镜像适应性随机测试（MART），通过在输入域中选择与已有测试用例“镜像”的新测试用例，实现最远距离测试用例的选取，，减少了适应性随机测试的计算开销。Chen等人[40]提出了基于随机分区的适应性随机测试，使用已执行测试用例对程序的输入域进行划分，并从最大的分区中选取测试用例。。
* **基于搜索的测试用例生成**：使用搜索算法自动化从待测程序的输入空间中寻找能够最大程度满足测试目标且最大限度降低测试成本的测试用例集。在程序结构测试方面，研究者针对多种程序覆盖准则提出了不同的测试用例生成技术[51]，如分支覆盖[52-53]、判定覆盖[54]、数据流覆盖[55-56]等。在基于模型的测试方面，程序模型的状态迁移覆盖一直是研究的热点，代表性的工作有：Li等人[57-58]为UML状态图表达的程序模型生成满足状态迁移覆盖的测试用例集，Lefticaru和Ipate[59]使用遗传算法自动为程序的状态图生成测试用例集。在变异测试方面，研究者通过搜索的手段生成能够杀死待测程序变异体的测试用例[60-63].
* **动态随机测试技术：**Cai等人将控制论引入到软件测试中，提出了利用测试过程的历史信息更新测试剖面的适应性测试技术[101]，但是适应性测试技术的时间开销太高。为了降低适应性测试技术的时间开销，Cai等人根据引起故障的输入趋向于集簇在连续的区域这一观察[102,103]，提出了利用当前的测试信息更新测试剖面的动态随机测试技术[104]。该技术的主要特点是在测试的过程中根据每一次的执行结果动态改变测试剖面，使得具有较高失效率的分区的选取概率变大。动态随机测试的性能受两方面的影响：初始测试剖面和动态随机测试中的参数值。这方面代表性工作包括：（a）初始测试剖面：Li等人提出了在测试过程中出现预定标准时测试剖面转换成理论最优剖面的最优动态随机测试技术[105]。（b）参数值：Yang等人提出了在测试的过程中动态调整参数的方法[106]；Lv等人通过理论分析的方式分析两个参数的比值在一定的区间时可以保证失效率大的分区选取概率不断增大[107]。

测试结果判定是检查测试用例对应的实际输出是否符合预期的步骤，其结果是判断待测软件中是否潜藏故障的重要依据。然而，在很多实际情况中，难以验证测试用例对应的实际输出结果是否满足预期，此问题称为软件测试中的“测试预期问题”。测试预期问题一直是软件测试领域中的热点与难点问题[106]，代表性的研究工作包括：N-version 测试[64]、断言[65]、机器学习[66]、假设检验[67]、蜕变测试[68]。其中，蜕变测试依据待测软件的蜕变属性获取蜕变关系，执行原始测试用例（采用测试用例生成技术获得）与衍生测试用例（根据蜕变关系获得），检查对应的输出是否违反蜕变关系。如果违反了某种蜕变关系，则表明存在故障。不难看出，蜕变测试无需测试预期，因此有效地缓解了测试预期问题，并在多个领域得到成功的应用[69]。近年来，研究者在蜕变测试理论与应用等方面取得了丰富的研究成果。部分代表性工作如下：

* **蜕变关系识别与复合方面：**Chen 等人提出了一种基于范畴划分的蜕变关系识别方法[70]；Zhang等提出一种基于搜索的蜕变关系推理方法[71]；Su 等人提出了一种蜕变关系动态识别方法[72]；Sun 等人提出一种数据变异指导的蜕变关系获取方法[73]；Gotlieb 等人提出了一种蜕变关系验证框架[74]，对于给定的程序的某个蜕变关系植入故障，利用约束逻辑编程技术生成满足植错后的蜕变关系的测试用例集，如果测试用例集检测出故障，则表明植错后的蜕变关系是错误的；Liu 等人提出一种蜕变关系复合方法，通过对多个蜕变关系的复合形成新的蜕变关系[75]。
* **蜕变测试原始测试用例生成方面：**Batra等人提出了一种基于遗传算法的测试用例生成方法，旨在最大化覆盖程序路径[76]；Dong等人采用符号执行提取蜕变关系并生成相应的测试用例[77]；Chen 等人比较了特殊值法与随机方法产生的原始测试用例对蜕变测试效率的影响[78]。
* **蜕变测试与其它测试技术结合：**Xie等人将蜕变测试与基于频谱分析的故障定位技术结合，提出了无需预期的故障定位方法[79]； Dong 等人将蜕变测试与遗传算法相结合，在搜索过程中使用蜕变关系，将原始测试用例和衍生测试用例都视为候选方案，加速达到某种覆盖标准[80]。
* **蜕变测试在不同领域中的应用：** Sun 等人提出了面向 Web 服务的蜕变测试方法 [81-82]，首先从 Web 服务的 WSDL 描述文档中提取蜕变关系，然后根据 WSDL 文档随机产生测试用例并且运用蜕变关系得到相应的衍生测试用例；Chan 等人提出了一种面向SOA 的蜕变测试方法，将离线测试通过的测试用例作为原始测试用例，在线测试用例作为衍生测试用例[83-84]；Mayer等人将蜕变测试应用于图像处理程序[85]；Kuo等人将蜕变测试应用于图像处理程序时检测出一个真实故障[86]；Tse等人尝试将蜕变测试应用于上下文敏感的中间件软件[87]；Chan 等人将蜕变测试应用于能量感知的无线传感器网络应用软件[88]；Sun 等人将蜕变测试成功地用于几类典型加密程序的测试[89]。

#### 1.2.4 深度学习系统的测试现状

在人工智能领域中，深度学习系统在近年受到学术界的广泛关注并得以快速发展，然而，在深度学习带给人们巨大便利的同时，其本身也存在一些隐含缺陷或者安全问题。对于一个非正常的输入，深度模型是否依然能够产生满意的结果，关于深度学习系统测试的重要性也渐渐引起研究者的注意，代表性的对深度学习系统进行测试的研究工作包括：

* **DeepXplore**：Kexin[1]等人首次提出对真实深度学习系统进行系统性白盒测试的框架。DeepXplore旨在解决两个难题：（1）生成能够激发一个深度学习系统逻辑的不同部分的输入（input）；（2）自动识别深度学习系统的不正确行为。该框架有4个主要思想：提出了神经元覆盖（neuron coverage）；将多个相似的 DNN 的输出与待测试的 DNN 的输出进行比较（差分测试（differential testing））；在实现深度学习算法高神经元覆盖率时，将找到触发不同行为的输入的过程表示为一个联合优化问题；然后使用基于梯度的优化技术解决上述优化问题。但是值的注意的是，完全的神经元覆盖并不能保证能找到所有的漏洞。
* **DLFuzz**：Guo提出用于指导DL系统暴露不正确的行为的模糊测试框架[3]。DLFuzz通过不断改变输入以最大化神经元覆盖，以及预测原始输入和变异输入的差异，无需手动标记或交叉引用来自其他DL系统的oracles。与DeepXplore相比，DLFzz是最先进的DL白盒测试框架，不需要为相似功能的DL系统做额外的交叉引用检查工作，但可以产生338.59％的对抗输入，同时减少89.82％的扰动，平均提高2.86％的神经元覆盖率，节省20.11％的时间代价。
* **DeepTest**：Tian等人提出的以神经元覆盖为导向的测试用例生成[4]。DeepTest利用了神经元覆盖的概念，通过一组测试输入能够激发的神经元数量，系统地探索DNN逻辑的不同部分，并从经验上证明，神经元的覆盖率变化与自动驾驶汽车(如转向角)的变化统计相关；除此之外DeepTest还对图像的转换生成增加神经元覆盖率的合成测试用例，在合成图像的基础上进行再训练以提升DNN效果。DeepTest的主要目的就是尽可能地得到更贴近现实状况的图像，以贴合真实的使用场景。但是，极端行为（corner-case behaviors）不是单靠图像转换操作就可以完全覆盖的。除此以外，对于自动驾驶汽车这一实际问题来说，不仅需要考虑方向盘转向，还有汽车的行驶速度、加速度等也需要考虑。
* **DeepCover**：Sun等人提出DNN测试标准DeepCover[5]。DeepCover受传统MC/DC覆盖标准的启发，依照传统软件测试方法中的测试覆盖标准和测试用例生成算法的思想，针对DNN的不同特征提出四个测试标准。对于每个标准，文章给出一种基于线性规划生成测试用例的算法，算法通过扰动任意给定的一个DNN的输入来产生新的测试用例。在MNIST数据集上进行实验后能够达到发现错误、DNN安全统计、提高测试效率以及DNN内部结构分析等目标。但是通过与其他论文对比，该研究仅在较小规模的神经网络上进行评估（不超过400个神经元），并没有在复杂的真实DL系统上进行实验，其测试标准的有效性有待验证。但是该系统已开源，可作为后续研究的基础。
* **DeepGauge**：Ma等人提出针对深度学习系统的多粒度测试覆盖方法[6]。DeepGauge仿照传统软件行为的划分将DNN的行为划分为主要功能行为（major function behaviors）和极端行为（coner-case behaviors）。基于对DNN内部状态的观察，提出了神经元粒度的覆盖标准和神经网络层粒度的覆盖标准，并与在DeepXplore中提出的神经元覆盖标准做出比较。论文还通过对DNN的内部状态的分析来评价测试套件的多样性，作为有效测试用例生成的指导。
* **DeepConcolic**: Sun等人提出了第一个用于深度神经网络（DNN）的分析测试方法（concolic testing approach）[7]。论文通过结合程序执行和符号分析，以探索软件程序的执行路径。使用各种覆盖指标指导生成测试输入，实现了一个导向性随机测试工具DeepConcolic，具有高覆盖率并且能够发现大量的对抗样本。。
* **DeepCT**：Ma等人提出的用于深度学习系统的组合测试方法[8]，DeepCT对DL系统上的组合测试（CT）进行了探索性研究。论文中通过调整CT中的概念进而提出一套用于DL系统的覆盖标准，以及用于CT覆盖引导的测试生成技术。论文实验结果表明CT为测试DL系统提供了一个很有前景的途径。
* **DeepMutation**：Ma等人提出的用于深度学习系统的变异测试框架[9]。DeepMutation是一种针对DL系统的变异测试方法，用于衡量测试数据的质量。与传统软件的变异测试类似，DeepMutation从数据、源码级别和模型级别角度设计变异算子。通过对数据引入变化、对源码进行变异操作、对模型注入错误等方法，将故障引入DL模型。最终可以从分析中评估测试数据的质量，并可以检测注入的故障程度，进而对测试数据质量进行改善。
* **DeepRoad**：Zhang等人提出的基于GAN的变异测试方法[10]。DeepRoad通过生成具有各种天气条件的驾驶场景，以检测自主驾驶系统的不一致行为。主要工作包括：1）DeepRoad自动合成大量不同的驾驶场景，而无需使用图像转换规则（例如，缩放，剪切和旋转）。特别是，DeepRoad能够生成具有各种天气条件的驾驶场景。2）DeepRoad利用变异测试技术来检查使用合成图像时系统行为的一致性。3）DeepRoad通过使用VGG网络特征计算输入图像和训练图像的距离来判定变异测试的程度。
* **FITEST**：Abdessalem提出了一种通过将检测特征交互的问题转换为基于搜索的测试生成问题，进而检测特征交互是否失败的技术。方法定义了一组混合测试目标（距离函数），将传统的基于覆盖的启发式方法与专门用于揭示特征交互失败的新启发式相结合。FITEST就是由混合测试目标指导的基于搜索的测试生成算法。 另外，FITEST提出了多目标进化算法，以减少计算适应度值所需的时间。实验使用两种版本的工业自动驾驶系统来验证方法的有效性。

#### 1.25软件安全漏洞识别及定位发展

软件系统的安全性是软件系统在应对非正常的操作或恶意攻击情况下依然能够满足可用性、授权访问、数据的机密性等需求的关键。软件存在的内在缺陷，且缺陷可能被利用造成系统安全状态受损或破坏，被称为软件脆弱性。软件开发者和组织为了提高软件系统的安全性，研究提出了一系列软件开发安全原则和最佳实践，然而由于程序的复杂性和软件的广泛应用，针对软件的恶意攻击行为和软件系统安全缺陷越来越多地涌现出来，成为威胁信息系统安全的主要原因之一。目前关于软件缺陷的研究主要集中在以下两方面：

1.软件的脆弱性检测（Security vulnerability detection）

软件脆弱性分析是检查并发现软件系统中的内在安全缺陷和利用途径的主要手段之一，一直是软件安全领域的研究热点。传统的软件脆弱性检测主要依靠程序分析技术，如静态分析和动态分析，然而静态分析技术普遍存在误报率较高以及过分依赖检测规则的缺点，而动态分析技术则由于难以覆盖全部路径导致漏报问题并且有较大的时间开销和空间开销。近几年，面对静态和动态脆弱性分析方法的技术瓶颈，研究者开始重视软件漏洞或者脆弱性历史数据的挖掘与分析，并且随着开源代码量的增加，利用深度学习技术从大量的代码中学习软件漏洞模式也成为了可能，这为代码漏洞检测的研究提供了新的方向。国内外对于代码漏洞检测技术已经有了很多研究，这些技术大体上可以分成两类：基于程序分析的方法和基于统计模型的方法。

（1）基于程序分析的方法

基于程序分析的漏洞检测技术又可以分为静态分析技术和动态分析技术。静态分析技术在不执行程序的情况下获得程序的编译信息，并根据这些信息进行漏洞检测。DA Wagner等人[137]将静态分析技术用于缓冲区溢出和入侵检测，证实静态分析技术可以作为保证软件安全的有效手段。Kong等人[138]实现了一个基于数据融合的静态分析系统ISA，他们通过将不同的静态分析工具的检测结果进行有效融合，取得了更好的漏洞检测效果。静态分析克服了人工分析效率低下的缺点，但是普遍存在误报率较高的问题，并且可扩展性差，同时忽略了对程序功能的检测。

动态分析方法通过执行程序来得到程序的运行状态和运行数据，并利用这些运行信息对软件漏洞进行检测。动态分析是在程序运行时进行检测，因此准确率高。而且动态分析还具有针对性强、不需对源代码进行挖掘等优点。但是动态分析存在分析结果不完整的问题，一次或多次的程序执行情况并不能代表程序以后所有可能的运行情况。同时，动态分析对于检测人员的技术和经验也有着较高的要求。许多研究采用了模糊测试、渗透测试等方法检测软件漏洞。Wu等人[139]提出了一种多维模糊测试方法，并且避免了组合爆炸问题，该方法能够挖掘到单维模糊测试技术无法挖掘到的软件漏洞。Holler等人[140]提出了一种模糊测试方法LangFuzz，并应用到Mozilla Javascript解释器和PHP解释器的安全漏洞检测中，取得了很好的效果。Tian等人[141]提出了一种基于模型的渗透测试方法来检测SQL注入漏洞，该方法产生的测试用例能够覆盖更多的SQL注入攻击类型。

（2）基于统计模型的方法

近年来，机器学习和深度学习在语音识别、图像处理、自然语言处理等领域已经取得了重大成功。随着开源代码在数量和种类上的不断丰富，直接从大量的代码中学习软件漏洞模式也成为了可能。已经有许多研究将机器学习和深度学习技术应用到漏洞检测中，取得了很好的效果。Yamaguchi等人[142]将代码表示成向量，并利用主成分分析技术提取API使用模式，从而能够识别具有相似特征的代码漏洞。Pang等人[143]结合n元语法模型和基于机器学习的特征选择算法进行软件漏洞预测，并将该方法用于Java Android 应用的漏洞检测。除了传统的机器学习技术外，Li等人[144]首次将深度学习用于代码漏洞检测，他们根据代码中的库函数和API调用来提取程序切片并表示成向量，然后用Bi-LSTM训练漏洞预测模型，并用来检测两种类型的漏洞。随后，Li等人[145]又进一步提出了漏洞检测框架SySeVR，该框架利用代码的语法和语义信息生成代码的特征向量，并利用Bi-GRU训练模型，该框架能够检测等多类型的漏洞并取得更好的检测效果。

不过，目前基于统计模型的代码检测技术仍然存在一些问题有待进一步解决。比如如何将代码转换成能够体现代码的语义信息和结构信息的向量表示就是目前亟待解决的重要问题。

2.安全缺陷报告识别（Security bug report identification）

安全缺陷报告识别通常采用自然语言处理和机器学习技术，通过挖掘历史缺陷报告中自然语言描述的文本信息和缺陷分类信息，将安全缺陷报告识别转化为一个分类、预测或者推荐问题，其中文本挖掘的质量对模型性能的影响很大，因此有效地挖掘缺陷报告中的文本信息是提高模型性能的关键问题。

Gegick[8]根据缺陷报告中自然语言描述的文本信息，利用词袋模型根据从缺陷报告中抽取的关键词将所有缺陷报告表示成一个词项-文档矩阵，并使用该矩阵和相应的缺陷类别标签作为输入来训练统计模型，实现安全缺陷报告的识别。考虑到不同的关键词在识别安全缺陷报告时具有不同的权重，Behl[9]在利用词袋模型表示缺陷报告的基础上，引入了TF-IDF值作为词的权重并通过朴素贝叶斯模型来实现安全缺陷报告识别。Chawla[10]进一步利用了缺陷报告所包含的语义信息，通过TF-IDF、LSI模型，以及多项式朴素贝叶斯模型对缺陷报告进行分类。而Zou[11]不仅使用缺陷报告的文本信息，还充分利用了缺陷报告中的非文本信息来训练模型，以提高安全缺陷报告识别的准确率。

以上基于文本挖掘的安全缺陷报告识别模型面临的主要难点问题是：类不平衡、数据噪音和数据稀疏问题。

由于在安全缺陷报告识别的任务中，训练数据存在严重的类不平衡问题，即安全缺陷报告的数量远远小于非安全缺陷报告的数量，这使得预测模型往往具有较高的准确率，但召回率较低，召回率低意味着漏检率高，即未被模型正确识别的安全缺陷报告数量较多，这给软件带来的安全隐患是无法容忍的，因此一些学者在针对安全缺陷报告识别的类不平衡问题进行了研究。Yang[12]利用词袋模型将缺陷报告表示成向量，并分别利用四种不平衡数据的处理方法（随机欠采样、随机过采样、合成少数类样本的过采样、代价矩阵调节方法）对训练集进行预处理，再利用预处理后的训练集训练分类器来降低类不平衡对安全缺陷报告分类造成的影响。Zhou[13]提出了K折叠加算法来集成多个个体分类器，利用集成学习缓解类不平衡的影响。Postojanova[14]提取了三种缺陷报告的特征向量：二元词袋频率（BF）、词频（TF）和TF-IDF，并分别利用监督学习和基于异常监测的非监督学习算法来识别安全缺陷报告，利用非监督学习克服监督学习需要大量人工标注数据以及准确标注为安全缺陷的数据较少带来的类不平衡问题。

除了类不平衡问题外，数据噪音也是导致模型召回率较低的原因之一。Peters[15]的研究发现一些非安全缺陷报告同样包含和安全相关的关键词，同时出现在安全和非安全缺陷报告中的安全相关关键词被称为安全交叉词，包含安全交叉词的非安全缺陷报告作为训练数据集相当于在模型的训练中引入了噪音，从而增大了模型将安全缺陷报告误标记为非安全缺陷报告的概率，而这种不利的影响在类不平衡的情况下又被进一步扩大。因此他们提出了一个称为FARSEC的框架，其主要思想是在模型训练前将含有安全交叉词的非安全缺陷报告从训练集中移除，从而降低数据噪音对安全缺陷报告识别的影响。该方法虽然通过过滤部分含有安全交叉词的非安全缺陷报告缓解了类不平衡的影响，但类不平衡问题依然严重，而且由于该方法仅根据TF-IDF值最高的前100个词作为安全相关的关键词，未考虑缺陷报告的内容以及项目包含的实际安全相关关键词的规模，召回率的提高依然是有限的，并且还增大了误检，为了降低误检的影响，该方法将安全报告的识别转化为推荐问题，通过生成一个安全缺陷报告的排序列表，来缓解安全工程师确认安全缺陷的工作量。

数据稀疏问题主要是指缺少项目的历史数据或者项目历史数据中包含的安全缺陷报告极少，解决这一问题的主要方法是采用迁移学习的方法，利用项目外的其他项目作为训练数据集训练模型，从而实现跨项目的安全缺陷报告识别。

在安全缺陷报告定位方面的研究主要集中在以下方面：

（1）基于信息检索技术的故障定位

Gay等人[16]采用信息检索技术，将初始缺陷报告视为查询，并根据与查询的相关性对源代码文件进行排名，从而确定包含缺陷的源文件。

Ye X等人[17]通过计算缺陷报告与源代码文件的相关度来对可能存在缺陷的源文件进行排名，他们采用的四种相关度计算方法，包括：1）通过简单的向量空间模型（VSM，Vector Space Model）来计算源码和缺陷报告的相似性。2）利用API描述来弥补缺陷报告和源代码之间的词汇鸿沟。3）利用先前已修复的缺陷报告的创建时间来辅助计算相关度排名。4）采用协同过滤的方法，通过检查源码文件之前所涉及的所有缺陷报告并将其提取摘要，然后和新的缺陷报告计算余弦相似度来对缺陷报告对应的修复文件进行排名。

2016年Ye X等人[18]又对之前的方法提出了改进，扩充了另外两种对源代码文件排名有参考价值的方法：1）将源代码文件分解为类库、方法库、变量库和评论库，然后分别与新缺陷报告的摘要和描述做相似度计算。2）通过源文件的依赖关系建立文件网络图，然后用类似PageRank的方法计算新缺陷报告与源文件的相似度。

2017年Reza等人[133]提出了一种结合缺陷报告不同方面的文本特征来来进行缺陷定位的模型。这个模型包括三个部分：Token Matching Component, Similarity Based Component, Classification Component。其中，Token Matching Component用来找出那些在缺陷报告和源文件中都出现的字符，得到一个分数。Similarity Based Component利用修正词向量空间模型(rVSM)得到另一个分数。Classification Component利用历史修复的缺陷报告和一个多标签分类算法算出最后一个得分。最后将三个部分的得分进行加权组合。这个方法考虑到了缺陷报告不同方面的文本特征，在两个开源项目SWT和ZXing上的效果都很好。

（2）基于深度学习的故障定位

An Ngoc Lam等人[19]将深度学习技术应用于故障定位。他们使用改进的向量空间模型来计算缺陷报告和源文件之间的相似度，并用深度神经网络（DNN，Deep Neural Networks）来学习缺陷报告的具体术语和源文件文本信息之间的关联，最后将两者结合，给出源代码文件和缺陷报告的相似度排名。

LiMing等人[134]提出了NP-CNN，NP-CNN主要包括两个连续的部分，第一部分是infra-language feature 提取层，是基于多层卷积层来提取缺陷报告和源代码的特征。第二部分是cross-language feature融合层，用来组合上一层提取出的两种特征。在这之前的方法对缺陷报告和源代码文件的相似性度量都是在相同的词向量空间，忽略了源代码文件的结构信息。NP-CNN同时利用了结构信息和词汇信息，达到了更好的效果。

LiMing等人[135]后来又提出了LSTM-CNN，利用LSTM把代码语句序列的长期依赖也考虑进来。LS-CNN是LSTM和CNN的结合，这种方法的关键就是 intra-language feature extraction网络，利用CNN提取源代码局部和结构信息，尽管之前的NP-CNN在对源代码进行卷积的时候在某种程度上考虑了序列信息，但是没有考虑到长期依赖的关联。利用LSTM提取能够反应序列特征的语义信息并解决语句间的长期依赖问题。

1. 基于信息检索和深度学习相结合的故障定位

An Ngoc Lam等人[136]提出了一种利用DNN和rVSM结合的方法。其中rVSM用来提取缺陷报告和源文件的文本相似性，DNN用来学习缺陷报告和源文件的关联度。考虑到最近修复的源文件相比于很久没有进行修复的源文件有更大可能含有错误，所以元数据得分对最近修复的缺陷报告有所侧重，最后对文本相似性得分，关联度得分，元数据得分三个分数进行加权组合。这种方法利用信息检索的方法提取缺陷报告和源文件的文本相似性，再利用深度学习来学习缺陷报告和源文件的关联度，实验表明两者结合的效果比单独使用任何一种模型都要好。

（4）其他故障定位方法

2018年Mehran Hassani等人[116] 对日志相关bug的不同方面进行了特征研究，即包含与日志相关的bug，报告和修复时间以及开发者修改过程。发现造成日志相关bug的七个根本原因。提出了一种自动化工具，可以从源代码中检测四种不同类型的与日志相关的bug。2018年Boyuan Chen 等人[117]提出一个利用执行日志自动评估代码覆盖范围的新方法LogCoCo，使用程序分析技术匹配执行日志。LogCoCo可以通过比较和研究他们的代码覆盖率指标，评估和提高各种测试套件（单元测​​试，集成测试和基准测试）的质量。该项目是与百度合作完成的，该服务被数亿万用户使用，也与有合作企业考虑采用LogCoCo，表明该方法有用且实用。

2018年Afsoon Afzal [115]创建了一个先进的ArduPilot自动化测试的高级框架系统，目标是开发一个低保真软件仿真自动化系统，实现bug自动检测、定位和修复。提出了一种新的自动推理系统模型的表示方法并生成测试套件，以增强自动故障定位性能和描述失败的根本原因。鼓励和支持机器人专家提前在模拟中系统地检测机器人系统，降低成本和失败的危险。Afsoon Afzal等人提出了一种叫作Houston的高层次框架用来测试ARDUPILOT系统[146]，可以利用delta调试简化测试命令[147]。

### 1.3 智能化软件特点及测试存在的问题

与传统软件相比，智能化软件系统呈现出**学习深层知识、融合跨界数据、数据处理层级化、决策逻辑不受控、系统输出难以验证、集成群体智慧**的特点，具体来说：

**（1）在学习深层知识方面：**智能化软件系统能够从大数据表示的知识中进行深层次的认知、学习、推理，并通过动态调整自身业务逻辑实现持续演化，适应外部动态变化的需求，而传统软件依据既定的业务逻辑执行各种操作，难以进行自主演化并适应外部环境变化。

**（2）在融合跨界数据方面：**智能化软件系统将跨领域的多源异构数据进行协同处理，实现跨界信息的关联融合，而传统软件系统仅仅分类型处理这些多源异构数据，难以实现信息的深度融合；多领域的数据进行组合导致智能化软件系统的输入空间异常庞大，此外，大量异构的数据造成智能化软件系统数据预处理模块异常复杂。

**（3）在数据处理方面：**智能化软件系统首先采集数据，并对采集的数据进行预处理，得到符合决策模块输入类型的规则数据，然后将规则数据输入决策模块，并得到智能化软件系统的输出。智能化软件系统多层次的数据处理过程导致数据出错的可能性增加。

**（4）在获得决策逻辑方面：**智能化软件系统的决策模块通常由程序开发人员利用高级语言（Java、Python等）实现，但是决策的逻辑从数据中习得，而不是开发人员指定。这种“在代码上堆代码”的方式让系统开发人员以及测试人员难以理解。

**（5）在验证测试结果方面：**智能化软件系统多源化的输入、难以理解的内部逻辑以及所处场景的多样性使得测试人员判断系统行为耗时、耗力。

**（6）在集成群体智慧方面：**智能化软件系统可依托互联网或大数据无缝整合多种智能，形成群体智慧，而传统软件系统更加聚焦于个体的智能。

由于上述新特点，**智能化软件系统的测试面临诸多新的问题与挑战，具体来说：**

**（1）智能化软件系统的测试任务难以准确描述**：传统软件系统的运行时功能依据明确定义的规格说明实现，因此，软件测试人员可以根据规格说明清晰且准确地了解与描述系统的待测功能，并以此制定测试计划与任务。然而，智能化软件系统具有认知、学习、推理的能力，其功能随着应用场景与任务的变化、认知范围的增加和学习内容的改变而发生变化与衍进，难以预测且呈现出智能性。这些问题导致软件测试人员难以清晰而准确地了解与描述其待测功能。在此情况下，软件测试人员难以对智能化软件系统制定测试任务，从而难以实施有组织有目标的软件测试。

**（2）智能化软件系统的模拟测试平台难以搭建：**智能化软件系统通常与外界环境存在密切的交互，所以智能化软件系统的测试无可避免地涉及到与环境的交互。然而，在真实环境中测试智能化软件系统将产生高昂且难以负担的成本。一个较好的解决方案是搭建智能化软件系统的模拟环境测试平台。如何让模拟测试平台接近智能化软件系统的真实应用场景是一个重要的问题。通常，模拟测试平台需要模拟三大类事物，包括人、人造事物、自然事物。 模拟人与自然事物存在诸多困难，其原因是人类在不同场景下的行为是不确定的，很多自然事物的规律及内在机制至今仍未研究透彻。因此，搭建智能化软件系统的模拟测试平台存在诸多困难。

**（3）测试用例的可靠性与科学性难以验证：**智能化软件系统的测试数据可能不准确并且基于学习的智能化软件系统的学习模型经常出现过拟合的现象，出现不正确或者不被期待的行为。其次，智能软件系统的模块交互复杂，输入空间巨大，也有可能存在一些约束关系，测试用例的能否正常执行尚且认识不足，所以按照软件测试生成算法生成的测试用例很有可能难以执行。因此，保证测试用例的科学性需要考虑。

**（4）测试判定的困难：**目前主要的判定有两种：（a）将系统的行为与执行数据对应的标签对比；（b）测试人员判断。这两种方式都需要人工参与，然而在长时间的工作下，资深测试工程师的准确性也不能得到保证。对于智能化软件系统而言，系统的可靠性往往较高，所以通常不会出现软件崩溃等执行结果，更多的可能是与预期结果相比微小的执行差异，包括色彩、音频、数据等，如果单纯依靠测试人员进行主观判断，那么工作量巨大的同时，精确度也很难保证，所以对于智能软件系统的测试执行判定，我们需要开发出更加智能化的判定方式。

**（5）测试充分性无法保证：**智能化软件系统的输入空间十分巨大，例如无人驾驶汽车通过各种传感器获取无数种可能的外界信息。目前的测试技术主要通过三种方式获得测试数据：**（a）随机生成测试用例：**随机地在测试用例集中挑选测试用例或者随机地生成测试数据具有简单、易用的特点。该方法被谷歌等世界著名公司使用来生成无人驾驶车的测试数据[90-92]。然而，随机的方式没有利用任何系统内部信息以及测试过程信息，使得很多智能系统不正确的行为不能被发现[93-94]；**（b）人工生成测试用例：**该方式通过在普通测试数据的基础上做微小改动来得到新的测试数据[95-96]，然后测试人员判断每一个新测试数据对应的行为。这种方式成功地运用在多种智能化软件系统中[97-100]。随机生成和人工生成测试用例的方式都没有考虑智能化软件系统的内部结构，不能覆盖智能化软件系统的大部分逻辑，导致了这些测试用例只能发现少量的系统异常行为。Pei等人通过经验研究发现：随机生成一个测试用例几乎能够覆盖无人驾驶车的所有代码，但是只激活了不到 10%的神经元。基于上述观察，Pei等人提出了神经元覆盖指标，评估测试用例集激活的神经元数目[94]。Tian等人利用神经元覆盖指标，生成测试用例，激活智能系统大多数的神经元，并通过经验研究的方式验证该方法生成的测试用例集可以发现更多智能系统不正确的行为；**（c）基于场景和功能：**基于场景的测试技术通过给定的场景及任务测试智能化软件系统的实际表现。例如，前文提到的 DARPA 无人驾驶挑战赛中，无人驾驶汽车需要在指定时间内安全地穿越莫哈韦沙漠中的一个区域。对于一些较为简单的智能化软件系统的测试，测试人员可以枚举测试场景及任务以验证系统在这些场景下的表现是否符合预期。但是，对于复杂的智能化软件系统，穷举测试场景与任务是十分耗时耗力且效率低下的。并且，在组合场景与任务下出现的组合爆炸问题使这种技术难以应用。此外，这种技术的测试结果仅从宏观层面对智能化软件系统进行定性评价[44]，难以从微观层面对智能化软件系统的功能质量进行定量评价。基于功能的测试技术将智能化软件系统按功能划分子模块，并针对每个子模块生成测试用例并针对功能进行测试。例如，一个无人驾驶系统可以被划分为感知及识别模块、决策模块及动作执行模块[45][46]。该技术为这些模块分别生成测试用例，并对模块的功能正确性进行检验。

**（6）智能化软件系统的质量评价指标及模型存在缺失：**对于传统的软件系统，测试人员可根据McCall质量模型、ISO/IEC 25010质量模型等成熟的软件质量模型为待测系统选定质量特性，并评估系统在测试中（的表现是否）满足给定需求或特性的程度。智能化软件系统与传统软件系统最大的不同是前者所特有的人工智能，因此，对智能化软件系统的质量进行评价不可不免地要涉及到对系统智能程度的评估。现有的智能度评估方法主要有三类[109]，包括人工辨别法、基准任务法、伙伴对抗法，然而这三类方法分别存在实验设置困难、评估结果片面、结果依赖于对矿伙伴的问题，且未有较为系统的评价指标及模型。有研究者提出通过评价智能系统与人类在行为上的相似性来评估智能系统的质量[48][49]，但仍然没有合适的度量标准。此外，很多智能化软件系统是多目标的，例如，无人驾驶系统会考虑驾乘舒适性、燃油消耗等。智能化软件系统在多目标情景下会依据目标优先级的不同而展现出不同的行为[42]。在缺乏针对性质量评价指标及模型的情况下，难以对智能化软件系统的质量进行系统有效的评估。

**（7）缺少对安全缺陷的识别和调试支持：防危性（**Safety）和安全性（Security）是智能软件系统的两个重要属性。尽管智能软件系统工程师可能为系统写下了很多安全规则，但也很难穷尽所有的情况。国家互联网应急中心发布了《开源软件代码安全缺陷分析报告——人工智能类开源软件专题》，对国际、国内知名的开源软件项目进行了评测，结果表明，这些人工智能类开源软件都存在不同程度的安全问题。Security漏洞无法被及时地修复，容易被恶意攻击者所利用，破坏系统；safety漏洞无法被及时识别和修复，则会有可能引发严重的安全事故。特别是随着智能软件系统越来越多地应用于医疗、交通、安全和国防等安全关键的领域中，如何检查和调试安全相关的软件缺陷，尽早消除近期和长期可能的危险，防御恶意攻击，保证系统可靠地工作，成为智能软件系统评测中的一项关键任务。然而由于智能软件系统的特殊性，用于测试、调试、验证软件的传统方法根本不适用于智能软件系统中，需要开发识别和调试安全漏洞的新方法。

综上所述，智能化软件系统测试方面的研究工作仍存在诸多不足，**亟待探索有效且高效的新型智能化软件系统的测试方法与技术。本课题旨在针对智能化软件系统测试的测试方面的问题与挑战，探索智能化软件系统的测试关键技术，以系统有效的方式检测与评估智能化软件系统的质量，为开发可靠的智能化软件系统提供新型测试理论与工具支持。**

**参考文献**

1. Pei, Kexin, et al. "Deepxplore: Automated whitebox testing of deep learning systems." Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017.
2. Yosinski, Jason, et al. "Understanding neural networks through deep visualization." arXiv preprint arXiv:1506.06579 (2015).
3. Guo, Jianmin, et al. "DLFuzz: differential fuzzing testing of deep learning systems." Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2018.
4. Tian, Yuchi, et al. "Deeptest: Automated testing of deep-neural-network-driven autonomous cars." Proceedings of the 40th International Conference on Software Engineering. ACM, 2018.
5. Y. Sun, X. Huang, and D. Kroening, “Testing Deep Neural Networks,” ArXiv e-prints, Mar. 2018.
6. Ma, Lei, et al. "Deepgauge: Multi-granularity testing criteria for deep learning systems." Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018.
7. Sun, Youcheng, et al. "Concolic Testing for Deep Neural Networks." arXiv preprint arXiv:1805.00089 (2018).
8. Ma, Lei, et al. "Combinatorial Testing for Deep Learning Systems." arXiv preprint arXiv:1806.07723 (2018).
9. Ma, Lei, et al. "DeepMutation: Mutation Testing of Deep Learning Systems." arXiv preprint arXiv:1805.05206 (2018).
10. Zhang, Mengshi, et al. "DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing." arXiv preprint arXiv:1802.02295 (2018).
11. Abdessalem, Raja Ben, et al. "Testing autonomous cars for feature interaction failures using many-objective search." Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018.
12. Chen, Boyuan, et al. "An automated approach to estimating code coverage measures via execution logs." Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018.
13. Han, Xue, Tingting Yu, and David Lo. "PerfLearner: learning from bug reports to understand and generate performance test frames." Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018.
14. V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, J. K. Hedrick. Learning a

Deep Neural Net Policy for End-to-End Control of Autonomous Vehicles. *Proceedings of the 2017 American Control Conference* (ACC’17), IEEE Computer Society, 2017, pp. 4914-4919.

1. K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition.

*Proceedings of the IEEE conference on computer vision and pattern recognition* (CVPR’16), IEEE Computer Society, 2016, pp. 770-778.

1. M. A. O. Vasilescu, D. Terzopoulos. Multilinear Image Analysis for Facial Recognition. *Proceedings of the 16th In International Conference on Pattern Recognition* (ICPR’02), IEEE Computer Society, 2002, pp. 511-514.
2. W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, G. Zweig. Toward Human Parity in Conversational Speech Recognition. *Proceedings of the IEEE/ACM Transactions on Audio, Speech and Language Processing* (TASLP’17), IEEE Computer Society, 2017, pp. 2410-2423.
3. Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.
4. Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676): 354-359.
5. <https://www.iyiou.com/p/73145.html>.
6. <http://m.elecfans.com/article/674086.html>.
7. <https://deepmind.com/blog/alphafold/>.
8. <https://search.proquest.com/docview/2015377429?pq-origsite=gscholar>.
9. <http://www.theverge.com/2016/2/29/11134344/google-selfdriving-car-crash-reprt>A Google Self-Driving Car Caused a Crash for The Frst Time.
10. [https://electrek.co/2016/07/01/understandingfatal-tesla-accident-autopilot-nhtsa-p robe/](https://electrek.co/2016/07/01/understandingfatal-tesla-accident-autopilot-nhtsa-p%20robe/), Understanding the Fatal Tesla Accident on Autopilot and the NHTSA Probe.
11. <https://en.wikipedia.org/wiki/Tay_(bot)>.
12. G. J. Myers, C. Sandler, T. Badgett. The Art of Software Testing. John Wiley and Sons, 2011.
13. Anand S, Burke E K, Chen T Y, et al. An orchestrated survey of methodologies for automated software test case generation[J]. *Journal of Systems and Software*, 2013, 86(8): 1978-2001.
14. Baldoni R, Coppa E, D’elia D C, et al. A survey of symbolic execution techniques[J]. *ACM Computing Surveys* (CSUR), 2018, 51(3): 50:1-50:39.
15. King J C. A new approach to program testing[C]. In *Proceedings of the International Conference on Reliable Software*. ACM, 1975, 228-233.
16. De Moura L, Bjørner N. Z3: An efficient SMT solver[C]. In *Proceedings of the 14th International conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS’08). Springer, 2008: 337-340.
17. Ganesh V, Dill D L. A decision procedure for bit-vectors and arrays[C]// In *Proceedings of the 19th International Conference on Computer Aided Verification* (CAV’07). Springer, 2007: 519-531.
18. Choco Solver [EB/OL]. [2018-11-18]. <http://www.choco-solver.org/>.
19. Chipounov V, Georgescu V, Zamfir C, et al. Selective symbolic execution[C]// In *Proceedings of the 5th Workshop on Hot Topics in System Dependability* (HotDep). 2009.
20. Odefroid P, Klarlund N, Sen K. DART: directed automated random testing[C]. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI’15). ACM, 2005, 213-223.
21. Cadar C, Dunbar D, Engler D R. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]// In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (OSDI’08). 2008, 209-224.
22. Godefroid P, Levin M Y, Molnar D A. Automated whitebox fuzz testing[C]// In *Proceedings of the 15th Annual Network and Distributed System Security Symposium* (NDSS’08). 2008, 151-166.
23. Dias Neto A C, Subramanyan R, Vieira M, et al. A survey on model-based testing approaches: a systematic review[C]. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, in conjunction with the *22nd IEEE/ACM International Conference on Automated Software Engineering* (ASE’07). ACM, 2007: 31-36.
24. Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey[C]. In *Proceedings of the IEEE*. IEEE Computer Society, 1996, 84(8): 1090-1123.
25. Kansomkeat S, Rivepiboon W. Automated-generating test case using UML statechart diagrams[C]// In *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology* (SAICSIT’03). South African Institute for Computer Scientists and Information Technologists, 2003: 296-300.
26. Cohen D M, Dalal S R, Fredman M L, et al. The AETG system: An approach to testing based on combinatorial design[J]. *IEEE Transactions on Software Engineering*, 1997, 23(7): 437-444.
27. Lei Y, Kacker R, Kuhn D R, et al. IPOG: A general strategy for t-way software testing[C]//In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (ECBS’07). IEEE Computer Society, 2007: 549-556.
28. Lei Y, Kacker R, Kuhn D R, et al. IPOG/IPOG-D: efficient test generation for multi‐way combinatorial testing[J]. *Software Testing, Verification and Reliability*, 2008, 18(3): 125-148.
29. Ghazi S A, Ahmed M A. Pair-wise test coverage using genetic algorithms[C]. In *Proceedings of the Congress on Evolutionary Computation* (CEC’03). IEEE Computer Society, 2003, 1420-1424.
30. Bryce R C, Colbourn C J. One-test-at-a-time heuristic search for interaction test suites[C]. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (GECCO’07). ACM, 2007: 1082-1089.
31. Williams A W. Determination of test configurations for pair-wise interaction coverage[M]. Testing of Communicating Systems. Springer, Boston, MA, 2000: 59-74.
32. Kobayashi N, Tsuchiya T, Kikuno T. A new method for constructing pair-wise covering designs for software testing[J]. *Information Processing Letters*, 2002, 81(2): 85-91.
33. Calvagna A, Fornaia A, Tramontana E. Random versus combinatorial effectiveness in software conformance testing: A case study[C]. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (SAC ’15). ACM, 2015: 1797-1802.
34. Vilkomir S, Starov O, Bhambroo R. Evaluation of t-wise approach for testing logical expressions in software[C]. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation Workshops* (ICSTW’13). IEEE, 2013: 249-256.
35. Medeiros F, Kästner C, Ribeiro M, et al. A comparison of 10 sampling algorithms for configurable systems[C]. In *Proceedings of the 38th International Conference on Software Engineering* (ICSE’16). ACM, 2016: 643-654.
36. Chen T Y, Leung H, Mak I K. Adaptive random testing[C]. In *Proceedings of the 9th Annual Asian Computing Science Conference* (ASIAN’04). Springer, 2004: 320-329.
37. Chen T Y, Kuo F C, Merkel R G, et al. Mirror adaptive random testing[J]. *Information and Software Technology*, 2004, 46(15): 1001-1010.
38. Chen T Y, Merkel R, Wong P K, et al. Adaptive random testing through dynamic partitioning[C]. In *Proceedings of the 4th International Conference on Quality Software* (QSIC’04). IEEE, 2004: 79-86.
39. Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. *IEEE Transactions on Software Engineering*, 2015, 41(5): 507-525.
40. Li L, Lin Y L, Zheng N N, et al. Artificial intelligence test: a case study of intelligent vehicles[J]. *Artificial Intelligence Review*, 2018: 1-25.
41. Campbell M, Egerstedt M, How J P, et al. Autonomous driving in urban environments: approaches, lessons and challenges[J]. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2010, 368(1928): 4649-4672.
42. Li L, Huang W L, Liu Y, et al. Intelligence testing for autonomous vehicles: a new approach[J]. *IEEE Transactions on Intelligent Vehicles*, 2016, 1(2): 158-166.
43. Huang W L, Wen D, Geng J, et al. Task-specific performance evaluation of UGVs: case studies at the IVFC[J]. *IEEE transactions on Intelligent Transportation Systems*, 2014, 15(5): 1969-1979.
44. Li L, Wen D, Zheng N N, et al. Cognitive cars: A new frontier for ADAS research[J]. *IEEE Transactions on Intelligent Transportation Systems*, 2012, 13(1): 395-407.
45. Pei K, Cao Y, Yang J, et al. DeepXplore: Automated whitebox testing of deep learning systems[C]//In *Proceedings of the 26th Symposium on Operating Systems Principles* (SOSP’17). ACM, 2017: 1-18.
46. Argall B D, Chernova S, Veloso M, et al. A survey of robot learning from demonstration[J]. *Robotics and autonomous systems*, 2009, 57(5): 469-483.
47. Kuefler A, Morton J, Wheeler T, et al. Imitating driver behavior with generative adversarial networks[C]// In *Proceedings of IEEE Intelligent Vehicles Symposium* (IV). IEEE, 2017: 204-211.
48. Tian Y, Pei K, Jana S, et al. DeepTest: Automated testing of deep-neural-network-driven autonomous cars[C]. In *Proceedings of the 40th International Conference on Software Engineering* (ICSE’18). ACM, 2018: 303-314.
49. Harman M, Mansouri S A, Zhang Y. Search based software engineering: A comprehensive analysis and review of trends techniques and applications[R]. Technical Report TR-09-03, Department of Computer Science, King’s College London, 2009.
50. Ahmed M A, Hermadi I. GA-based multiple paths test data generator[J]. *Computers & Operations Research*, 2008, 35(10): 3107-3124.
51. Alshraideh M, Bottaci L. Search-based software test data generation for string data using program-specific search operators[J]. S*oftware Testing, Verification and Reliability*, 2006, 16(3): 175-203.
52. Xiao M, El-Attar M, Reformat M, et al. Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques[J]. *Empirical Software Engineering*, 2007, 12(2): 183-239.
53. Girgis M R. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm[J]. *Journal of Universal Computer Science*, 2005, 11(6): 898-915.
54. Ghiduk A S, Harrold M J, Girgis M R. Using genetic algorithms to aid test-data generation for data-flow coverage[C]//In *Proceedings of the 14th Asia-Pacific Software Engineering Conference* (APSEC’07). IEEE, 2007: 41-48.
55. Li H, Lam C P. An ant colony optimization approach to test sequence generation for state-based software testing[C]// In *Proceedings of the 5th International Conference on Quality Software* (QSIC’05). IEEE Computer Society, 2005, 255-264.
56. Li H, Lam C P. Using anti-ant-like agents to generate test threads from the UML diagrams[C]//In *Proceedings the 17th IFIP International Conference on Testing of Communicating Systems* (TestCom’05). Springer, 2005: 69-80.
57. Lefticaru R, Ipate F. Automatic state-based test generation using genetic algorithms[C]//In *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC’07). IEEE, 2007: 188-195.
58. Emer M C F P, Vergilio S R. GPTesT: A testing tool based on genetic programming[C]//In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (GECCO’02). Morgan Kaufmann Publishers Inc., 2002: 1343-1350.
59. Masud M, Nayak A, Zaman M, et al. Strategy for mutation testing using genetic algorithms[C]//In *Proceedings of Canadian Conference on. Electrical and Computer Engineering*. IEEE, 2005: 1049-1052.
60. Baudry B, Fleurey F, Jézéquel J M, et al. Automatic test case optimization: A bacteriologic algorithm[J]. *IEEE Software*, 2005, 22(2): 76-82.
61. Baudry B, Fleurey F, Jézéquel J M, et al. From genetic to bacteriological algorithms for mutation-based testing[J]. *Software Testing, Verification and Reliability*, 2005, 15(2): 73-96.
62. S. S. Brilliant, J. C. Knight, P. E. Ammann. On the Performance of Software Testing Using Multiple Versions. *Proceedings of the 20th international symposium on fault-tolerant computing* (FTCS’90), IEEE Computer Society, 1990, pp. 408–415.
63. K. Y. Sim, C. S. Low, F. C. Kuo. Eliminating Human Visual Judgment from Testing of Financial Charting Software. *Journal of Software*, 2014, 9(2): 298-312.
64. W. K. Chan, S. C. Cheung. PAT: A Pattern Classification Approach to Automatic Reference Oracles for the Testing of Mesh Simplification Programs. *Journal of Systems and Software*, 2009, 82(3), 422–434.
65. R. Guderlei, J. Mayer. Statistical Metamorphic Testing Testing Programs with Random Output by Means of Statistical Hypothesis Tests and Metamorphic Testing. *Proceedings of the 7th international conference on quality software* (QSIC’07), IEEE Computer Society, 2007, pp. 404–409.
66. T. Y. Chen, S. C. Cheung, S. M. Yiu. Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.
67. S. Segura, G. Fraser, A. B. Sanchez, A. R. Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 2016, 42(9): 805-824.
68. T. Y. Chen, P. L. Poon, X. Xie. METRIC: Metamorphic Relation Identification Based on the Category-Choice Framework. *Journal of Systems and Software*, 2016, 116: 177-190.
69. J. Zhang, J. Chen, D. Hao. Search-Based Inference of Polynomial Metamorphic Relations. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (ASE’14), ACM Press, 2014, pp. 701-712.
70. F. H. Su, J. Bell, C. Murphy. Dynamic Inference of Likely Metamorphic Properties to Support Differential Testing. *Proceedings of the 10th International Workshop on Automation of Software Test* (AST’15), Co-located with the *37th IEEE International Conference on Software Engineering* (ICSE’15), IEEE Computer Society, 2015, pp. 55-59.
71. C. Sun, Y. Liu, Z. Wang, W.K. Chan. μMT: A Data Mutation Directed Metamorphic Relation Acquisition Methodology. *Proceeding of the First International Workshop on Metamorphic Testing* (MET 2016), collocated with ICSE 2016, IEEE Computer Society, 2016, pp.12-18.
72. A. Gotlieb, B. Botella. Automated Metamorphic Testing. *Proceedings of the 27th Annual International Conference on Computer Software and Applications* (COMPSAC’03), IEEE Computer Society, 2003, pp. 34-40.
73. H. Liu, X. Liu, T. Y. Chen. A New Method for Constructing Metamorphic Relations. *Proceedings of the 12th International Conference on Quality Software* (QSIC’12), IEEE Computer Society, 2013, pp. 59-68.
74. G. Batra, J. Sengupta. An Efficient Metamorphic Testing Technique Using Genetic Algorithm. *Proceedings of 5th International Conference on Information Intelligence, Systems, Technology and Management* (ICISTM’11), Springer, 2011, pp. 180-188.
75. G. Dong, T. Guo, P. Zhang. Security Assurance with Program Path Analysis And

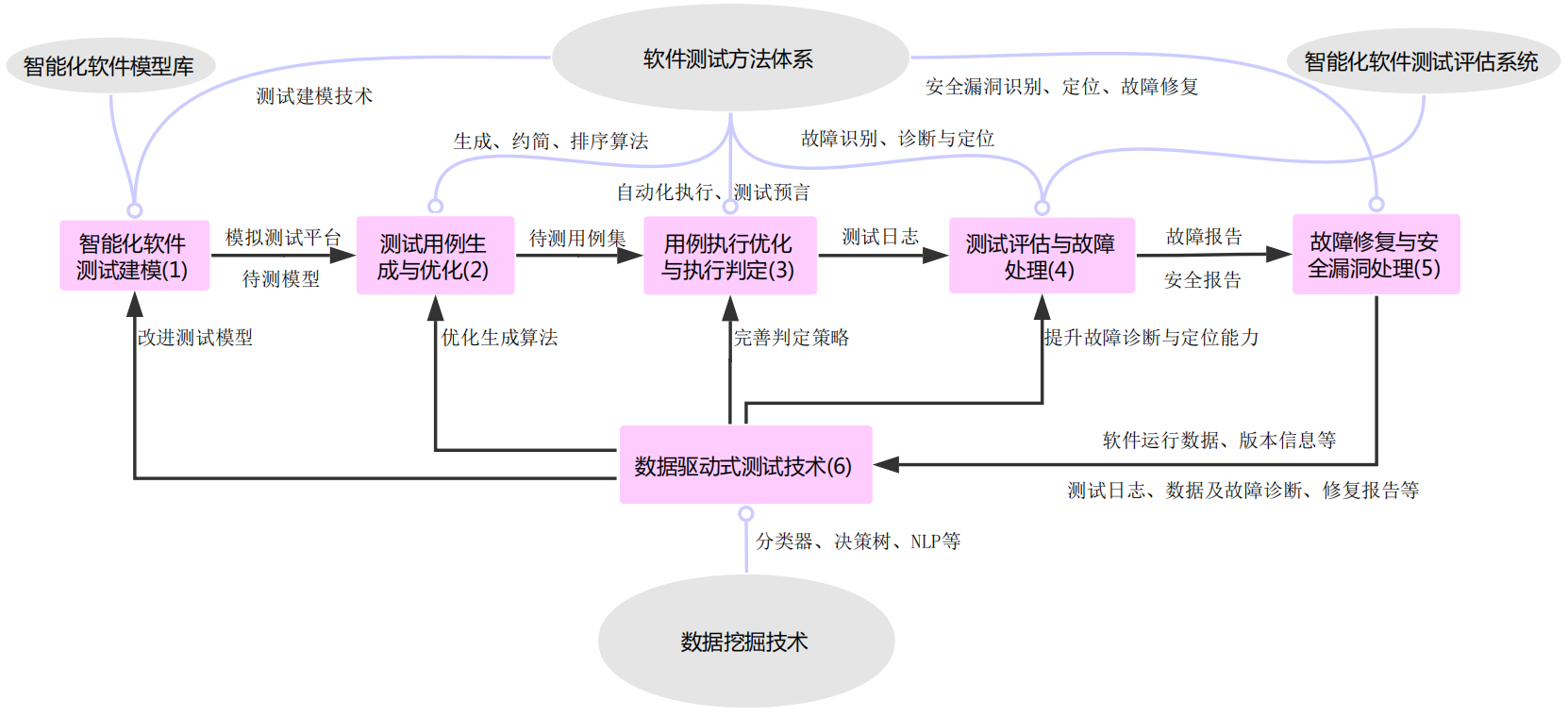
Metamorphic Testing. *Proceedings of the 4th International Conference on Software Engineering and Service Science* (ICSESS’13), IEEE Computer Society, 2013, pp. 193-197.

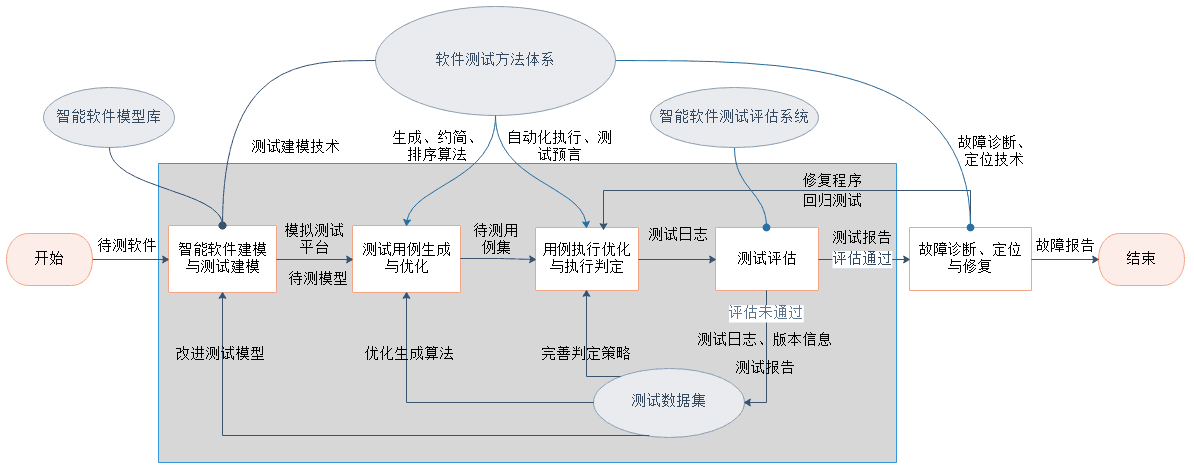
1. T. Y. Chen, F. C. Kuo, Y. Liu. Metamorphic Testing and Testing with Special Values. *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (SNPD’04), 2004, pp. 128-134.
2. X. Xie, W. E. Wong, T. Y. Chen, B. Xu. Metamorphic Slice: An Application in SpectrumBased Fault Localization. *Information and Software Technology*, 2013, 55(5): 866-879.
3. G. Dong, S. Wu, G. Wang, T. Guo, Y. Huang. Security Assurance with Metamorphic Testing and Genetic Algorithm. *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (WI-IAT’10), IEEE Computer Society, 2010, pp. 397-401.
4. C. Sun, G. Wang, B. Mu. Metamorphic Testing for Web Services: Framework and A Case Study. *Proceedings of the 9th IEEE International Conference on Web Services* (ICWS’11), IEEE Computer Society, 2011, pp. 283-290.
5. C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T. Y. Chen. A Metamorphic Relation-Based Approach to Testing Web Services without Oracles. *International Journal of Web Services Research*, 2012, 9(1): 51-73.
6. W. K. Chan, S. C. Cheung, K. R. Leung. Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications. *Proceedings of 5th International Conference on Quality Software* (QSIC’05), IEEE Computer Society, 2005, pp. 470-476.
7. W. K. Chan, S. C. Cheung, K. R. Leung. A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications. *International Journal of Web Services Research*, 2007, 4(2): 61-72.
8. J. Mayer, R. Guderlei. On Random Testing of Image Processing Applications. *Proceedings of the 6th International Conference on Quality Software* (QSIC’06), IEEE Computer Society, 2006, pp. 85-92.
9. F. C. Kuo, S. Liu, T. Y. Chen. Testing a Binary Space Partitioning Algorithm with Metamorphic Testing. *Proceedings of the ACM Symposium on Applied Computing* (SAC’11), ACM Press, 2011, pp. 1482-1489.
10. T. H. Tse, S. S. Yau. Testing Context-Sensitive Middleware-Based Software Applications. *Proceedings of the 28th Annual International Computer Software and Applications Conference* (COMPSAC’04). IEEE Computer Society, 2004, pp. 458-466.
11. W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse, Z. Zhang. Towards the Testing of PowerAware Software Applications for Wireless Sensor Networks. *Proceedings of the 12th AdaEurope International Conference on Reliable Software Technologies* (ICRST’07), Springer, 2007, pp. 84-99.
12. C. Sun, Z. Wang, G. Wang. A Property-based Testing Framework for Encryption Programs. *Frontiers of Computer Science*, Springer, 2014, 8(3): 478-489.
13. https://www.dmv.ca.gov/portal/wcm/connect/946b3502-c959-4e3b-b119 -91319c27788f/GoogleAutoWaymodisengagereport2016.pdf?MOD=AJPERES, Google Auto Waymo Disengagement Report for Autonomous Driving.
14. https://www.theatlantic.com/technology/archive/2017/08/insidewaymos-secret-testing-andsimulation-facilities/537648/, Inside Waymo’s Secret World for Training Self-Driving Cars.
15. I. H. Witten, E. Frank, M. A. Hall, and J. P. Christopher. Data Mining:Practical Machine Learning Tools and Techniques. *Journal of Management Science*, Ubon Ratchathani University, 2005, 3(6): 92-96.
16. http://www.cleverhans.io/security/privacy/ml/2017/06/14/verifcation.html, The Challenge of Verifcation and Testing of Machine Learning.
17. K. Pei, Y. Cao, J. Yang, S. Jana. Deepxplore: Automated Whitebox Testing of Deep Learning Systems. *Proceedings of the 26th Symposium on Operating Systems Principles* (SOSP’17), ACM Press, 2017, pp. 1-18.
18. N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami. The Limitations of Deep Learning in Adversarial Settings. *Proceedings of the 1st IEEE European Symposium on Security and Privacy* (EuroS&P’16), IEEE Computer Society, 2016, pp. 372-387.
19. O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi. Measuring Neural Net Robustness with Constraints. *Proceedings of the 10th Annual Conference on Neural Information Processing Systems* (NIPS’16), 2016, pp. 2613-2621.
20. N. Carlini, D. Wagner. Towards Evaluating the Robustness of Neural Networks. *Proceedings of the IEEE Symposium on Security and Privacy* (SP’17), IEEE Computer Society, 2017, pp. 39-57.
21. O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi. Measuring Neural Net Robustness with Constraints. *Proceedings of the International Conference on Machine Learning* (ML’17), IEEE Computer Society, 2017, pp. 2613-2621.
22. S. Gu, L. Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *Proceedings of the International Conference on Learning Representations* (ICLR’15), IEEE Computer Society, 2015, pp. 777-780.
23. X. Huang, M. Kwiatkowska, S. Wang, M. Wu. Safety Verification of Deep Neural Networks. *Proceedings of the International Conference on Computer Aided Verification* (CAV’17), Springer, 2017, pp. 3-29.
24. K.-Y. Cai. Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics [J]. *Information and Software Technology*, 2002, 44(14): 841–855.
25. P. E. Ammann and J. C. Knight. Data Diversity: an Approach to Software Fault Tolerance [J]. *IEEE Transactions on Computers*, 1988, 37(4): 418–425.
26. G. B. Finelli. NASA Software Failure Characterization Experiments [J]. *Reliability Engineering and System Safety*, 1991, 32(1): 155–169.
27. K.-Y. Cai, H. Hu, and F. Ye. Random Testing with Dynamically Updated Test Profile [C]. *Proceedings of the 20th International Symposium on Software Reliability Engineering* (ISSRE’09), IEEE Computer Society, 2009, pp. 198.
28. Y. Li, B. B. Yin, J. Lv, and K.-Y. Cai. Approach for Test Profile Optimization in Dynamic Random Testing [C]. *Proceedings of the 39th IEEE Annual International Computer Software and Applications Conference* (COMPSAC’15), IEEE Computer Society, 2015, pp. 466–471.
29. Z. Yang, B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu. Dynamic Random Testing with Parameter Adjustment [C]. *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe*, Co-located with the *38th IEEE Annual International Computer Software and Applications Conference* (COMPSAC’14), IEEE Computer Society, 2014, pp. 37–42.
30. J. Lv, H. Hu, and K.-Y. Cai. A Sufficient Condition for Parameters Estimation in Dynamic Random Testing [C]. *Proceedings of the 3rd IEEE International Workshop on Software Test Automationthe*, Co-located with the *35th IEEE Annual International Computer Software and Applications Conference* (COMPSAC’11), IEEE Computer Society, 2011, pp. 19–24.
31. Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. *IEEE Transactions on Software Engineering*, 2015, 41(5): 507-525.
32. Hernández-Orallo J. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement[J]. *Artificial Intelligence Review*, 2017, 48(3): 397-447.
33. Xue Han, Tingting Yu, and David Lo. 2018. PerfLearner: Learning from Bug Reports to Understand and Generate Performance Test Frames. In Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE ’18), September 3–7, 2018, Montpellier, France.ACM, New York, NY, USA, 12 pages.
34. Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska,and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE ’18), September 3–7, 2018, Montpellier, France.ACM, New York, NY, USA, 11 pages
35. Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Diﬀerential Fuzzing Testing of Deep Learning Systems. In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ’18), November 4–9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY, USA, 5 pages.
36. Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 1–18
37. Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE ’18), September 3–7, 2018, Montpellier, France. ACM, New York, NY, USA, 11 pages.
38. Afsoon Afzal. 2018. Quality Assurance Automation in Autonomous Systems.In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE’18), November 4–9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY,USA, 4 pages
39. https://doi.org/10.1007/s10664-018-9603-z
40. [117]Boyuan Chen, Jian Song, Peng Xu, Xing Hu, and Zhen Ming (Jack) Jiang.2018. An Automated Approach to Estimating Code Coverage Measures via Execution Logs. In Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE ’18), September 3–7,2018, Montpellier, France. ACM, New York, NY, USA, 12 pages.
41. Afsoon Afzal. 2018. Quality Assurance Automation in Autonomous Systems.In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE’18), November 4–9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY,USA, 4 pages.
42. Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest:Automated Testing of Deep-Neural-Network-driven Autonomous Cars. In ICSE ’18: ICSE ’18: 40th International Conference on Software Engineering, May 27-June 3, 2018, Gothenburg, Sweden. ACM, New York, NY, USA,12 pages.
43. Gharibi R, Rasekh A H, Sadreddini M H. Locating relevant source files for bug reports using textual analysis[C]//Computer Science and Software Engineering Conference (CSSE), 2017 International Symposium on. IEEE, 2017: 67-72.
44. Huo X , Li M , Zhou Z H . Learning unified features from natural and programming languages for locating buggy source code[C]// International Joint Conference on Artificial Intelligence. AAAI Press, 2016.
45. Huo X, Li M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code[C]//Proceedings of the 26th International Joint Conference on Artificial Intelligence. AAAI Press, 2017: 1909-1915.
46. Lam A N , Nguyen A T , Nguyen H A , et al. Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N)[C]// IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2016.
47. Wagner D A. Static analysis and computer security: New techniques for software assurance[M]. University of California, Berkeley, 2000.
48. Kong D, Zheng Q, Chen C, et al. ISA: a source code static vulnerability detection system based on data fusion[C]//Proceedings of the 2nd international conference on Scalable information systems. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007: 55.
49. Wu Z, Atwood J W, Zhu X. A new fuzzing technique for software vulnerability mining[C]//International Conference on Software Engineering. 2009.
50. Holler C, Herzig K, Zeller A. Fuzzing with Code Fragments[C]//USENIX Security Symposium. 2012: 445-458.
51. Tian W, Yang J F, Xu J, et al. Attack model based penetration test for SQL injection vulnerability[C]//Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual. IEEE, 2012: 589-594.
52. Yamaguchi F, Lindner F, Rieck K. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning[C]//Proceedings of the 5th USENIX conference on Offensive technologies. USENIX Association, 2011: 13-13.
53. Pang Y, Xue X, Namin A S. Predicting vulnerable software components through n-gram analysis and statistical feature selection[C]//Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on. IEEE, 2015: 543-548.
54. Li Z, Zou D, Xu S, et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection[J]. arXiv preprint arXiv:1801.01681, 2018.
55. Li Z, Zou D, Xu S, et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities[J]. arXiv preprint arXiv:1807.06756, 2018.
56. Afsoon Afzal，Quality assurance automation in autonomous systems .October 2018 ESEC/FSE 2018: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering
57. Andreas Zeller. 1999. Yesterday, my program worked. Today, it does not. Why?.In European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE). 253–267

## **2．项目的研究内容、研究目标，以及拟解决的关键科学问题（此部分为重点阐述内容）；**

在智能化软件测试与软件测试方法理论研究成果基础上，我们拟围绕智能化软件系统测试流程中关键步骤进行研究，探索智能化软件系统的测试框架与优化技术。测试流程主要包括**软件测试建模(1)**、**测试用例生成与优化(2)**、**测试执行优化与结果判定(3)**、**测试评估与故障注入(4)、安全漏洞识别与故障定位(5)及数据驱动式测试技术(6)**（图2-1所示）**。**

为了尽可能提升测试的科学性，我们拟采用**测试产生数据，数据优化测试**的测试理念，数据驱动式测试技术旨在通过软件运行数据及测试数据的研究学习，利用数据挖掘技术，对测试建模、测试生成、测试执行及测试评估等步骤进行改进，开启新一轮测试，逐步提升测试充分性，主要研究内容如下：





**图2-1 智能化软件系统测试流程**

### 2.1 主要研究内容

#### 2.1.1智能化软件的测试模型建立

智能化软件往往功能庞大，模块之间交互频繁，系统复杂性较高，很多测试任务甚至难以描述，测试内容过于模糊，这些都给智能化软件的充分性测试带来诸多难题，我们拟研究基于场景的智能化软件模型建立方法，明确细致的描述测试任务，分析智能化软件系统应用场景及任务中与软件质量相关的特征，通过这些特征分析智能化软件系统在运行时所应满足的特性， 针对不同类型的智能化软件系统，分析软件系统在主要应用场景下的真实历史运行数据，细化应用场景下的事物、交互对象、交互方式，并以此为基础**构建智能化软件系统在各应用场景下的测试模型**。而构造适合的测试模型，具体研究内容如下：

**（1）智能化软件模型库研究：**智能化软件广泛应用于医疗、教育、科技、生活等诸多方面，表现形式是多样的，我们需要研究主流智能化软件的显著特征，丰富智能化软件模型库，对具有显著特征的智能化软件或软件某一模块研究具体的测试方略。

**（2）智能化软件待测任务描述：**智能化软件系统功能庞大，充分性测试需要深入浅出，由表及里，既要对智能化软件的诸多功能进行完备测试，也要对功能与功能的交互进行充分测试，所以我们需要研究如何对智能化软件待测功能进行细分，对待测任务进行明确描述。

**（3）智能化软件功能模块整合：**智能化软件系统模块众多，交互频繁，不同模块都有各异的输入域与输出域，需要我们细致研究诸多模块之间复杂的交互方式及协作关系，对智能化软件的模块构造有细致了解，进而对每个待测任务执行方式、测试环境和测试对象和测试目标有系统认识。

**（4）智能化软件模拟平台搭建：**针对真实环境下的智能化软件系统测试存在高昂且难以负担的成本的问题，我们需要研究如何科学搭建智能化软件系统的模拟测试平台，即以智能化软件系统的应用场景模型为指导，分析软件系统在主要应用场景下的真实历史运行数据，细化应用场景下的事物、交互对象、交互方式，并以此为基础构建智能化软件系统在各应用场景下的模型，模拟系统实际应用环境下的各种事物及对象，较为真实地构建智能化软件系统的实际应用环境，为智能化软件系统的测试奠定基础。

**（5）软件建模技术研究：**测试对象明确和需求确定之后，我们需要研究如何借鉴软件测试方法体系中测试建模技术，结合智能化软件特征和模型特点建立合适的智能化软件测试模型，故而需要对传统软件建模的相关技术进行详细总结和研究，归纳出适合于本课题的测试建模技术。

#### 2.1.2测试用例的生成与优化

智能化软件测试的主要步骤包括测试用例生成、测试执行与测试结果判定。测试用例生成是智能化软件测试领域至关重要的研究问题。智能化软件系统要求具有类似人类的智能行为，且大多数智能化软件需要和外界进行频繁的交互，导致智能化软件系统的输入域异常庞大、系统复杂性较高。显然，穷尽测试无法实现。生成有效的、多方位测试智能化软件系统的测试用例是一项具有挑战性的工作。由于智能化软件系统的执行逻辑复杂，测试结果判定更加困难，在某些情况下甚至无法确定测试预期。现有的智能化软件测试技术利用随机或人工的方式生成测试用例并且主要依靠测试人员验证系统行为，导致测试效率可能不高。另一方面，现有的智能化软件测试工作忽略了测试过程对测试效率的影响。在测试资源有限的情况下，利用数据驱动的方式（基于测试的历史信息），控制测试过程，提高测试效率。

针对现有智能化软件系统测试用例生成技术的诸多局限性，考虑智能化软件系统的特点，依据现有测试用例生成技术与相关理论，研究新型数据驱动的智能化软件系统的测试用例生成技术。在此基础上，研究测试用例的优化技术，包括：测试用例生成与选择，测试用例执行与测试用例优先级排序等。需要研究以下内容：

**（1）在测试用例生成与选择方面：**智能化软件系统输入参数的多源性导致输入空间巨大。增加测试用例集的多样性有助于更有效地检测智能化软件系统的异常行为。为了实现测试用例集多样性，探索：（a）满足某种覆盖准则的测试用例生成方法；（b）结合场景与功能的测试用例生成技术，将场景测试数据与功能测试数据相结合来实现智能化软件系统自顶向下的测试；（c）基于参数组合优化的测试生成技术，实现减少测试用例数目的同时覆盖尽可能多的组合。（d）生成的测试用例需要与后续的执行判定结合起来，我们需要尽可能生 成一系列易于判定或者为可判定的测试用例。

**（2）在测试用例优化排序方面：**测试用例集很多情况下存在着较大的优化空间，我们可以通过真实场景分析及执行结果预测等方式删减一批冗余或者无效测试用例，在不影响测试效果的情况下，简化测试用例集，降低测试成本。另一方面，通过改变测试用例的执行顺序以提早执行到揭示故障的测试用例。为此，即将执行的测试用例应尽可能与已经执行的测试用例不同。通过测试用例距离度量的方式选择最不同的测试用例，并以此为基础提出基于距离的智能化软件系统测试用例优先级排序技术等。

#### 2.1.3用例的执行优化与执行判定

测试用例的执行优化主要体现在用例执行操作采用自动化进行，得益于模拟测试平台的搭建，我们旨在研究面向智能化软件系统的自动化测试执行工具，减轻测试人员工作负担。其次，智能化软件系统的测试执行结果主要通过测试人员主观判断系统行为是否正确，这种方式不仅需要大量的测试资源，还易出现误判的情况（将错误的行为判断为正确的行为）。在传统软件测试过程中，测试人员主观判断的方式尽管也存在一定误判情况，但是总的来说是非常有效的测试结果判定方式，但是对于智能化软件而言，测试执行之后，可能系统输出繁杂，或者输出加密数据，或者根本没有显著结果，这都给测试人员主观判断带来极大的阻碍。

因此，我们提出自动化验证系统行为的方法，可以极大提高测试效率与准确度。旨在研究以下内容：

**（1）提出一种自动或者直观的方式判断系统行为是否正确：** 人工验证系统行为的方式不仅需要占用大量的测试资源而且不能保证判断的准确度。因此，提出一种自动地验证系统行为的方式是有必要的。传统软件测试中缓解测试预期问题的技术有很多（例如：N版本、断言等），其中一些技术运用到智能化软件测试效果可能不好。例如，断言技术需要在代码中插入一些可以获取执行信息的代码，但很多智能化软件系统的执行逻辑不是由程序控制的。考虑将N版本以及交叉验证的方式作为验证系统行为的机制。

**（2）基于蜕变测试的智能化软件系统的测试结果验证技术：**蜕变测试是一种无需测试预期的软件测试技术，在待测软件的测试预期不存在的情况下也能对其进行有效测试。该测试技术通过判断待测软件的多个测试用例之间是否满足一些必要的属性来测试程序。这些必要的属性被称为蜕变关系，隐含于待测软件的功能规格说明中，是验证测试结果及判断待测软件是否满足特定的功能需求。研究基于蜕变测试的智能化软件系统测试结果验证技术，需要研究如下内容：**（a）面向智能化软件系统的蜕变关系识别方法：**智能化软件系统中隐含的蜕变关系是验证测试结果的关键。如何从智能化软件系统中识别蜕变关系是一个重要的问题。研究基于范畴划分方法的和基于数据变异的智能化软件系统蜕变关系识别方法；**（b）基于蜕变测试的智能化软件系统测试结果验证实现机理：** 通过判断蜕变关系所涉及测试用例的输出结果之间是否满足蜕变关系，实现智能化软件系统测试结果的验证，判断待测软件是否满足功能需求。

#### 2.1.4测试评估与故障处理

测试用例执行之后，会生成详尽的测试报告，我们的研究重心转变为对测试报告的研究分析，我们需要利用传统软件测试方法体系中测试评估技术，对测试报告进行有效评估，评估测试用例集是否达到充分完全的覆盖，评估测试用例执行判定是否科学，评估是否还需要额外测试集加强测试充分性，评估未通过的测试用例集中是否包含隐藏故障等，生成完备的测试报告与安全报告。

智能化软件系统获取决策逻辑的方式使得如何评估被测对象的测试充分性以及测试用例集的充分程度是一个重要问题。传统软件测试中的测试充分性指标（语句覆盖、分支覆盖和MC/DC覆盖等）不能完全适用于智能化软件测试。原因是，在智能化软件测试中，不仅要考虑数据处理模块、决策模块等对应的代码是否正确实现，还需要考虑影响决策逻辑的内部因素（神经元、激活函数和阈值等）。另一方面，由于智能化软件获取逻辑的方式，传统软件测试中评估测试用例集充分性的技术（变异分析和各种覆盖准则）也不能完全适用于智能化软件系统。

测试评估之后，需要对测试过程中暴露出的问题进行具体分析，即故障处理，主要包含故障识别、故障诊断、故障定位等阶段，对于测试集中不符合预期效果的部分用例，首先需要识别是否有内部故障存在，还是由测试环境异常导致的，如果是前者，我们需要进行故障诊断，分析故障可能的存在形式，故障的原因，故障触发的条件和故障可能导致的后果，之后需要进行故障定位，对于故障可能存在的模块点进行尽可能的精确定位，其次我们也需要研究故障可能的分布点有哪些，其他模块是否还隐藏类似的故障。故障处理阶段研究具体内容如下：

**（1）智能化软件系统的故障特点分析：**智能化软件系统的故障集中出现在：数据预处理和决策模块的代码实现部分以及决策模块的逻辑部分。实现数据处理和决策模块的程序语言不同，相应地，故障类型与特点也不同。另外，决策模块的逻辑是从数据中习得且呈现出异常复杂的特点，与传统代码故障差异较大。因此，与决策模块逻辑相关的故障难以理解。

**（2）智能化软件系统的故障检测机制：**测试用例生成是智能化软件系统测试的关键步骤。在测试过程中，为了增加发现系统行为异常的概率，生成的测试用例应当考虑系统的内部结构。智能化软件系统的故障检测不仅要关注测试用例的生成方式，还要考虑如何尽可能地覆盖不同的执行逻辑。

**（3）智能化软件系统的故障定位机制：**智能化软件系统中的故障如果被测试用例有效触发之后，我们要考虑如何用较少的额外测试或者是通过历史测试数据分析，定位出故障发生的模块及功能，尽可能重现故障，掌握故障发生的场景及环境。

#### 2.1.5安全漏洞识别与故障定位

智能化软件系统的测试不仅仅是为了发现隐藏在软件系统内部的故障，还要能够辅助系统维护人员快速识别出软件中存在的安全漏洞，准确定位到相应的源文件，以便调试和及时修复这些缺陷，然后对修复后的代码再进行回归测试，以避免缺陷给智能软件等安全关键系统带来严重的安全隐患，因此安全漏洞识别和定位对于软件测试的后期维护具有重要的指导意义。

然而，在软件测试后给出的缺陷报告中，准确识别出哪些缺陷是安全相关的缺陷并不是一件容易的事情，由于对信息安全知识的缺乏，报告提交者往往无法正确地区分安全漏洞与常规的软件缺陷，导致许多安全相关的缺陷被错误地标记为与安全无关的缺陷。一方面，从大量的安全无关缺陷中把误标记的安全相关缺陷召回不仅需要很强的专业领域知识，而且这项工作极其费时费力。另一方面，即使召回了这些安全相关的缺陷，也需开发和维护人员审查大量的源文件，利用其领域知识来定位错误的源代码文件。人工定位故障的过程非常耗时。因此，如何从海量的缺陷报告中快速而准确地识别出与安全漏洞相关的缺陷报告（以下简称“安全缺陷报告”）并辅助开发人员快速将注意力集中在与给定缺陷相关的潜在错误源代码文件上是一个亟待解决的难点科学问题。

基于以上分析，我们拟研究基于安全缺陷报告的安全漏洞识别与定位方法，通过对安全缺陷报告识别、缺陷报告定位到源代码等研究，提高软件维护效率，并保障软件安全。具体研究内容如下：

**（1）安全缺陷报告识别方法：**将排序学习和词嵌入模型相结合识别安全缺陷报告。在训练模型前对训练数据集按内容相关性进行去噪，然后将去噪后的训练数据集中的缺陷报告转化为分布式向量表示，用以训练预测模型，最后利用预测模型识别新的缺陷报告是否为安全缺陷报告。具体包括：训练数据集的去噪方法研究，缺陷报告的分布式向量表示方法研究，基于机器学习的预测模型研究。

**（2）基于缺陷报告的故障定位方法：**结合深度学习和信息检索技术，研究缺陷报告和源码的特征提取方法和特征表示方法以及缺陷报告和源码的相似度计算方法，进而建立基于缺陷报告的故障定位模型，实现从缺陷报告到源文件的定位，优先对定位出的具有高可疑度的源文件进行修复。在对故障修复之后，再进行回归测试。首先，结合程序分析和自然语言处理技术从源码和缺陷报告中提取相关特征，然后基于深度学习和信息检索技术计算源码和缺陷报告之间的语义和词法相似度，根据相似度给出疑似包含缺陷的源文件的排序列表。具体研究内容：源码的结构信息和文本信息提取研究，缺陷报告的文本信息和特征属性的提取研究，基于深度学习的源码和缺陷报告的分布式表示研究，语义相似度和词法相似度的计算研究，以及基于排序学习的可疑源码文件推荐方法研究。

，

2.1.6数据驱动式测试技术

随着互联网软件服务的普及，软件数据的种类更加丰富，规模也越来越大。软件生命周期中会产生大量的、各种类型的数据，例如开发过程中的源代码、需求文档、缺陷报告、测试用例；系统运行中的运行日志、性能度量、事件记录；用户交互中使用行为序列、调查问卷、社交媒体反馈，软件版本信息、更新日志等等。其次，在智能化软件测试过程中，也会产生很多数据，例如测试日志、测试数据、测试用例集及执行结果、故障报告、安全报告、故障诊断与修复日志等等。如何利用这一类数据，丰富数据驱动式测试技术，完善智能化软件测试框架，提升智能化软件测试能力。具体研究内容如下：

**（1）数据筛选：**大数据的五个基本特征为大量、高速、多样、价值、真实，对于多样且庞大的数据集，我们需要研究如何甄别高价值数据，过滤无效数据或噪音数据，我们拟构建数据价值密度指标，对海量数据的价值密度及可挖掘程度进行细致刻画，继而研究如何高效预处理，使得多样的原始数据处理成可利用的中间数据。

**（2）基于数据挖掘技术的数据处理：**我们拟利用数据挖掘技术，提取数据集智能蕴含的数据价值，构建合适的数据模型并应用适合的挖掘技术，例如采用NLP技术对文本信息进行提取，采用分类预测算法对智能化软件中测试报告的结果判定进行整理。

**（3）优化测试框架及改进测试技术：**直观而言，软件数据及测试数据可以帮助软件测试人员重现缺陷，识别缺陷以及定位缺陷。甚至可以对软件质量及可靠性做出有效评估。深层次而言，基于这些海量数据，软件从业者可以提取出关于软件质量和开发动态的重要信息。再依靠这些信息，指导我们进行高效测试，制定数据驱动式的测试框架，从而达到对智能化软件系统的高效测试。具体包括**(a)改进测试模型：**智能化软件的测试建模不是一成不变的，第一次建模也不是尽善尽美的，我们需要根据对测试不足的分析，动态修改测试模型的建立，进一步完善测试模型，**(b)优化生成算法：**测试用例的生成基于测试模型和软件测试方法体系，在智能化软件测试中，生成测试用例也需要灵活的根据测试场景及对象的差异而动态调整，数据驱动式测试旨在进一步优化生成算法，在效率上增强生成算法的竞争力。**(c)完善判定策略：**测试预言问题是测试执行的关键，而判定策略更多的基于对待测软件的系统研究而得出，例如蜕变关系的提取等，但判定策略的完善也不是一蹴而就的，需要根据测试数据不断优化，很多之前难以判定的测试场景可以在之后的测试中加以精确判定。**(d)提升故障诊断与定位能力：**因为智能化软件可靠性与冗余性较高，故智能化软件故障隐蔽性较高，而算法的故障诊断能力也随着测试的不断进行，数据的不断挖掘会不断提升，故障定位能力可能也会由可疑模块定位转向精确模块定位，甚至提升到代码定位等。

### 2.2 研究目标

针对上述诸多研究内容及对应的具体问题，围绕智能化软件系统测试中**智能化软件测试建模**、**测试用例生成与优化**、**测试执行优化与判定**、**测试评估与故障处理、故障修复与安全漏洞处理及数据驱动式测试**等关键问题开展研究，拟达到以下研究目标：

#### 2.2.1建立完备的智能化软件测试建模理论体系

我们拟对现有智能化软件系统进行调查研究，智能化软件非常庞大，种类众多，我们不可能开发出针对每一款智能化软件的测试工具，我们希望能够分析智能化软件系统的特点，对智能化软件系统的架构方式与开发运行方式进行有效总结，对多样的智能化软件进行科学分类，对智能化软件待测任务进行科学化描述，建立一个科学的智能化软件模型库，可以对典型的智能化软件或其中的某些模块进行模型抽象，继而进行充分性测试。具体目标如下：

**（1）建立一个完备的科学的智能化软件模型库：**该模型库可对当前典型智能化软件的进行特征提取，对于典型的智能化软件及其功能模块有对应的建模思想及模板，在建模科学性上有详细证明。

**（2）构建科学的智能化软件待测任务描述方式：**可根据某一场景或任务下的智能化软件典型特征，研究如何分析智能化软件系统为适应或满足这一特征而必须满足的运行时特性，达到对智能化软件系统的待测任务有系统、准确的描述。

**（3）归纳出智能化软件系统功能模块交互整合方法：**该方法可指导测试人员对待测软件的功能模块交互进行有效区分，对每个模块或功能的输入及输出有详见归纳，并依此绘制模块交互图以刻画待测软件内部结构。

**（4）搭建一个真实可靠的测试平台：**模拟测试平台是基于真实智能化软件搭建，必须保证真实可靠，抽象模型可应用在测试平台之上，对于诸多待测功能与模块，在模拟测试平台上都有对应的近似方法以保证真实性与可靠性，继而为开展模拟测试奠定基础。

**（5）提出智能化软件建模方案集：**从传统的软件测试方法体系中提取适合本课题的建模理论与技术，总结现有技术，并提出一系列软件建模方案，在后续的测试开展中加以比较和修改，最后进行综合。

#### 2.2.2挖掘高效的测试用例生成算法及优化排序算法

我们拟通过分析智能化软件测试模型的特征，依据现有测试用例生成技术与相关理论，研究新型数据驱动的智能化软件系统的测试用例生成技术及优化技术，具体目标如下：

**（1）提出一套面向智能化软件测试的测试用例生成算法：**该用例集可以对智能化软件中进行多角度，全方位进行充分性测试，满足场景与功能相结合技术，而且测试用例的执行结果也易于后续操作进行有效判定。

**（2）提出一套测试用例集优化算法理论：**该理论可以通过真实场景分析及执行结果预测等方式删减一批冗余或者无效测试用例，在不影响测试效果的情况下，简化测试用例集，降低测试成本。另外，可以提供一系列排序方案，例如基于故障覆盖的测试排序和基于最小切换成本的测试排序等方式。

#### 2.2.3开发自动化的测试用例执行工具及建立执行结果判定方法体系

为了提升测试执行的效率，我们拟开发出一套自动化的测试用例执行工具，代替人工执行，在模拟测试平台上批量高速执行测试用例，测试执行自动化是智能测试的发展趋势，另外，针对智能化软件系统执行结果判定困难的问题，我们拟建立执行结果判定方法体系，例如，寻找待测软件的蜕变关系，智能化对执行结果进行有效判定。智能化的结果判定是一项复杂的工作，我们会根据实际情况，尽可能的对执行结果进行有效判定，对于缺乏蜕变关系， 或者是蜕变关系不明确的情况，可能采用机器学习的方式，对执行结果进行预判定或者部分判定，继而使用人力或者其他方式进行逐一分析。具体目标如下：

**（1）归纳出可自动或者直观判断系统行为是否正确的场景或条件：** 该目标可以通过对一批易于直观判断执行结果的软件或场景进行归纳，自动判断或直观判断并不是一定不适用于智能化软件的执行判定的，相反在某些场景或某些条件下，直观判断或自动判断却是最有效的方法，我们需要归纳出适合于此的场景或条件。

**（2）建立基于蜕变测试的智能化软件系统的测试结果验证技术体系：** 该体系可以指导测试人员快速识别出隐藏在智能化软件系统内部的蜕变关系，另外可以提高测试人员在蜕变测试结果验证方面的工作效率。

#### 2.2.4提出科学的智能化软件测试的评估指标及故障处理流程

我们拟提出科学的智能化软件评估指标，包括测试充分性与测试集充分性的两种评估，智能化软件与传统软件存在诸多不同，意味着传统软件的评估体系往往并不适用于智能化软件的评估。我们拟从科学性角度提出一种新的面向智能化软件测试的评估指标，并给出其合理性、科学性与适用性说明，并应用其对我们的测试方法进行有效评估，对于评估结果进行合理分析，并通过合理比较的方式，提出我们数据驱动式测试方法的长处与不足。主要目标如下：

**（1）提出测试充分性的评估指标：**传统软件的测试充分性度量指标可以评估智能化软件中代码部分的测试充分性，不能对决策模型进行有效的评估。为此，借鉴传统软件测试充分性评估指标的提出思路，并结合决策模型的特点，提出适用于智能化软件的测试充分性指标。

**（2）提出测试用例集充分性的评估指标：**变异分析和覆盖准则可以评估传统软件测试用例集的充分性，但由于缺少针对智能化软件的变异算子，变异分析技术不能直接运用到智能化软件测试中。为了解决这一问题，分析以往的测试历史，提取故障模式，总结面向智能化软件的变异算子。

对于测试中暴露出的问题，我们拟提出一套科学的故障处理流程，利用传统软件测试方法中故障定位的相关研究来指导智能化软件故障定位的工作，利用较少的额外的测试集和历史测试结果分析，以及对测试模型的研究来尽可能的准确定位出故障的可能原因，故障的可能的触发方式，可能存在的模块位置，可采取的对应解决方案等，具体目标如下：

**（1）建立智能化软件系统的历史故障库：**该库对业界智能化软件系统的故障出现位置，时间，存在形式，故障类型与特点，修复手段等内容进行详细记录，通过对历史数据的分析可对当前软件系统测试进行良好的指导意义。

**（2）构建智能化软件系统的故障检测方法体系：**该体系包含一系列针对智能化软件系统测试报告的故障检测方法，每一种方法在特定场景下可检测出隐藏在测试中的软件故障，并对这些方法加以整合，构建一个丰富的故障检测方法体系。

**（3）提出智能化软件系统的故障定位算法：**该算法可对测试报告中识别检测出的软件故障进行功能定位或模块定位，尽可能重现故障，掌握故障发生的场景及环境。

#### 2.2.5制定基于测试缺陷报告的安全漏洞识别与故障定位策略

研究安全缺陷报告有助于我们及时发现软件安全漏洞，我们拟制定基于测试缺陷报告的安全漏洞识别和定位策略，提高安全缺陷报告识别的召回率，在软件测试的后期维护中，指导开发和维护人员消除安全漏洞，提高智能软件的质量和安全性。具体目标如下：

**（1）安全缺陷报告识别的目标：**能够在大量缺陷报告中有效识别安全相关的缺陷报告。

**（2）故障定位的预期目标：**建立基于缺陷报告的故障定位模型，辅助开发人员准确定位和修复缺陷。

#### 2.2.6建立数据驱动式的回归测试理论体系

我们拟建立一套完整的数据驱动式测试理论体系，对于软件生命周期中产生的数据和软件测试过程中产生的一系列数据，都作为数据驱动式测试方法的输入，进而建立集数据采集、数据筛选、数据处理、数据存储、数据挖掘、数据预测等诸多方法的理论体系，提取出与软件质量及测试相关的内容进而丰富数据驱动式测试技术，完善智能化软件测试框架，提升智能化软件测试能力。具体研究内容如下：

**（1）建立高效的数据预处理流程：**我们拟对软件生命周期及测试过程产生的数据进行采集与筛选，构建数据价值密度指标，对海量数据的价值密度及可挖掘程度进行细致刻画，剔除无效扰动信息，保留高价值数据，加以存储，并将不规则的文本信息规范化成可直接利用的数据信息。

**（2）提出基于数据挖掘技术的数据处理技术：**我们拟利用数据挖掘技术，提取数据集智能蕴含的数据价值，构建合适的数据模型并应用适合的挖掘技术，对于每一类数据类型，与对应的数据挖掘技术，产生对应的数据提取结果。

**（3）建立回归测试体系：**利用数据挖掘技术获取一定的数据信息之后，进而指导我们进行高效的回归测试，制定数据驱动式的回归测试框架，对之前的测试阶段进行重新改进与优化，从而达到对智能化软件系统的高效测试。具体包括测试模型的改进、生成算法的优化、判定策略的完善和故障诊断与定位能力提升。

### 2.3 拟解决的关键科学问题

#### 2.3.1智能化软件测试建模问题

首先是智能化软件本身建模问题，智能化软件与普通软件存在显著差异，真实场景运行成本较高，如何对智能化软件进行有效建模，功能模块抽象近似化处理，继而将模型运行在模拟平台，这样可以为测试奠定基础。

其次是测试模型的建立问题，智能化软件种类复杂，数量众多，功能庞大，对智能化软件应用合适的测试方法与测试技术取决于测试模型建立的优劣，测试建模有很多种，如何寻找最贴近智能化软件本身的建模方式直接决定测试的准确性与完备性。如何建立完备的智能化软件模型库，测试人员可以通过分析智能化软件的特征灵活选取合适的测试模型是很有价值的问题。另外，测试建模过程中，需要考虑待测任务的科学描述问题，待测任务的科学描述直接影响测试执行判定的效率，所以如何构建科学的描述方式，精确的刻画待测任务是测试建模中重要的子问题。

#### 2.3.2传统软件的测试用例生成与优化应用在智能化软件测试的问题

测试模型的建立之后，我们如何根据适当的测试生成算法，结合待测模型的特征生成一批有代表性的测试用例是当务之急，传统的软件测试在测试用例生成与优化上研究很多，但是在智能化软件上的应用却很少，智能化软件输入域庞大，功能复杂，测试用例易于受诸多约束影响，故而传统软件的测试生成与智能化软件如何结合是一个难题。

具体来说，测试用例集需要具有代表性、充分性、科学性、高效性、可执行性和可判定性，我们的测试内容都是与智能化软件直接相关的，测试对象保证不遗漏，测试方式科学可信，测试顺序高效化，测试用例真实可行且易于判定，如何提出达到这些要求的测试用例生成方案是我们研究的关键科学问题，其次对于现有用例集的约简排序，如何通过排序优先测试更有效的测试用例？如何通过约简删除重复用例或低效用例？

#### 2.3.3传统软件的测试预言及判定应用在智能化软件测试的问题

测试预言及判定一直是传统软件测试中一个很难解决的问题，目前主要的测试方法中蜕变测试可以在测试预言及判定上达到较好的效果，然而智能化软件的测试预言及判定却复杂的多，普通的测试执行很难触发软件崩溃或异常等问题，而会引发数据上的微小变化与不合理，其蜕变关系很难明确，约束条件比较模糊，随机性、灵活性较大，测试预言及判定与智能化软件测试如何完美契合是非常有意义的问题。

其次，测试判定策略能否动态调整也是一个难题，面对智能化软件，我们很多都是未知的，如何根据测试情况的变化动态调整判定策略，甚至基于判定策略的修改调整测试用例的生成与排序都是我们需要解决的科学问题。

#### 2.3.4传统软件的测试评估及故障处理应用在智能化软件测试的问题

智能化软件测试的目标同样是充分测试，但是与普通软件不同的是，智能化软件测试评价系统缺失，软件质量评价指标缺失，我们需要考虑如何建立科学的测试评价系统与软件质量评价指标，对测试情况进行科学的论述，回答待测软件是否已测试充分，测试不足在哪儿等问题。其次在测试评价较低时，考虑如何采用回归测试的问题，即如何修改测试模型，如何动态调整测试策略继而改进测试结果等。

对于智能化软件测试暴露出的问题，如何科学地识别故障，保证不遗漏，不误判，如何高效地诊断故障，尽可能快的找出故障存在的方式、类型、原因及触发形式等，如何精确地定位故障，保证故障定位效率高，准确度高等问题都是智能化软件测试过程中需要解决的新问题。

#### 2.3.5智能化软件的安全漏洞识别与故障定位问题

已有的关于软件安全漏洞的研究主要针对传统软件，不适合应用于智能软件系统。当今软件仓库和群体智能的发展，可以获得大量的缺陷报告及其修复的历史信息。如何有效利用软件历史仓库中的大量缺陷报告信息以及测试报告，自动识别可能危及智能化软件系统安全的缺陷是一个值得研究的科学问题。重点研究基于文本挖掘的安全缺陷报告识别模型面临的类不平衡、数据噪音和数据稀疏等难点问题。

此外，缺陷报告文本描述的多样性和缺陷描述信息的不完整性，尤其是缺陷报告中的自然语言描述与软件源码中的技术术语之间的语义失配和词汇鸿沟问题，无疑都加大了从报告自动定位到代码源文件的难度。因此，**如何**有效利用软件历史仓库中的大量缺陷报告信息**将可能危及智能化软件系统安全的缺陷准确定位到源文件辅助开发者修复这些缺陷是另一个值得研究的科学问题。**

#### 2.3.6回归测试与数据驱动式测试结合问题

在经过测试建模、测试用例生成与执行、测试评估与故障处理等步骤之后，很难保证测试充分性，故而将回归测试加入算法框架，对于算法执行过程中产生的软件数据和测试数据，需要考虑如何利用数据挖掘算法，怎样对数据进行充分利用，怎样构建数据驱动式测试框架，数据驱动式框架对测试算法的每一步能产生何种影响，怎样产生改进和修正作用等问题，例如，如何利用数据驱动改进测试模型，优化生成算法，完善判定策略及提升故障诊断与定位能力，都是值得研究的科学难题。

## **拟采取的研究方案及可行性分析**（包括研究方法、技术路线、实验手段、关键技术等说明）；

#### 3.1 广泛调研，科学建模

首先需要将智能化软件建模与智能化软件测试建模结合起来。智能化软件建模这一块，我们拟对当前智能化软件系统进行详细调查研究，加深对智能化软件系统的组织架构、功能特点的理解，构造一个智能化软件系统模型库，并可以根据智能化软件系统特征对其进行相应分类，对不同类别的智能化软件系统构造模型描述，基于软件模型搭建模拟测试平台，测试平台具有高仿真性与高可测性，能够支持一类智能化软件模型的模拟运行。智能化软件测试建模方面，我们在编写的教材《软件测试的概念与方法》中为每种方法整理了一些信息，将在此基础上，针对每种方法继续广泛收集资料，充分了解相关工作，认真总结出六个方面的要素，即概念、理论、方法、研究现状，优缺点和文献资料。并给出相应方法的支持工具和实例研究。 所以拟借鉴传统软件测试技术中经典建模方法，分析智能化软件系统特征，明确数据驱动式测试目标，建立科学的智能化软件测试模型，另一方面，我们拟调研现有的智能化软件的测试建模技术，分析不足及优点，加以借鉴历史智能化软件的测试建模的经验与教训，建立科学的测试模型依赖于对传统软件测试建模技术、智能化软件模型库和历史智能化软件测试建模案例三方面研究加以综合。

#### 3.2 技术整合，优化改进

传统软件测试方法中，**测试用例生成问题**是研究重点，我们在组合测试、蜕变测试等领域的测试用例生成研究上已经做了相当多的功课。对于智能化软件测试，传统的测试生成算法难以完全适用，我们拟首先明确智能化软件测试目标，进而明确测试用例覆盖评价指标，例如传统软件测试中的分支覆盖、路径覆盖、组合覆盖甚至完全覆盖等，我们需要对智能化软件系统的测试目标进行限定，例如功能覆盖、功能2维组合覆盖等。其次，我们考虑测试用例的生成算法，在达到一定覆盖率的基础上，尽可能约简测试用例规模，这方面的我们的研究比较丰富，我们编写的教材《组合测试的理论与方法》一书中，介绍了一系列基于组合覆盖的测试用例生成算法，例如贪心算法包括AETG、IPO等及启发式算法包括蚁群算法、遗传算法等，可以针对不同的测试要求快速生成高质量的测试用例集，广泛应用于产业界与学术界。

另外，测试用例集的优化改进部分的研究我们也做了很多，测试用例集的优化主要有几方面的工作，**首先是测试规模的约简工作**，在保证覆盖规模的基础上，降低测试成本，组合测试的理论与方法》一书也提到了，我们提出的基于组合覆盖的约简算法，约简效果显著，**其次是测试用例的排序工作**，排序目标通常是指最小化测试成本及更快的暴露测试问题，最小化测试成本即测试用例切换时会造成一定的切换成本，故而我们考虑使用基于切换成本的排序方式，例如基于海明距离的排序，另外，排序为了更快暴露测试问题的排序方式，主要指基于覆盖的排序方法，有时甚至可以通过排序实现前20%的测试用例达到80%的覆盖率，在测试用例规模庞大，用例集全部测试成本较高的情况下，基于覆盖的排序往往非常适用。最后是**基于测试用例执行判定的优化工作，**测试用例的设计最重要的一点是可测性，如果测试用例不可测或测试结果难以判定，那么测试用例就毫无意义，所以我们拟根据测试用例的可判定性优化用例，调整用例局部来保证测试可判定，例如构造两条或多条包含一定蜕变关系的测试用例加以判定测试结果等。

#### 3.3 技术革新，智能测试

测试用例的执行往往耗时耗力，我们拟实现测试用例执行自动化及执行结果判定智能化。首先测试用例执行智能化是指开发出辅助测试工具，将测试用例集逐条自动调用，注入到模拟平台待测系统中，例如可以在待测软件系统中注入测试单元，与测试工具存在接口调用关系，满足自动调用新一轮测试用例的要求。对于计算要求庞大的测试，可以使用云计算技术提升测试能力，也可以采用并行技术实现对多条用例的并行测试，降低测试成本。

对于执行结果判定智能化方面，我们拟根据对智能化软件系统特征分析、提取、归纳等步骤，寻找软件系统中可用的蜕变关系或约束条件，进而总结成一系列用于判定执行结果的准则。另一方面，机器智能特征提取代替人工特征提取，用例执行之后，很多测试结果需要对智能化软件表现出的特征进行提取，例如音频、视频、图像信息等，但之前的工作很多都需要人工来进行判定是否正常，我们拟开发出机器智能提取特征的方式，解放测试人员工作，提升测试智能化。

#### 3.4 科学评估，重点排查

测试评估是软件测试的重要研究阶段，衡量测试是否具有高效性，测试结果是否满足预期等。我们拟借鉴传统软件测试的评估技术，提出科学的智能化软件系统测试评估指标，类似于传统软件测试中的路径覆盖率、组合覆盖率等，首先对该指标的科学性与可靠性进行规范说明，提出该指标的适用场景，所揭示的意义等，并利用案例研究对该指标的科学性进行论证，之后我们利用所提出的智能化软件系统测试评估指标对之前的测试用例集进行有效评估，发现测试用例集在覆盖强度上的缺失与冗余，生成完备的测试报告，对测试集的结果及特点进行详细表述。

对于测试集中未通过的测试用例进行重点分析，我们拟借鉴传统软件测试方法中故障诊断技术，分析智能化软件系统可能存在的故障，甚至需要生成一批额外的测试用例，辅助我们进行测试的定位，帮助测试人员快速诊断、识别、定位出隐藏的软件故障。

#### 3.5 综合运用，安全诊断

传统软件的故障自动修复技术方面已经有了许多研究，而智能化软件的故障修复研究较少，我们拟借鉴软件测试方法体系中故障修复的相关研究，应用在智能化软件的故障修复故障中，首先需要充分调研智能化软件系统的历史故障报告，了解故障一般出现的位置，故障特点，故障触发条件及故障修复手段等，建立一个智能化软件系统的故障数据库，其次对于故障的特点也需要重点分析，故障的种类很多，智能化软件故障更可能是多重故障引发的，我们拟分析故障特点，归纳每一类故障的自动修复策略及故障修复后测试策略，故障修复技术可靠性需要保证，我们要确保每一次故障修复完成之后，原故障得到了正确修正的同时，没有新的故障引入到系统中。

安全漏洞问题对软件发展至关重要，软件的安全可靠并不意味着没有Bug，我们在传统软件的基于安全报告的安全缺陷报告识别及故障定位方面研究非常深入，我们拟根据智能化软件系统的安全报告，识别智能化软件中的安全缺陷及故障定位，并依此在测试报告中提出相关的安全问题，作为后续软件开发与维护的重要考量因素。

#### 3.6 回归测试，数据驱动

智能化软件的测试技术与方法是一个全新的领域，为了尽可能提升测试的科学性，我们拟采用测试产生数据，数据优化测试的回归测试理念，数据包含软件数据，如软件版本信息、输入数据、训练数据及中间数据等，也包含测试数据，如测试报告、安全报告、测试日志等，我们拟利用数据挖掘技术及大数据处理技术等，对数据进行信息挖掘，与测试目标相结合，进一步优化测试技术。

测试模型建模阶段，拟采用自适应建模机制，利用数据驱动进一步优化测试模型，例如一开始对模型输入域的认识可能不够导致模型输入覆盖不够，也可能对模型诸多模块之间的数据交互认识不足导致模块之间建模不充分等，这些都可以通过数据驱动逐渐加以改进优化模型。

测试生成阶段，拟采用自适应用例生成算法，利用数据驱动改进用例生成的效率，例如一些测试用例执行后可能测试结果不够明显或用例存在一定的约束关系，我们可以根据测试日志优化改进生成算法，减少或修改此类用例的生成策略等。

测试执行判定阶段，拟采用可扩展的用例判定策略，利用数据挖掘技术，分析获取新的蜕变关系，优化及新增判定策略，继而提升执行结果判定的效率。

故障诊断阶段，拟逐步优化现有的故障诊断与定位技术，利用数据挖掘技术中分类器和决策树等相关研究，通过对历史故障数据的学习，训练出一个可靠的故障分类器与定位器，从而提升故障诊断与定位的精确度。

## **本项目的特色与创新之处；**

1. 我们在前期已经做了非常充分的调研，收集整理了迄今为止，几乎所有的  
   软件测试方法和相关理论，并对这些理论和方法进行初步的分类和处理。以此为基础，期望在软件测试的理论和方面做一些新的创新。 另外，我们系统整理了智能化软件的典型应用，分析了不同智能化软件的特征。而且，“前事不忘，后事之师”，我们收集了相当一部分的智能化软件的历史故障数据，建立历史故障数据库，作为项目开展的重要指导与依据。
2. 我们采用灵活的自适应测试框架，对于测试建模、生成算法和测试策略都不是固定的，我们不必过于担心每一步骤是否考虑周全，而是可以根据测试结果反向改进，建模、生成等策略，测试用例的生成不是唯一的，可以采用多种生成算法进行横向比较，也可以采用回归优化进行纵向比较。另外，生成算法一方面借鉴了传统软件方法体系中典型生成算法，也融入了对判定智能化的考虑，测试用例的生成必须保证可判定性。
3. 我们采用蜕变测试解决智能化软件测试中执行判定复杂的问题，本项目可以根据测试数据，利用数据挖掘方法，提取可能的蜕变关系，加以利用在执行判定的过程中，这是蜕变测试应用在全新领域的一次尝试与创新。
4. 传统软件的评估技术应用在智能化软件的测试领域，提出了一种科学可靠的测试评估指标，并具有一定的广泛性与应用性，可以很好衡量任意一种面向智能化软件的测试集的充分性，类似于传统软件测试的覆盖率指标，可以对未来面向智能化软件的测试提供很好的指导意义。
5. 传统的软件测试更多关注导致软件失效的故障，较少关注潜在的软件安全漏洞，本项目将安全缺陷的识别处理技术应用在智能化软件的测试领域，基于自然语言处理和机器学习技术实现软件安全漏洞的识别和定位，为软件维护提供指导，从而使智能软件系统的部署更安全可靠。

6、我们将数据驱动式的测试技术应用在测试框架中，分析了数据来源，数据种类及数据处理方式与手段等，提出了数据驱动式测试框架，并且利用了数据挖掘中一系列的前沿技术，改进了测试流程，优化了测试步骤与算法，是数据驱动式测试一个很有代表性的应用。

## **5．年度研究计划及预期研究结果**（包括拟组织的重要学术交流活动、国际合作与交流计划等）。

# （二）研究基础与工作条件

## 1．**研究基础**（与本项目相关的研究工作积累和已取得的研究工作成绩）；

## 2．**工作条件**（包括已具备的实验条件，尚缺少的实验条件和拟解决的途径，包括利用国家实验室、国家重点实验室和部门重点实验室等研究基地的计划与落实情况）；

南京大学计算机软件新技术国家重点实验室目前拥有各类高性能服务器300余台套,其中大型设备公共云计算平台一套（20余台多核高性能服务器、2台IBM Flex刀片服务器及13台GPU服务器作为计算资源，同时具备60T的高速数据存储能力和160T的备份存储、IBM虚拟存储网关SVC 2台、IBM Flash System 840闪存阵列1台）、大数据并行计算共享平台一套（140个节点，提供MapReduce、MPI、BSP等多种并行编程模型和环境）、EasyStack云平台一套（包括Dell 高性能计算服务器12台）、IBM刀片服务器2台（14片HS21、12片HS22、2片JS12）、IBM 3850 M2高性能服务器4台（1台96核、1台64核，2台32核）、TMS和华赛高速存储各1台、40余台GPU服务器。机房采用160KVA UPS三机系统并机运行，保证各设备24小时不间断工作。万兆接入校园网，同时支持IPv4/IPv6双栈协议，串接万兆安全网关一台，支持防火墙、IPS、WAF等多项安全功能。实验验证、仿真和测试环境非常好。此外，实验室还购买了大量软件资源并订阅了大量的电子刊物全文数据库，例如ACM、Blackwell、Elsevier、IEEE/IEE、Kluwer、Springer、Wiley等，方便研究人员查阅国际最新文献资料。

## 3．**正在承担的与本项目相关的科研项目情况**（申请人和项目组主要参与者正在承担的与本项目相关的科研项目情况，包括国家自然科学基金的项目和国家其他科技计划项目，要注明项目的名称和编号、经费来源、起止年月、与本项目的关系及负责的内容等）；

## 4．**完成国家自然科学基金项目情况**（对申请人负责的前一个已结题科学基金项目（项目名称及批准号）完成情况、后续研究进展及与本申请项目的关系加以详细说明。另附该已结题项目研究工作总结摘要（限500字）和相关成果的详细目录）。

# （三）其他需要说明的问题

## 1. 申请人同年申请不同类型的国家自然科学基金项目情况（列明同年申请的其他项目的项目类型、项目名称信息，并说明与本项目之间的区别与联系）。

## 2. 具有高级专业技术职务（职称）的申请人或者主要参与者是否存在同年申请或者参与申请国家自然科学基金项目的单位不一致的情况；如存在上述情况，列明所涉及人员的姓名，申请或参与申请的其他项目的项目类型、项目名称、单位名称、上述人员在该项目中是申请人还是参与者，并说明单位不一致原因。

## 3. 具有高级专业技术职务（职称）的申请人或者主要参与者是否存在与正在承担的国家自然科学基金项目的单位不一致的情况；如存在上述情况，列明所涉及人员的姓名，正在承担项目的批准号、项目类型、项目名称、单位名称、起止年月，并说明单位不一致原因。

## 4. 其他。