**拟攻读博士学位科研计划**

姓名：付安 报考专业：计算机科学与技术 导师：孙昌爱

1 研究方向与目标

解决蜕变测试中蜕变关系获取困难的问题及如何构建高故障检测能力蜕变关系集的问题。

2研究背景与意义

软件测试是一种保障软件质量的技术，目的在于发现软件中潜藏的故障。随着软件工程的复杂程度不断提升，软件测试在整个软件开发周期中占比越来越大，并得到广泛的重视。据统计，在软件开发总工作量中，软件测试的工作量占到40%以上[1]。通常，软件测试通过有限的输入判定输出是否与预期一致来检查软件中是否潜藏故障。只有在可以毫不含糊地确认软件测试是否通过时才能认为这一次测试有用[2]，但是在某些情况下，软件测试中存在着Oracle问题，即软件测试人员无法确定软件的执行结果与预期结果是否相同，或者很难构造预期输出结果[3]。

蜕变测试[4]是缓解Oracle问题的一种软件测试技术，从成功的测试用例中构建新的测试用例，以揭露深层次的软件故障。其原理是通过检查程序的多个执行结果之间是否满足一定的关系来检查程序中的故障[5]。蜕变测试的原理如图1所示：



**图 1 蜕变测试原理图**

首先，利用其他测试用例生成方法为待测程序生成一定数量的测试用例，使用上述测试用例测试待测程序，其中成功的测试用例作为待测程序的原始测试用例x；然后，根据待测程序的必要属性，构造符合实际的蜕变关系(r, rf)，并根据构造的蜕变关系中输入关系r为原始测试用例生成对应的衍生测试用例r(x)； 最后，使用原始测试用例与衍生测试用例执行待测程序，得到对应的输出结果P(x)与P(r(x))，检查原始测试用例的输出结果P(x)与衍生测试用例的输出结果P(r(x))是否满足蜕变关系的输出关系rf。若不满足rf则程序有故障，若满足rf则说明此蜕变关系不能检测到程序的故障。

综上所述，蜕变测试通过比较原始测试用例对应的输出与衍生测试用例对应的输出之间是否满足蜕变关系来检测程序的故障，不需要构造预期输出，避开了oracle问题。将蜕变测试应用于预期输出结果获取困难的程序，能够有效地缓解这些程序中特有的测试困难问题。

蜕变测试在很多领域能够成功应用，包括医疗服务[6]、空中交通管制[7]、机器学习[8]等。尽管如此，蜕变测试技术与其他软件测试技术的创新性集成和蜕变测试技术的理论基础研究发展甚微，尤其体现在测试用例选取和蜕变关系识别这两个核心方面。Groce等人认为蜕变关系的识别是一项需要手工完成的困难任务，即使专家也会感到发现蜕变关系是一件很困难的事情[9]。

由蜕变测试的原理图我们可以清晰地看到，蜕变关系在整个蜕变测试的过程中占有重要的地位，其原因是蜕变关系对于程序故障检测至关重要。在传统的软件测试方法中，测试人员通过比较待测程序的预期输出与实际输出是否一致来判断程序中是否潜藏故障；在蜕变测试中，测试人员通过比较程序的多个实际输出是否满足蜕变关系来判断程序中是否潜藏故障。对于测试预期获取困难的程序来说，蜕变关系扮演着测试预期的角色。若蜕变关系的获取存在障碍，则直接影响蜕变测试的实际应用效果及故障检测能力。因此，研究蜕变关系的获取技术对蜕变测试及解决测试预期获取困难问题具有重要的价值。

另一方面，蜕变关系集的故障检测能力有高低之分，高故障检测能力的蜕变关系集能够帮助测试人员更高效地找到软件中潜藏的故障。如何构建高故障检测能力的蜕变关系集对于提高蜕变测试故障检测效率具有重要的价值。

3 国内外研究现状

蜕变测试是一种缓和软件测试中Oracle问题的革新性方法。原理是通过比较程序的多个执行结果之间的关系来检验程序是否出错，从而不需要构造程序的预期输出，缓和了软件测试中的Oracle问题。Chen的观点认为，成功的测试用例中包含着丰富的信息，不可任意抛弃。选取成功的测试用例为蜕变测试中的原始测试用例，然后根据一定的蜕变关系来构造衍生测试用例。将原始测试用例与衍生测试用例输入至程序中运行得到各自对应的执行结果。蜕变测试通过比较原始测试用例的执行结果与衍生测试用例的执行结果之间是否满足一定的蜕变关系来测试程序的。

Chen等人在2004年研究了蜕变测试和特殊用例测试在程序错误检测上的差异[12]。研究得出的结论是：蜕变测试能够发现特殊用例测试不能够发现的错误；特殊用例测试的缺陷在于不能够确定一个特殊用例测试用例集是否是可靠的，蜕变测试通过判断输出是否满足蜕变关系缓和了这一问题；蜕变测试可以作为特殊用例测试的一种补充。

Chen等人在2004年通过案例研究了关于自动化测试的优良蜕变关系选取策略[13]，得出的结论是：（1）一个问题领域的理论知识不能够作为区分蜕变关系好与坏的标准；（2）优良的蜕变关系应该是那些可以使程序的多个执行尽可能不同的蜕变关系。

Mayer和Guderlei在2006年通过实证研究方法研究了优良蜕变关系的选择[14]。研究显示：（1）形成等式的蜕变关系相对于其他蜕变关系来说检错能力较弱；（2）如果蜕变关系是一个两边都为线性组合的等式，那么它不会对错误的增加有影响；（3）好的蜕变关系由许多逻辑符号构成；（4）如果蜕变关系接近于典型算法所使用的策略，那么他们的适用性有限。

Liu等人在2014年通过对五个程序的实验研究了蜕变测试缓解Oracle问题的效果[15]，回答了蜕变测试缓解Oracle问题的能力、测试人员是否能简单并成功地使用蜕变测试发现错误和影响蜕变测试效果的关键因素三个问题。研究发现，对于每个程序来说，蜕变关系集合与预期测试结果有相近的错误发现能力，并且蜕变测试的成本效率可以通过减少使用蜕变关系的数目来提升。另一个发现是影响蜕变测试的成本效率的关键因素是所使用蜕变关系的多样性。

Chen等人在2015年提出了METRIC蜕变关系识别框架。METRIC基于CHOC’LATE测试用例生成方法，实现了蜕变关系识别的半自动化。它让软件测试人员集中精力去比较两个由不同输入构成的测试帧，判断两者对应的输出之间是否存在一定的蜕变关系。相比于传统的同时考虑一个系统的输入输出两方面，METRIC以系统地方法进行蜕变关系识别，大大减少了软件测试人员识别蜕变关系的工作量。

吴鹏提出了迭代蜕变测试[20]，并就此方法提出了迭代蜕变测试算法(IMT)。吴鹏通过MTest测试环境对迭代蜕变测试进行研究后发现：迭代蜕变测试技术优于蜕变测试技术，并且在测试用例生成方面具有显著的高效性。

董国伟等人认为，一些蜕变关系在蜕变测试中的检错能力不足，并在2008年提出了复合蜕变关系构造方法[21]。这种方法以成对的方式将现有的蜕变关系组合起来产生新的蜕变关系。由于新的复合蜕变关系含有所有形成它的蜕变关系的优点，故其错误检测能力优秀；另一方面，蜕变关系的数量会大幅减少，因此一个程序可以使用更少的测试用例来测试。研究发现：在复合蜕变关系使用后，测试效率大幅增加。

Chen等人在2009年将蜕变测试技术应用于生物信息学软件的测试[16]。在研究中蜕变测试被应用于GNLab和SeqMap进行软件错误的检测。实验发现蜕变测试技术在测试时简单易用，缓解了生物信息学软件测试中难以确定软件输出的问题，蜕变测试技术也成功应用于实际软件的错误检测。

Segura等人在2011年使用蜕变测试技术让特征模型测试数据的生成自动化[17]，并对利用变异测试方法生成的特征模型进行错误检测。结果发现绝大多数错误可以被自动发现，且蜕变测试技术在特征模型错误检测方面具有高效性和有效性。

Poon等人在2014年对电子数据表的错误检测应用了蜕变测试技术[18]。结果显示蜕变测试技术在电子表格错误检测方面卓有成效，并且作为一种普通的技术，终端用户可以很容易地将它使用在各种各样的电子表格程序上。

吴鹏等人在2005年对于蜕变测试和特殊用例测试进行了实例研究[19]，提出了基于蜕变测试方法的集成测试环境MTest。分别以变异得分和错误发现率为度量指标,分析和对比了特殊用例测试、以特殊测试用例和随机测试用例为源测试用例的蜕变测试这三种方法的测试能力和效率。实验结果表明,蜕变测试与特殊用例测试之间是互补的，随机测试用例在测试能力和效率上优于特殊测试用例。

4 研究内容与思路

目前，对蜕变测试的研究主要集中于蜕变测试和其他软件测试方法的结合以及蜕变测试的实际应用。针对蜕变测试理论的研究十分有限，针对蜕变关系获取方法的研究也十分有限。蜕变关系获取方法的研究能够帮助测试人员系统地获取软件中存在的蜕变关系，促进蜕变测试理论的发展与蜕变测试的实际应用。高故障检测能力蜕变关系集构造方法的研究能够帮助测试人员更加高效地进行软件测试，提高蜕变测试故障检测的效率。

针对蜕变测试中蜕变关系获取困难的问题，本人希望提出一种新的蜕变关系获取方法来解决这一问题；针对如何构建高故障检测能力蜕变关系集的问题，本人的基本思路是，分析高故障检测能力的蜕变关系集所具有的特征，发现构建高故障检测能力蜕变关系集的相关规则。

6 开展相关研究的基础

目前，本人参与了蜕变关系识别框架METRIC改进方法的研究课题，并取得了一定的研究成果。METRIC是一种基于CPM方法的蜕变关系识别方法，能够帮助软件测试人员系统地识别软件中存在的蜕变关系。METRIC的优点在于，让软件测试人员集中精力去比较两个由不同输入构成的完整测试帧，判断两者对应的输出之间是否存在一定的蜕变关系，而不是同时考虑一个系统的多个输入参数与输出参数，简化了蜕变关系的识别问题。METRIC的不足之处是没有完整测试帧选取策略，影响了识别蜕变关系的效率。课题组将软件的输出行为引入METRIC中对其进行了改进，以提高其蜕变关系识别效率。

以上述工作为基础，可以开展高故障检测能力蜕变关系集所具有特征的研究。发掘高故障检测能力蜕变关系集的特征，可以指导测试人员构建高故障检测能力蜕变关系集，并形成构建高故障检测能力蜕变关系集的启发式规则。

7 参考文献

1. 单锦辉, 姜瑛, 孙萍. 软件测试研究进展[J]. 北京大学学报:自然科学版, 2005, 41(1):134-145.
2. Meyer B. Seven Principles of Software Testing [J]. Computer, 2008, 41(8):99-101.
3. Weyuker E J. On Testing Non-Testable Programs [J]. Computer Journal, 1982, 25(4):465-470.
4. Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: A new approach for generating next test cases[R]. Technical Report HKUST-CS98-01, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, 1998.
5. 董国伟, 徐宝文, 陈林,等. 蜕变测试技术综述[J]. 计算机科学与探索, 2009, 3(2):130-143.
6. Murphy C, Raunak M S, King A, et al. On effective testing of health care simulation software [C]//Proceedings of the 3rd Workshop on Software Engineering in Health Care. ACM, 2011: 40-47.
7. Hui Z, Huang S, Ren Z, et al. Metamorphic Testing Integer Overflow Faults of Mission Critical Program: A Case Study [J]. Mathematical Problems in Engineering, 2013, 29(1):61-61.
8. Xie X, Ho J W K, Murphy C, et al. Testing and validating machine learning classifiers by metamorphic testing [J]. Journal of Systems and Software, 2011, 84(4): 544-558.
9. Groce A, Kulesza T, Zhang C, et al. You are the only possible oracle: Effective test selection for end users of interactive machine learning systems [J]. IEEE Transactions on Software Engineering, 2014, 40(3): 307-323.
10. Chen T Y, Poon P L, Tse T H. A Choice Relation Framework for Supporting Category-Partition Test Case Generation [J]. IEEE Transactions on Software Engineering, 2003, 29(7):577-593.
11. Chen T Y, Poon P L, Xie X. METRIC: METamorphic Relation Identification based on the Category-choice framework [J]. Journal of Systems & Software, 2015, http://dx.doi.org/10.1016/j.jss.201507037.
12. Chen T Y, Kuo F C, Liu Y, et al. Metamorphic Testing and Testing with Special Values.[C]// Proceedings of Acis International Conference on Software Engineering. 2004:128-134.
13. Chen T Y, Huang D H, Tse T H, et al. Case studies on the selection of useful relations in metamorphic testing[C]//Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004). 2004: 569-583.
14. Mayer J, Guderlei R. An Empirical Study on the Selection of Good Metamorphic Relations[C]// Proceedings of IEEE 37th Annual Computer Software and Applications Conference. IEEE Computer Society, 2006:475-484.
15. Liu H, Kuo F C, Towey D, et al. How Effectively Does Metamorphic Testing Alleviate the Oracle Problem? [J]. IEEE Transactions on Software Engineering, 2014, 40(1):4-22.
16. Chen T Y, Ho J W, Liu H, et al. An innovative approach for testing bioinformatics programs using metamorphic testing [J]. Bmc Bioinformatics, 2009, 10(2):343-347.
17. Segura S, Hierons R M, Benavides D, et al. Automated metamorphic testing on the analyses of feature models [J]. Information and Software Technology, 2011, 53(3): 245-258.
18. Poon P L, Kuo F C, Liu H, et al. How can non-technical end users effectively test their spreadsheets? [J]. Information Technology & People, 2014, 27(4): 440-462.
19. Wu P, Shi X C, Tang J J, et al. Metamorphic Testing and Special Case Testing: A Case Study[J]. Journal of Software, 2005, 16(7):1210-1220.
20. Wu P. Iterative metamorphic testing [C]//Proceeding of the 29th Annual International Computer Software and Applications Conference (QSIC 2007), 2005, 1: 19-24.
21. Dong G W, Xu B W, Chen L, et al. Case studies on testing with compositional metamorphic relations [J]. Journal of Southeast University (English Edition), 2008, 24(4): 437-443.