适应性分区测试

孙昌爱

教授, 博士生导师

北京科技大学 计算机与通信工程学院

Our research effort

■ **Developing techniques and tools for *reliable* and *adaptive* Service-Centric Systems**

Adaptive Service Composition
- -Language
- -Methodology
- -Process and Framework
- -Supporting Platform (engine, analysis, design, and management)

1

Transaction Management
- -Requirement analysis
- -Integration Solution (Two approaches)
- -Transactional WS and Middleware

2

Testing SOA Software
- - Metamorphic Testing (MT)
- - Dynamic Random Testing
- - Scenario-Oriented Testing
- - Mutation Testing

3

Debugging BPEL Programs
- - Framework and guidelines
- - Block- and Predicate-based
- - Slicing- and Distance-based

4

Our research effort (cont.)

■Developing novel software testing techniques and tools

Mutation Testing

-Mutant reduction/selection
-Mutation acceleration
-Mutation testing of new types of applications (BPEL, Apps, EB)

1

Fault-based Testing

-Developing new strategies
-Empirical evaluation of strategies
-Developing tools for Boolean specifications

2

Metamorphic Testing

-MT Theory (MR Acquisition, Evaluation and Selection)
-MT framework for different domains of applications
-Empirical Evaluation

3

Model-based Testing

-Concurrent Programs
-Web Services
-BPEL Programs
-Tools and Case studies

4

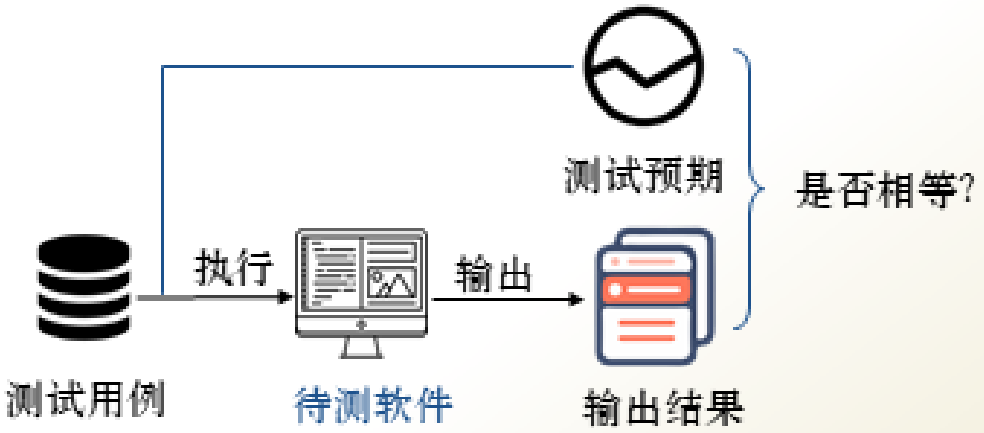
报告大纲

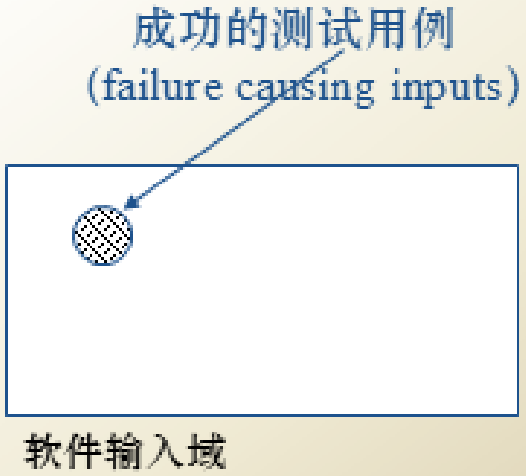
- 研究背景
  - ✓ 随机测试/分区测试/随机性分区测试
  - ✓ 研究动机
- 适应性分区测试
  - ✓ 原理
  - ✓ 算法
- 实验评估
- 相关工作
- 总结

研究背景—软件测试

■ 软件测试

➢ 从待测软件的输入域中选择一组测试用例，利用这些测试用例执行软件，比较实际输出结果与预期输出结果是否相符。若不符合，则表明待测软件中存在故障。





研究背景—随机测试

■ 随机测试

- 一种广泛采用的测试技术，从**整个输入域**中**随机地**与**独立地**选择测试用例。
- **简单且易于实现。**
- 没有利用测试的过程信息和待测软件的信息，导致测试效率可能不高。
- 提高随机测试的故障检测效率：
  - **Anti-random testing**：选择与已执行所有测试用例距离之和最大的测试用。
  - **Adaptive random testing**：在软件输入域中均匀地选择测试用例。

•Y. K. Malaiya. Antirandom testing, Getting the most out of black-box testing. *Proceedings of the 6th International Symposium on Software Reliability Engineering (ISSRE1995)*, 1995, pp. 86-95.
•T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. *Proceedings of the 9th Asian Computing Science Conference (ASIAN2004)*, 2004, pp. 320 – 329.

研究背景—分区测试

■ 分区测试 (Partition Testing)

- 将待测试软件的**输入域划分为多个不相交的分区(范畴划分、等价类划分决策表、分类树等)**，从每个分区中选择一个或多个测试用例执行。
- 理想情况下，分区具有同构性；实际情形中，该性质难以保证，分区测试的效率不稳定。
- 分区测试与随机测试比较
  - T.Y. Chen等人通过理论分析得到分区测试的故障检测效率**高于或等于**随机测试的充分条件：

$$\frac{n_1}{d_1} = \frac{n_2}{d_2} = \cdots = \frac{n_m}{d_m}$$

m为分区数；n_i是在第i^{th}分区内执行的测试输入；d_i是第i^{th}分区所有输入的数量

• T. Y. Chen, Y. T. Yu. On the relationship between partition and random testing. *IEEE Transactions on Software Engineering*, 1994, 20(12): 80-90.

研究背景—研究动机

■ 随机分区测试 (Random Partition Testing)

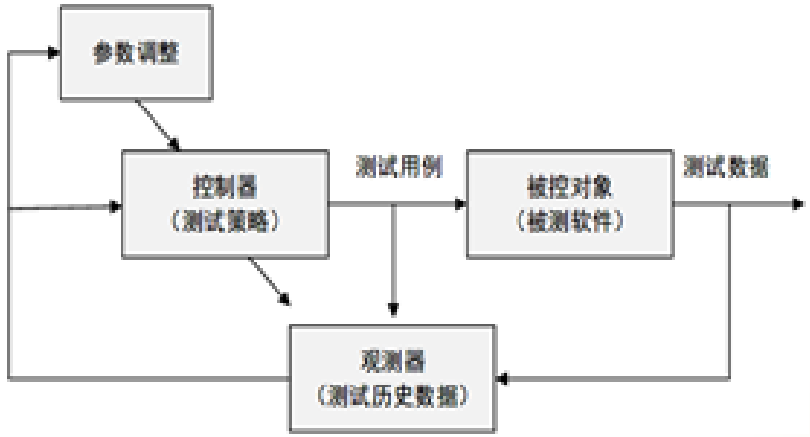
- 依据不同的方法将待测试软件的输入域划分为m个不相交的分区$C_1, C_2, \ldots, C_m$，并设置每一个分区的选择概率$p_i (i = 1,2,\ldots,m)$，然后依据测试剖面选择分区C_i，最后从C_i中随机地选择测试用例并执行。
- 在测试的过程中，不变地测试剖面无法适应不断变化的分区失效率，影响分区测试的故障检测效率。

■ 如何提高分区测试的故障检测效率?

• K-Y Cai. Optimal software testing and adaptive software testing in the context of software cybernetics. *Information and Software Technology* , 2002, 44(14): 841 – 855.

适应性分区测试—基本原理

■ 软件测试的控制论原理



- **蔡开元教授在APAQS 2001提出。**
- **软件控制论**旨在探讨计算机软件与控制领域的跨学科问题。
- **核心科学问题包括**：如何建立软件行为的控制模型、设计方法与控制理论，以及有效、定量化的控制策略。

■ 适应性分区测试

- 将软件控制论引入到分区测试，利用“引起故障的输入趋向于集簇在连续的区域”这一观察。
- 根据测试的结果信息适应性地**调整测试剖面**：**失效率大**的分区**被选择的概率高**，**失效率小的分区**被选择的**概率小**。

适应性分区测试—MAPT算法

■ MAPT：基于Markov链的适应性分区测试

➢在测试的过程中，故障不断移除，最近的若干次测试结果能比较准确地反应分区的失效率。

➢Markov链具备“无后效应”，即要确定过程将来的状态，只需要知道当前的情况：

$$P\{X_{n+1}=j \mid X_0=i_0, X_1=i_1, \cdots, X_{n-1}=i_{n-1}, X_n=i_n\}=P\{X_{n+1}=j \mid X_n=i\}$$

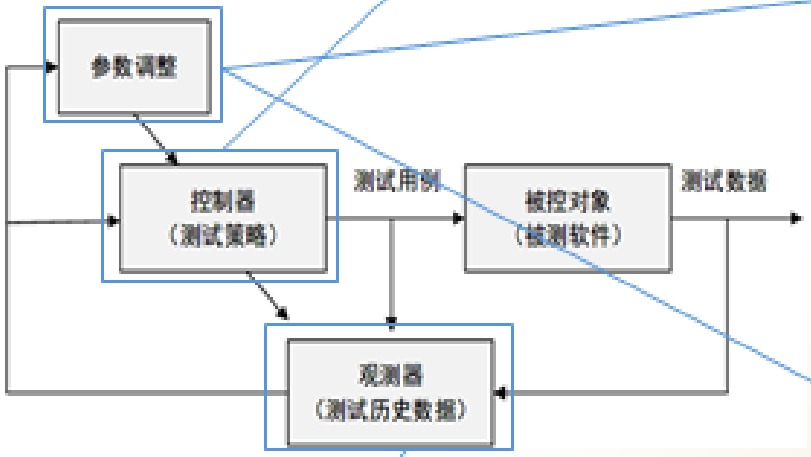
其中，$\{i_0, i_1, \ldots, i_n\}$为有限个不同的状态

➢**MAPT将分区视为待测软件的状态，则选择分区的过程看成状态转移的过程**，利用当前的测试结果更新不同状态之间的转移概率。

适应性分区测试—MAPT算法

■ MAPT算法

1. 初始状态转移矩阵$P = p_{i,j}\ (i,j = 1,2,\ldots,m)$
2. 依据状态转移矩阵选择分区，并随机选择一个测试用例并执行



3. 观察选中的测试用例执行的结果。如果在执行从分区C_i中选择的测试用例检测到故障，那么记$z_i = 1$，否则记$z_i = 0$

4. 如果$z_i = 1$，增大目前的状态转移到本身的概率$p_{i,i}$,同时减小转移到其它状态的概率$p_{i,j}\ (i \neq j)$,并把缺陷移除。

$$p_{ij} = \begin{cases} p_{ij} - \dfrac{\gamma \times p_{ii}}{m-1}, & p_{ij} \geq \dfrac{\gamma \times p_{ii}}{m-1} \\ p_{ij}, & p_{ij} < \dfrac{\gamma \times p_{ii}}{m-1} \end{cases} \quad (i \neq j)$$

$$p_{ii} = 1 - \sum_{i \neq j} p_{ij}$$

5. 如果$z_i = 0$，减小目前的状态转移到本身的概率$p_{i,i}$,同时增大转移到其它状态的概率$p_{i,j}\ (i \neq j)$,并把缺陷移除。

$$p_{ij} = p_{ij} + \frac{\gamma \times p_{ij}}{m-1}$$

$$p_{ii} = p_{ii} - \frac{\gamma \times (1 - p_{ii})}{m-1}$$

6. 检查终止条件，若满足，则终止测试；否则跳转步骤2

适应性分区测试—RAPT算法

■ RAPT：基于奖惩机制的适应性分区测试

➢ 反馈信息和奖惩机制广泛运用于软件测试，取得了良好的测试效果。

- 蔡开元等人利用部分历史数据来估计待测程序中的故障数目与分区的失效率；
- Zhou等人将检测出故障的测试用例设置较高的执行优先级，没有检测出故障的测试用例设置较低的执行优先级，提出了DPIS分区策略。

➢ **RAPT**利用测试的**历史数据**更新**测试剖面**：

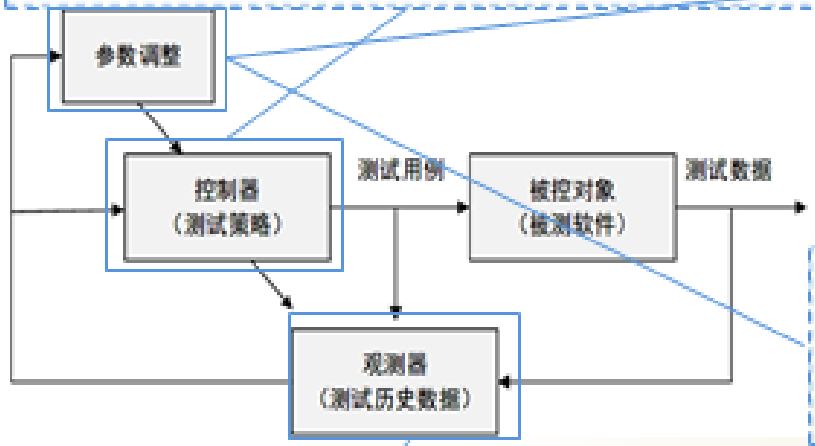
- 揭示故障数目多的分区应当“奖励”更大的被选择概率；
- 揭示故障数目少的分区应当“惩罚”接受较小的被选择概率。

- K-Y Cai, B. Gu, H. Hu H. Adaptive software testing with fixed-memory feedback. *Journal of Systems and Software*, 2007, 80(8):1328-1348.
- Z. Q. Zhou, A. Sinaga, W. Susilo. A cost-effective software testing strategy employing online feedback information. *Information Sciences*, 422 (2018): 318-335.

适应性分区测试—MAPT算法

■ RAPT算法

1. 初始测试剖面$\{<C_1, p_1>, <C_2, p_2>, \ldots, <C_m, p_m>\}$，并设置惩罚上限惩罚上限boundary = Z
2. 依据测试剖面选择分区，并随机选择一个测试用例并执行



3. 观察选中的测试用例执行的结果。如果在执行从分区C_i中选择的测试用例检测到故障，设置C_i分区的奖励因子$reward_i = reward_i + 1$，惩罚因子$punishment_i = 0$；如果没有检测出故障，则C_i分区的惩罚因子$punishment_i = punishment_i + 1$。

4. 如果$reward_i \neq 0$，增大C_i分区被选择的概率p_i,同时减小其它分区被选择的概率p_j ,并把缺陷移除。

$$p_j = \begin{cases} p_j - \dfrac{(1+\ln reward_i)\times\varepsilon}{m-1}, & p_j \geq \dfrac{(1+\ln reward_i)\times\varepsilon}{m-1} \\ 0, & p_j < \dfrac{(1+\ln reward_i)\times\varepsilon}{m-1} \end{cases}$$

$$p_i = 1 - \sum_{i\neq j} p_j$$

5. 如果$reward_i = 0$，减小目前的状态转移到本身的概率$p_{i,i}$,同时增大转移到其它状态的概率$p_{i,j}$ $(i \neq j)$，如果该分区的惩罚因子$punishment_i \geqslant boundary$则该分区对应的概率$p_i$=0，并把缺陷移除。

$$p_i = \begin{cases} p_i - \delta, & p_i \geq \delta \\ 0, & p_i < \delta \ or \ \text{punishment}_i = boundary \end{cases}$$

$$p_j = \begin{cases} p_j + \dfrac{\delta}{m-1}, & p_i \geq \delta \\ p_j + \dfrac{p_i}{m-1}, & p_i < \delta \ or \ \text{punishment}_i = boundary \end{cases}$$

6. 检查终止条件，若满足，则终止测试；否则跳转步骤2

实验评估—实验对象

实验对象

- 从Software-artifact Infrastructure Repository（SIR）中选取如下软件的多个版本（包括相应的测试用例集与故障）：
  - gzip：GNU开发的一个常用的压缩工具
  - grep：可以利用正则表达式进行全局搜索的工具
  - make：自动化编译工具，从makefile中读取指令，然后编译目标文件

程序	代码行数	测试用例数目	版本号	故障数目
grep	10068	470	V1	2
			V2	2
			V3	2
			V4	1
gzip	5680	214	V1	3
			V2	1
			V3	2
			V4	3
make	35545	793	V1	2
			V2	1

实验评估—实验设置

■ 分区设置

➢ 利用Category-Partition Method (CPM)方法进行分区，考虑不同粒度的分区模式。

程序	分区模式		
	粗粒度	中等粒度	细粒度
grep	3	9	13
gzip	4	6	12
make	4	8	16

■ 初始化测试剖面

➢ 均等和成比例的方式设置初始测试剖面。

■ 度量指标

- **F-measure:** 检测第一个故障需要的测试用例数
- **F2-measure:** 检测第一个故障后再次检测一个故障需要的测试用例数
- **T-measure:** 检测第一个故障需要的时间
- **T2-measure:** 检测第一个故障后再次检测一个故障需要的时间

实验评估—结果分析

■ 有效性评估

➢ 利用Holm-Bonferroni 方法和effect size来区分不同测试技术的故障检测能力

	RPT	DRT	MAPT	RAPT
RPT	—	49	59	59
DRT	11	—	56	55
MAPT	1	4	—	40
RAPT	1	5	20	—

Pair of techniques	Effect size
DRT vs. RPT	0.6249
MAPT vs. RPT	1.3500
RAPT vs. RPT	1.5234
MAPT vs. DRT	0.5749
RAPT vs. DRT	0.8134
RAPT vs. MAPT	0.3258

	RPT	DRT	MAPT	RAPT
RPT	—	31	36	36
DRT	11	—	33	38
MAPT	6	9	—	33
RAPT	6	4	9	—

Pair of techniques	Effect size
DRT vs. RPT	0.6221
MAPT vs. RPT	0.8956
RAPT vs. RPT	1.1822
MAPT vs. DRT	0.1889
RAPT vs. DRT	0.4769
RAPT vs. MAPT	0.3119

➢ MAPT和RAPT的F-measure和F2-measure小于RPT和DRT， RAPT和MAPT能够比RPT和DRT更快地揭示故障。

实验评估—结果分析(续)

■ 性能评估

➢ 利用Holm-Bonferroni 方法和effect size来区分不同测试技术的性能

	RPT	DRT	MAPT	RAPT
RPT	—	30	36	41
DRT	30	—	36	41
MAPT	24	24	—	39
RAPT	19	19	21	—

Pair of techniques	Effect size
DRT vs. RPT	0.2140
MAPT vs. RPT	0.2911
RAPT vs. RPT	0.4715
MAPT vs. DRT	0.3351
RAPT vs. DRT	0.4522
RAPT vs. MAPT	0.1793

	RPT	DRT	MAPT	RAPT
RPT	—	24	30	33
DRT	18	—	22	29
MAPT	12	20	—	32
RAPT	9	13	10	—

Pair of techniques	Effect size
DRT vs. RPT	0.1306
MAPT vs. RPT	0.2685
RAPT vs. RPT	0.6961
MAPT vs. DRT	0.1368
RAPT vs. DRT	0.5205
RAPT vs. MAPT	0.4640

➢ MAPT和RAPT的T-measure和T2-measure比RPT和DRT小;RAPT和MAPT能够比RPT和DRT更快地揭示故障，并且需要更少的时间来选择测试用例。

实验评估—结果分析(续)

■ 分区数目对APT故障检测效率的影响

➢利用Holm-Bonferroni 方法检验分区数目对APT故障检测效率的影响。

Comparison for F-measure

	Coarse	Medium	Fine
Coarse	—	36	32
Medium	44	—	**28**
Fine	48	**52**	—

Comparison for T-measure

	Coarse	Medium	Fine
Coarse	—	32	**27**
Medium	48	—	33
Fine	**53**	47	—

Comparison for F2-measure

	Coarse	Medium	Fine
Coarse	—	25	19
Medium	31	—	**17**
Fine	37	**39**	—

Comparison for T2-measure

	Coarse	Medium	Fine
Coarse	—	29	21
Medium	27	—	19
Fine	35	37	—

➢分区数目与APT的故障检测效率没有明显的关系。

- S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 1979, 6(1): 65 – 70.

实验评估—结果分析(续)

■ 初始测试剖面对APT故障检测效率的影响

➢ 利用Holm-Bonferroni 方法检验初始测试剖面对APT故障检测效率的影响。

Comparison for F-measure

	Equal	Proportional
Equal	—	67
Proportional	53	—

Comparison for F2-measure

	Equal	Proportional
Equal	—	28
Proportional	56	—

Comparison for T-measure

	Equal	Proportional
Equal	—	91
Proportional	29	—

Comparison for T2-measure

	Equal	Proportional
Equal	—	51
Proportional	33	—

- 成比例的初始测试剖面对APT检测第一个故障有积极的影响，然而对检测第二个故障有消极的影响。
- 初始测试剖面与APT的故障检测效率之间无明显的关系。

■ 适应性测试 (AT)

- 探索了控制论与软件测试的结合。
- 收集测试的过程信息，利用遗传算法、最小二乘和梯度下降等方法估计相关的参数值（故障数目、分区失效率等）。
- 具有较高的故障检测效率；需要递归计算大量的历史数据，导致选择测试用例的开销很大。

■ 动态随机测试 (DRT)

- 改进的AT测试策略。
- 基于当前的测试结果动态地更新测试剖面。
- 具有较高的故障检测效率其算法时间复杂度为O(m.n)(m为分区数，n为选择的测试用例数)；调整概率参数在测试过程中保持不变，应对分区失效率突变的情况时效果差。

■ 适应性随机测试 (ART)

- 在软件输入域中均匀地选择测试用例；
- 依据一种度量标准计算候选测试用例与已执行测试用例的距离，选择距离“最大”的测试用例作为下一个要执行的测试用例；
- 具有较高的故障检测效率；但是选择下一个测试用例的时间开销较大。

总结

- **提出了适应性分区测试**
  - 将随机测试和分区测试结合，在测试过程中引入反馈机制对不同分区被选概率进行自适应调整；
  - 对于给定的某个分区，测试用例的选取则是随机的。
- **开发了两种适应性分区测试算法**
  - 基于马尔可夫链的自适应分区测试算法(MAPT)
  - 基于奖惩机制的自适应分区测试(RAPT)
- **经验研究**
  - 采用三个大型开源程序的错误版本评估了所提算法的性能。

- C. Sun, H. Dai, H. Liu, T.Y. Chen, K.-Y. Cai. Adaptive Partition Testing. *IEEE Transactions on Computers*, 2019, 68(2): 157-169.

第二届蜕变测试研讨会，2019.8.14，北京