软件架构可信设计、度量和验证

李必信
东南大学软件工程研究所
东南大学计算机科学与工程学院
手机/微信 18602506179
Email: bx.li@seu.edu.cn

报告提纲

一. 什么是可信?

二. 什么是可信软件?

三. 什么是可信软件架构?

四. 如何提高软件架构的可信性?

4.1软件架构的可信设计：a priori way

4.2软件架构的可信评估（度量和验证）：a posteriori way

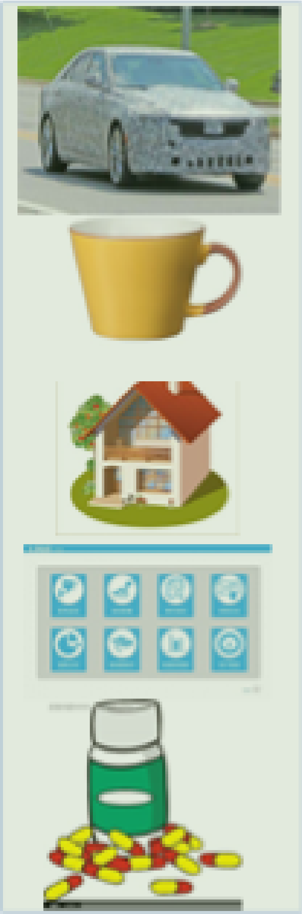
五. 结束语

什么是可信？

◆可信就是：一个实体（例如：汽车、房屋、杯子、药物、软件等）在实现给定目标时，其行为和结果总是可以预期的。

◆但现实生活中，可能是这样的：
- ◆食品安全、假药假医、偷工减料、以次充好、欺上瞒下、贿赂选举...
- ◆软件故障导致的飞机坠毁、火箭爆炸、火车相撞、......

例子观察



您买的轿车，可能是二手改装车，车子中有可能被植入了窃听器；虚夸宣传、以次充好等。

杯子是用来喝水的，功能需求的满足没有问题，质量也不错、摔不烂。但是无良厂家为了节省成本，烧制过程加了某种有害健康的物质。人们在使用该类杯子喝水时可能给身体带来伤害。

您买的房子可能使用劣质材料、有害材料等，存在产权纠纷等问题。

您买的系统或者服务存在漏洞、存在木马、存在信息泄漏风险、存在知识产权纠纷等。

假药、有毒药；有害食品。

…… ……　　**但是，这些事，你事先都不知道，没有人告诉你！**

什么是可信软件？

◆如果某个软件提供的服务（功能）*总是*与用户的预期相符，即使在运行过程中出现一些*特殊情况*也是如此，这样的软件就是可信软件。*特殊情况*包括：

- ◆ 硬件环境（计算机、网络）发生故障
- ◆ 低层软件（操作系统、数据库）出现错误
- ◆ 其它软件（病毒软件、流氓软件）对其产生影响
- ◆ 出现有意（攻击）、无意（误操作）的错误操作

什么是可信软件？

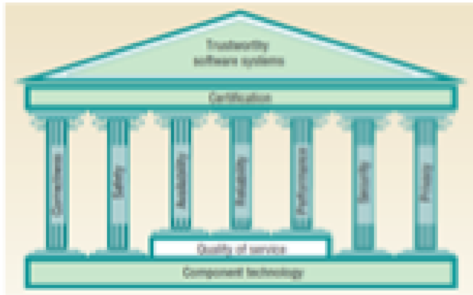
- 具体来讲，**可信软件**就应该是：
  （1）可用（可用性高）
  （2）功能：正确、不少、不多
  （3）可靠性（容错）：高
  （4）安全性（机密性、完整性）：高
  （5）响应时间（从输入到输出）：小
  （6）资源消耗/占用：低
  （7）维护费用（监测、演化）：小
  （8）其他

- 可见，要判断软件是否可信，要从多个方面来评估，只有每个方面都满足要求的软件才是**可信软件**（或者说才是**高可信软件！**）。与其他属性相比，软件的*可信性是一种综合属性*。
- **例如：软件可靠性**是指在给定时间内，特定环境下软件无错运行的概率。软件可靠性包含了以下三个要素：
  （1）规定的时间；
  （2）规定的环境条件；
  （3）规定的功能。

Toward Trustworthy Software Systems

Wilhelm Hasselbring, University of Oldenburg; **Ralf Reussner**, University of Karlsruhe 2006 Security

- Software trustworthiness consists of several attributes
  - *Correctness* refers to the absence of improper system states.
  - *Safety* indicates the absence of catastrophic environmental consequences.
  - *Quality of service* includes three quantifiable attributes:
    - *Availability*—probability of readiness for correct service.
    - *Reliability*—probability of correct service for a given duration of time.
    - *Performance*—response time and throughput.
  - *Security* refers to the absence of unauthorized access to a system.
  - *Privacy* indicates the absence of unauthorized disclosure of information.



TrustSoft "research building"

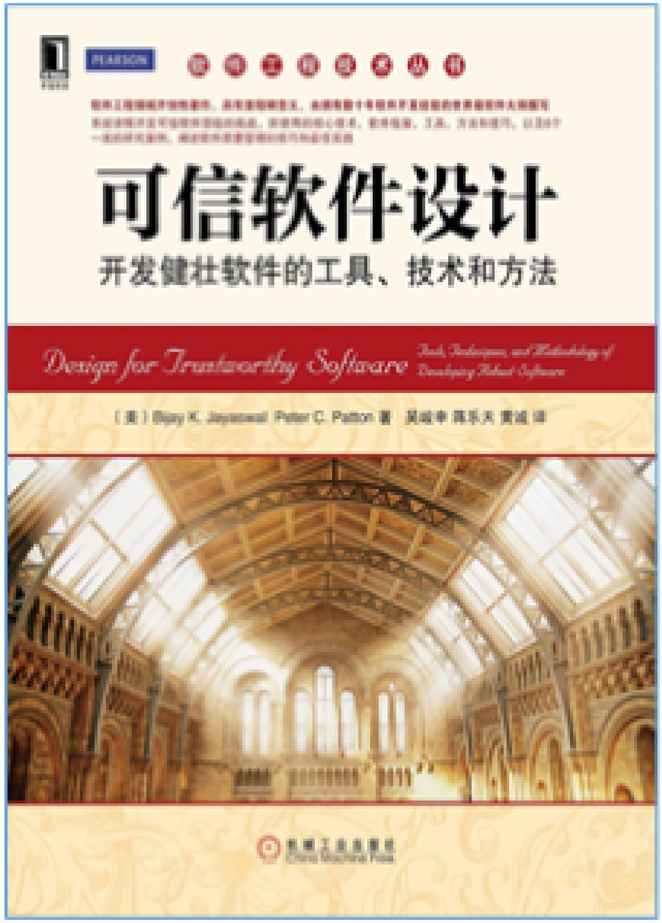
ATTRIBUTE RELATIONSHIPS:

These quality attributes can have two basic types of relationships.

(1) *Intrinsic* relationships exist if one attribute affects another.

(2) *Extrinsic* relationships occur when attributes behave in an opposing way.

可信软件设计



《可信软件设计》是2013年机械工业出版社出版的图书，作者是贾亚斯瓦和巴顿(Bijay K. Jayaswal and Peter C.Patton)

◆本书主要内容：

- 计划、构建、维护和改进可信软件开发系统。
- 在独一无二的软件开发环境中，运用质量、领导力、学习和管理的最佳实践。
- 倾听客户心声，引导用户期望，开发出易用和可靠的软件产品。
- 重点关注可靠性、可信任性、可用性和可升级性等以客户为中心的问题。
- 激励员工拥有强大的设计创意和创新力。
- 确认、验证、评估、集成和维护可信软件。
- 分析软件质量的经济成本影响。
- 为实施DFTS（Design for Trustworthy Software）培养你的领导力和企业内部结构管理能力。

三.什么是可信架构?

什么时软件架构?

◆A critical issue in the design and construction of any complex software system is its architecture: that is, *its gross organization as a collection of interacting components*. A good architecture can help ensure that a system will satisfy key requirements in such areas as **performance, reliability, portability, scalability, and interoperability**. A bad architecture can be disastrous. [David Garlan 2000]

◆Software architecture typically plays a key role as a bridge between requirements and implementation

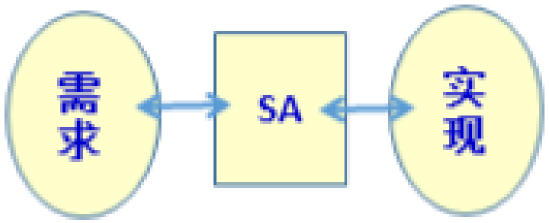


Figure 1: Software Architecture as a Bridge

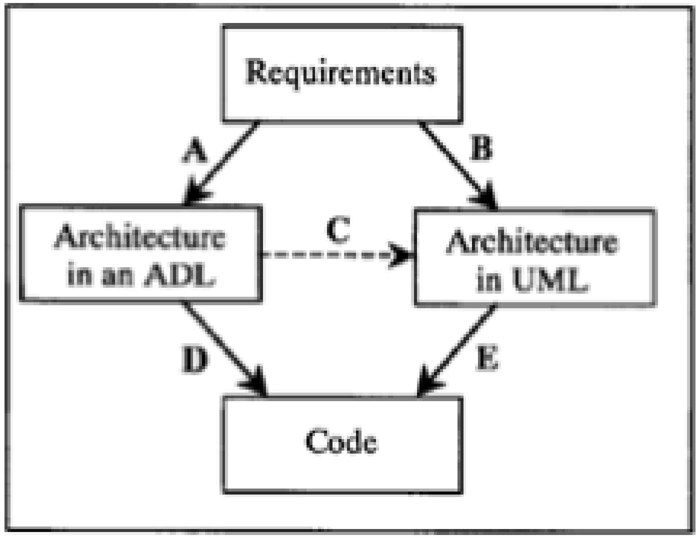


Figure 2: Software Architecture as a Bridge

Path A-D is one in which an ADL is used as the modeling language. Path B-E is one in which UML is used as the modeling notation. Path A-C-E, is one in which an architecture is first represented in an ADL, but then transformed into UML before producing an implementation.

软件架构与可信的关系

经验分析

	可用	功能	可靠性	安全性	性能	维护费用	其他
用户需求		√√√√√					
软件需求		√√√√√					
架构设计	**√√√**	**√√√**	**√√√√√**	**√√√**	**√√√√√**	**√√√√√**	**√√**
模块设计	√√√	√√√√√		√√√			
接口设计	√√√	√√√		√√√	√	√	
数据库设计	√√√	√√√		√√√			
算法设计	√√√	√√√			√√		
代码	√√√√	√√√	√√√√	√√√√	√√	√√√√	√√√√
人员	**√√√√√**	**√√√√√**	**√√√√√**	**√√√√√**	**√√√√√**	**√√√√√**	**√√√√√**

什么是可信架构?

- **software architecture**=(components, connectors, configurations, constraints)【这是4C模型, 或者**software architecture**=(components, compositions), 2C模型】
  - Components: computation units
  - Connectors: some relationships(composition, association, dependence…)
  - Configurations and constraints: decisions by people or organization
- **trusted software architecture**= (trusted components, trusted connectors, trusted configurations, trusted constraints)

如何提高架构的可信性？ ---两条腿走路

4.1 软件架构的可信设计： a priori way

4.2 软件架构的可信评估： a posteriori way

4.1 软件架构的可信设计

a priori way

◆先验方法（**a priori way** ）：主要是在软件架构设计和构造过程中采用一些提高软件可信性的方法，又称之为主动可信性提升方法(proactive trust increasing)。例如，在软件架构设计过程中使用可信基、可信组件、信誉好的人等。

- ◆**可信基**(trusted base)：The key idea is to localize the system's most critical requirements into small, reliable parts called trusted bases.
- ◆**可信计算基**(trusted computing base)：Design a system's security architecture based on a small, precisely defined, and application-specific trusted computing base.
- ◆**可信组件**(trusted component)：Define a formal framework for component composition, replacement, refinement in software design.

- **可信基(trusted base)**: The key idea is to localize the system's most critical requirements into small, reliable parts called trusted bases. For example,
  - a microkernel-based OS is designed so that only a small, trusted core of the system is responsible for ensuring its safety and security; an erroneous program in the user space may hinder normal system operations, but should not be able to crash the entire OS. [Tanenbaum, A. S., Herder, J. N., And Bos, H. 2006. Can We Make Operating Systems Reliable and Secure? IEEE Computer 39, 5, 44–51. 阿姆斯特丹自由大学vu]
  - Similarly, it may be desirable to design an electronic voting system so that its vote-tallying software, which is known to be susceptible to various security attacks, cannot compromise election integrity. Instead, third-party auditors are entrusted with examining the election process to ensure the correctness of the election outcome [Rivest, R. L. and Wack, J. P. 2008. On the Notion of "Software Independence" in Voting Systems. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 366, 1881, 3759 MIT.].
- 參考文献

- Kang, E. and Jackson, D. Patterns for Building Dependable Systems with Trusted Bases. In: Proceedings of the 17th Conference on Pattern Languages of Programs(PLOP 2010), MIT.

Key ideas:
(1) Property-Part Diagram
(2) End-to-End Check
(3) Trusted Kernel

无法显示该图片。

Property-part diagram for an input-output system. A box represents a part, a circle a property, an arrowed edge a dependency of a property on a part or another property, and a straight edge an interaction between two parts.

◆**可信计算基(trusted computing base)**: The method aims at designing a system's security architecture based on a small, precisely defined, and application-specific trusted computing base(TCB). The approach is based on the idea of replacing current off-the-shelf, large-scale, general-purpose TCBs by *tailored, application-specific* TCB's with a reduced functionality that is directly derived from the system's security policies.

◆Figure 1 shows an activity diagram illustrating the sequence of design steps for one iteration of the process.

- ◆First of all, there must be a requirements analysis for a project, wherein the requirements are refined and prioritized.
- ◆Based on it, the security policy with its rules for example for authentication and authorization is defined.
- ◆Once the policy is set up, the further steps security modelling, decomposition, usage-driven refinement, and derivation of security architecture can be performed.
- ◆The methodical way finally concludes in the implementation and validation of the security properties and architecture.
- ◆These steps are the important design activities for dealing with the nonfunctional goal security.

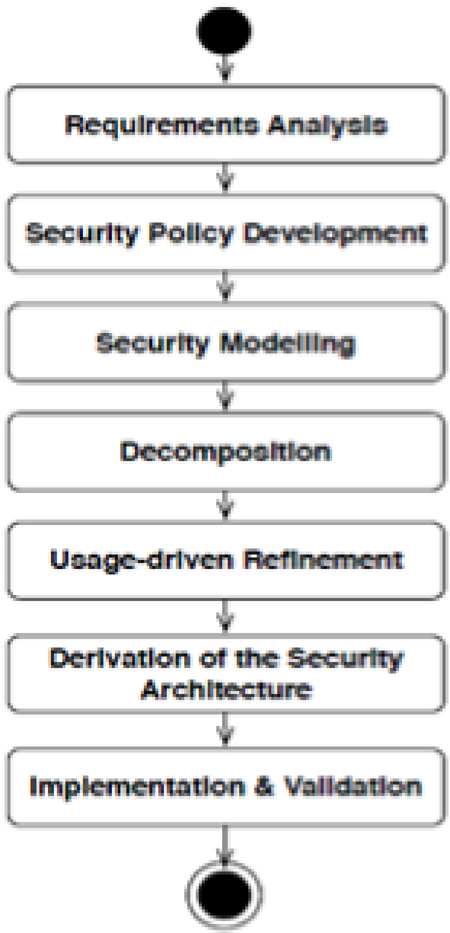


Figure 1. Activities of one iteration of the security architecting process.

◆参考文献

- ◆Stephan Bode, Anja Fischer, Winfried Kühnhauser, and Matthias Riebisch. Software *Architectural Design meets Security Engineering*. 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2009.德国伊尔梅瑙科技大学.

- **可信组件和可信组合 (trusted component/ composition)**
  - Define a formal framework for component **composition, replacement, refinement** in software design.
  - Wrong design decisions, errors, and inconsistencies can be detected early in the development process.
  - Design patterns, which document expert design experience in different applications, have been packaged as design components.
- 参考文献
  - Zheng Yan. A Comprehensive Trust Model for Component Software. SecPerU'08, July 7, 2008, Sorrento, Italy.

The system entities can be any parties that are involved into or related to the component software system. They can be related with each other in order to provide some services or functionalities. They can also be related via certain **trust relationships**. These entities include *a system user*, *a component provider*, *a service*, *a component* (composition of components), *an application*, *a sub-system* and *the whole system*. The system entity can play as either the trustor or the trustee.

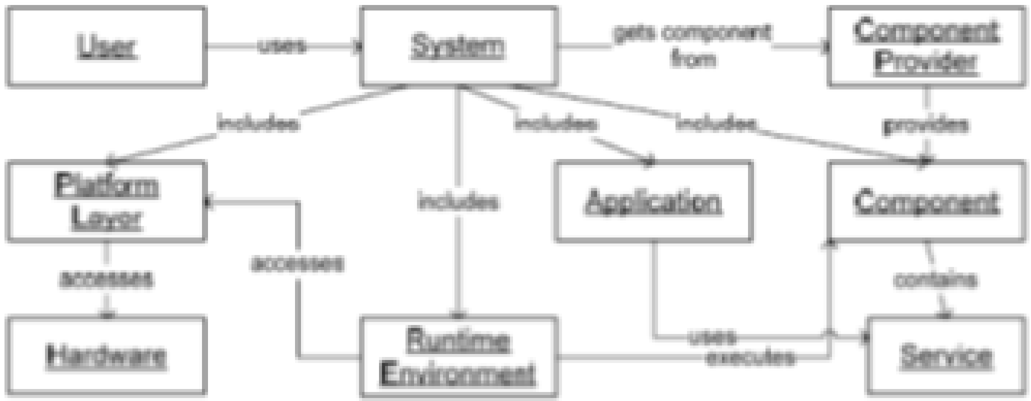


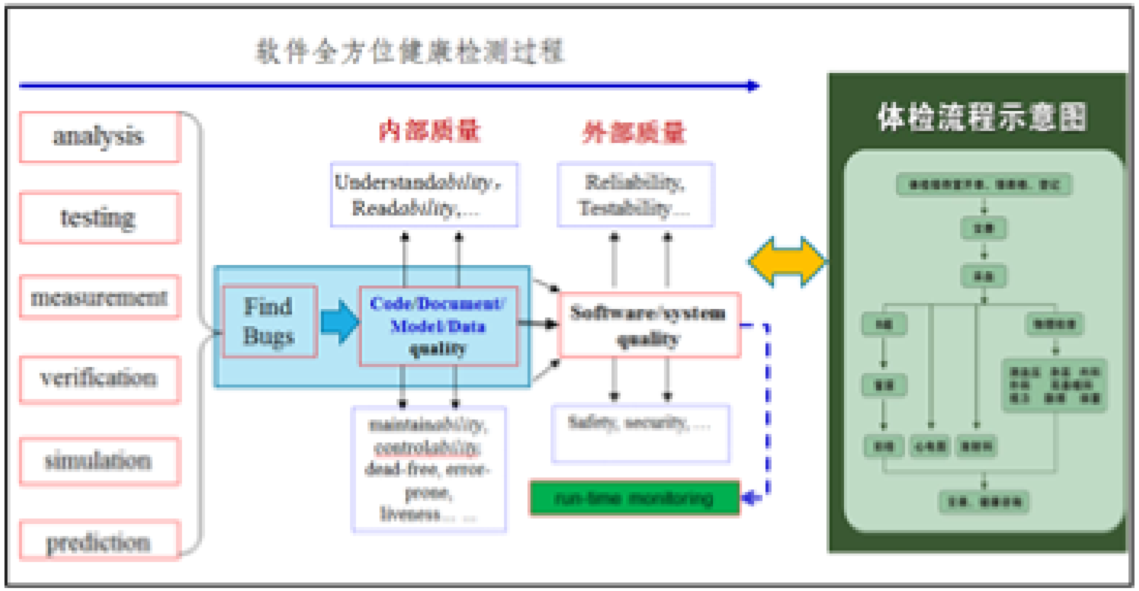
Figure 1: Relationships of component software system entities.

4.2 软件架构的可信评估

a posteriori way

◆**后验方法（a posteriori way）：**软件的初始架构设计完成之后，无论是否按照可信架构的设计思路和设计方法，还是其他的设计，都需要进行架构的可信评估。**后验方法是一种被动可信性提升方法**(reactive trust increasing)。软件架构的可信评估可以根据评估需求选择不同的评估技术。这点和软件全方位的健康检测过程、人的体检过程等类似（如下图）。

- ◆基于度量的评估
- ◆基于验证的评估
- ◆基于分析的评估
- ◆基于测试的评估
- ◆基于仿真的评估
- ◆基于监控的评估
- ◆基于预测的评估
- ◆基于推理的评估
- ◆……



◆可信软件

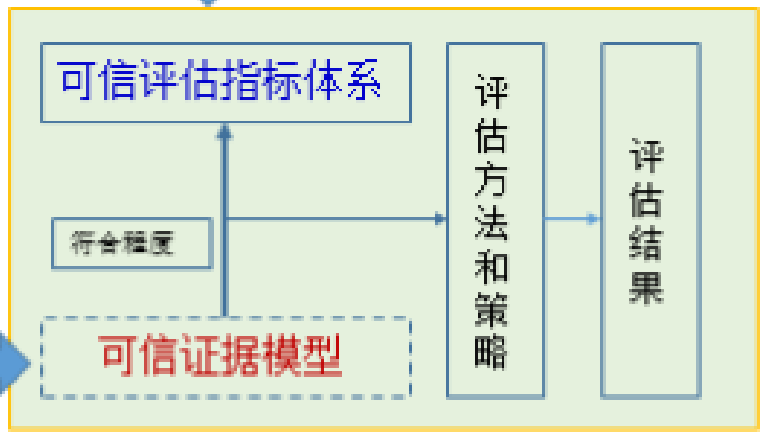
- 可用（可用性高）
- 功能：正确、不少、不多
- 可靠性（容错）：高
- 安全性（机密性、完整性）：高
- 响应时间（从输入到输出）：小
- 资源消耗/占用：低
- 维护费用（监测、演化）：小
- 其他

软件可信性（指标）

可用性	可靠性	安全性	性能	可维护性	可生存性
功能符合性 正确性 易理解性 易操作性 适应性 易安装性	成熟性 容错性 可靠度 失效率 失效强度 MTTF/MTTR/MTBF	数据保密性 代码安全性 控制严密性	响应时间 吞吐量 资源消耗	易修改性 可移植性 稳定性 易测试性 依从性	抗攻击性 攻击识别性 易恢复性 自我完善性

软件可信证据

开发阶段证据	提交阶段证据	应用阶段证据
需求阶段证据 设计阶段证据 编码阶段证据 测试阶段证据 过程审计证据	可用性证据 可靠性证据 安全性证据 性能证据 可维护性证据 可生存性证据	应用规模 用户满意度 应用案例 第三方评估



（1）软件架构基本度量和评估

◆静态度量
- 入度、出度
- 耦合、内聚
- 复杂性：圈复杂性、Halstead复杂性、认知复杂性
- 属性距离、物理距离
- 静态成熟度
- ……

◆动态度量
- 可靠性
- 性能
- 出错率
- **动态成熟度**
- ……

◆综合度量
- 综合成熟度
- ……

第三方用户评价

（2）软件架构演化度量和评估

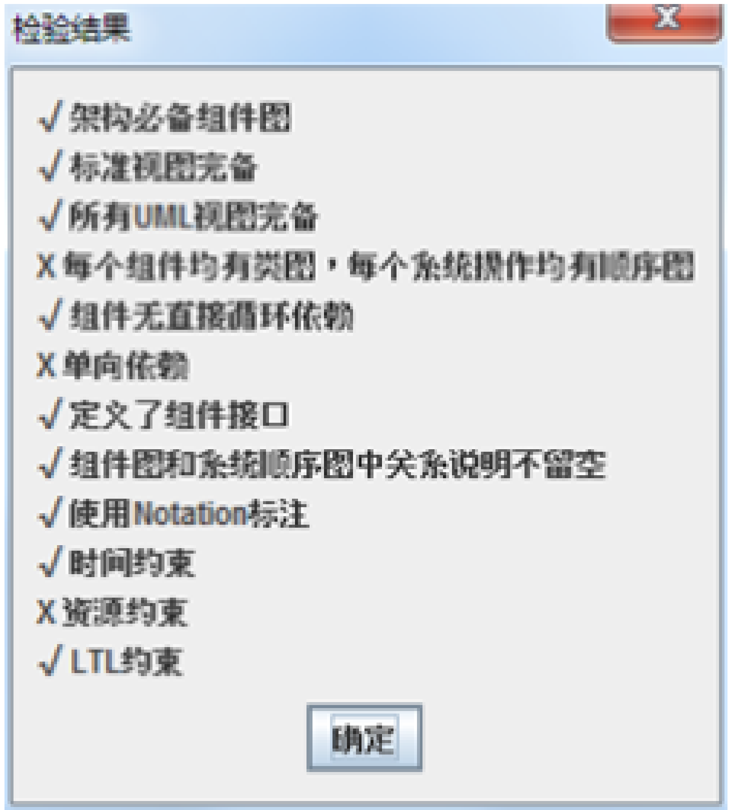
◆可演进性度量
- 易理解性
- 可修改性
- 可替换性
- 可扩展性
- 易测试性
- ……

$\left.\right\}$ **开发/提交阶段证据**

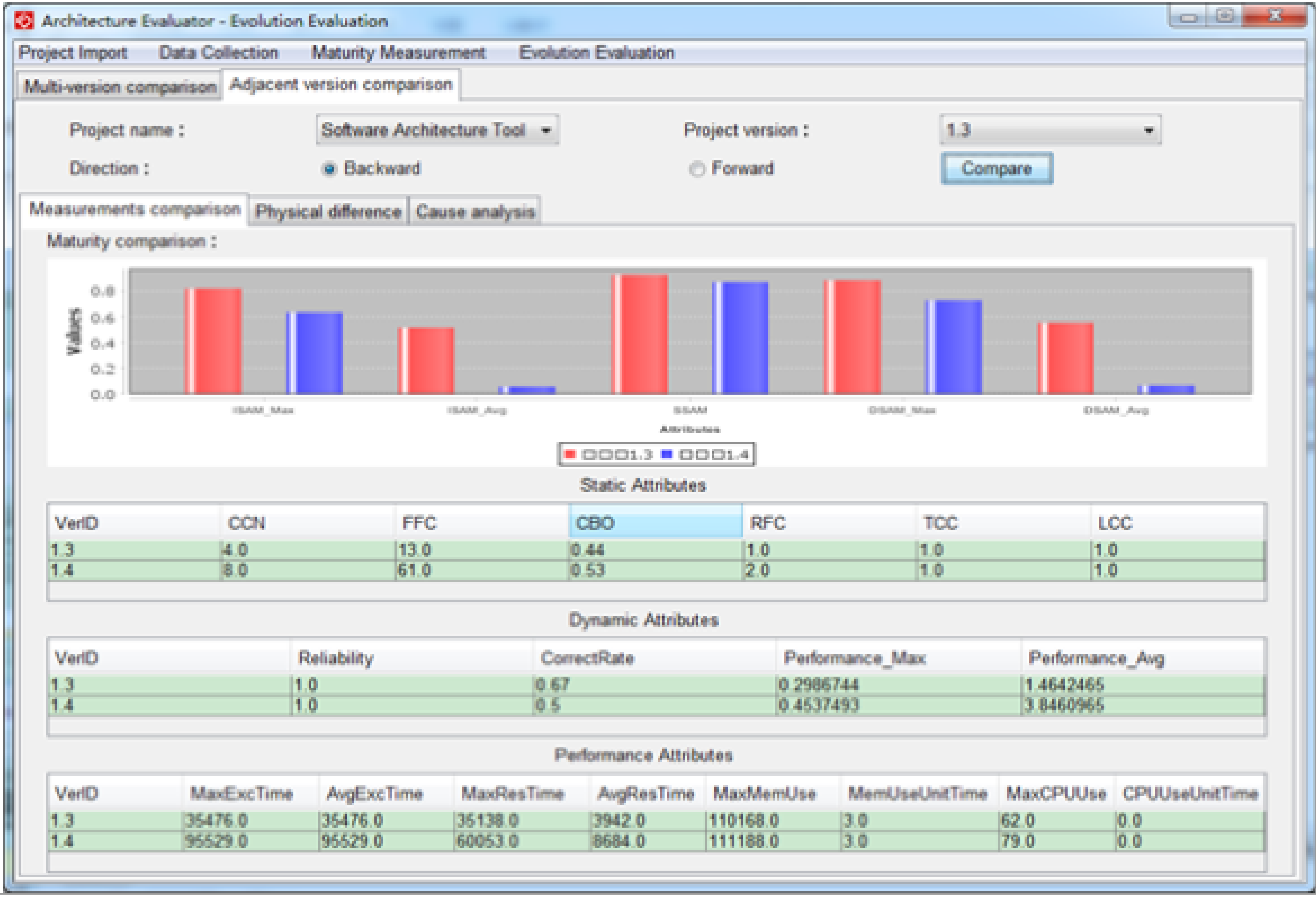
◆演化原则达成性度量
- 平滑演进原则
- 组件最小化原则
- 主体维持原则
- 模块独立演进原则
- 外部接口稳定原则
- 复杂性可控原则
- ……

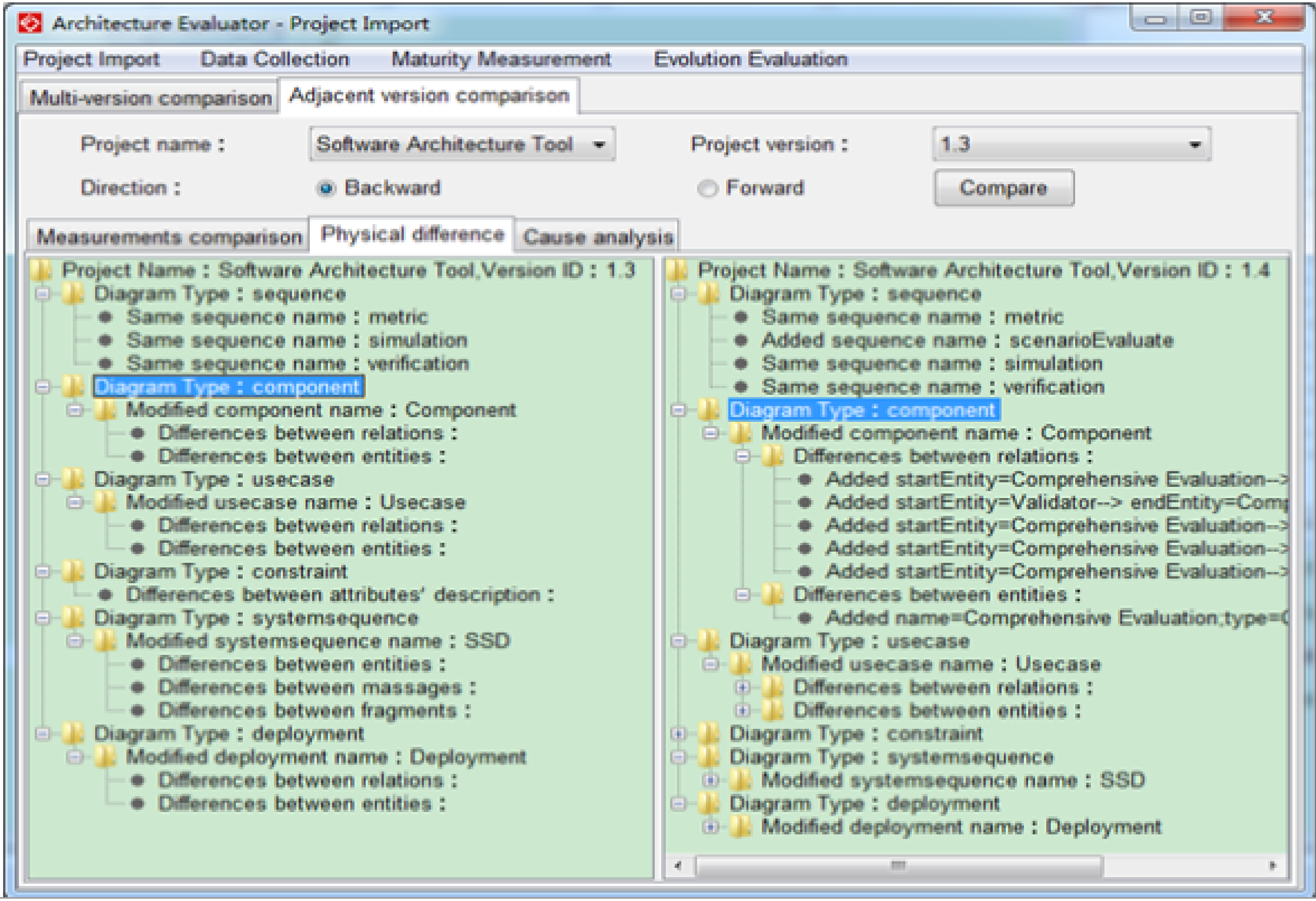
$\left.\right\}$ **开发阶段证据**

4.2.1 自顶向下(TOP-DOWN WAY)：从需求规约中构建软件架构



评估采用的技术：度量、仿真、形式化验证


Architecture Evaluator - Evolution Evaluation
Project Import
Data Collection
Maturity Measurement
Evolution Evaluation
Multi-version comparison
Adjacent version comparison
Project name :
Software Architecture Tool
Project version :
1.3
Direction :
Backward
Forward
Compare
Measurements comparison
Physical difference
Cause analysis
Maturity comparison :
Values
0.8
0.6
0.4
0.2
0.0
ISAM_Max
ISAM_Avg
SSAM
DSAM_Max
DSAM_Avg
Attributes
Static Attributes
VerID CCN FFC CBO RFC TCC LCC
1.3 4.0 13.0 0.44 1.0 1.0 1.0
1.4 8.0 61.0 0.53 2.0 1.0 1.0
Dynamic Attributes
VerID Reliability CorrectRate Performance_Max Performance_Avg
1.3 1.0 0.67 0.2986744 1.4642465
1.4 1.0 0.5 0.4537493 3.8460965
Performance Attributes
VerID MaxExcTime AvgExcTime MaxResTime AvgResTime MaxMemUse MemUseUnitTime MaxCPUUse CPUUseUnitTime
1.3 35476.0 35476.0 35138.0 3942.0 110168.0 3.0 62.0 0.0
1.4 95529.0 95529.0 60053.0 8684.0 111188.0 3.0 79.0 0.0


Architecture Evaluator - Project Import
Project Import
Data Collection
Maturity Measurement
Evolution Evaluation
Multi-version comparison
Adjacent version comparison
Project name :
Software Architecture Tool
Project version :
1.3
Direction :
Backward
Forward
Compare
Measurements comparison
Physical difference
Cause analysis
Project Name : Software Architecture Tool,Version ID : 1.3
Diagram Type : sequence
Same sequence name : metric
Same sequence name : simulation
Same sequence name : verification
Diagram Type : component
Modified component name : Component
Differences between relations :
Differences between entities :
Diagram Type : usecase
Modified usecase name : Usecase
Differences between relations :
Differences between entities :
Diagram Type : constraint
Differences between attributes' description :
Diagram Type : systemsequence
Modified systemsequence name : SSD
Differences between entities :
Differences between massages :
Differences between fragments :
Diagram Type : deployment
Modified deployment name : Deployment
Differences between relations :
Differences between entities :
Project Name : Software Architecture Tool,Version ID : 1.4
Diagram Type : sequence
Same sequence name : metric
Added sequence name : scenarioEvaluate
Same sequence name : simulation
Same sequence name : verification
Diagram Type : component
Modified component name : Component
Differences between relations :
Added startEntity=Comprehensive Evaluation-->
Added startEntity=Validator--> endEntity=Comp
Added startEntity=Comprehensive Evaluation-->
Added startEntity=Comprehensive Evaluation-->
Added startEntity=Comprehensive Evaluation-->
Differences between entities :
Added name=Comprehensive Evaluation,type=(
Diagram Type : usecase
Modified usecase name : Usecase
Differences between relations :
Differences between entities :
Diagram Type : constraint
Diagram Type : systemsequence
Modified systemsequence name : SSD
Diagram Type : deployment
Modified deployment name : Deployment

动态成熟度的定义

$$M_{动} = \begin{cases} 0.95^{Performance} & ,CorrectRate * Reliability = 1 \\ (CorrectRate * Reliability)^{Performance} & ,CorrectRate * Reliability \neq 1 \end{cases}$$

$$CorrectRate = \frac{正确功能数目}{总功能数目}$$

$$Performance = \begin{cases} (W_1 * \frac{AvgExcTime}{EpAET} + W_2 * \frac{AvgResTime}{EpART} + W_3 * \frac{MemUseUnitTime}{EpMUUT} \\ \quad + W_4 * \frac{CPUUseUnitTime}{EpCUUT})/4 \\ (W_1 * \frac{MaxExcTime}{EpMET} + W_2 * \frac{MaxResTime}{EpMRT} + W_3 * \frac{MaxMemUse}{EpMMU} \\ \quad + W_4 * \frac{MaxCPUUse}{EpMCU})/4 \end{cases}$$

$$Reliability = \sum_{i=1}^{m}\left(ActorEP_i * \sum_{j=1}^{n}\left(ucEP_j * \frac{1}{k} * \sum_{1}^{k}\left(\prod_p component_p^{x_p} * \prod_q connector_q^{y_q}\right)\right)\right)$$

仿真结果

	最大运行时间	平均运行时间	最大响应时间	平均响应时间
架构时间仿真结果：	95523.0	83020.3	60050.0	9224.48

	最大内存利用量	单位时间内存利用量	最大CPU占用量	单位时间CPU占用量
架构资源仿真结果：	111508.0	3.2	62.0	0.0

各个顺序图仿真结果：

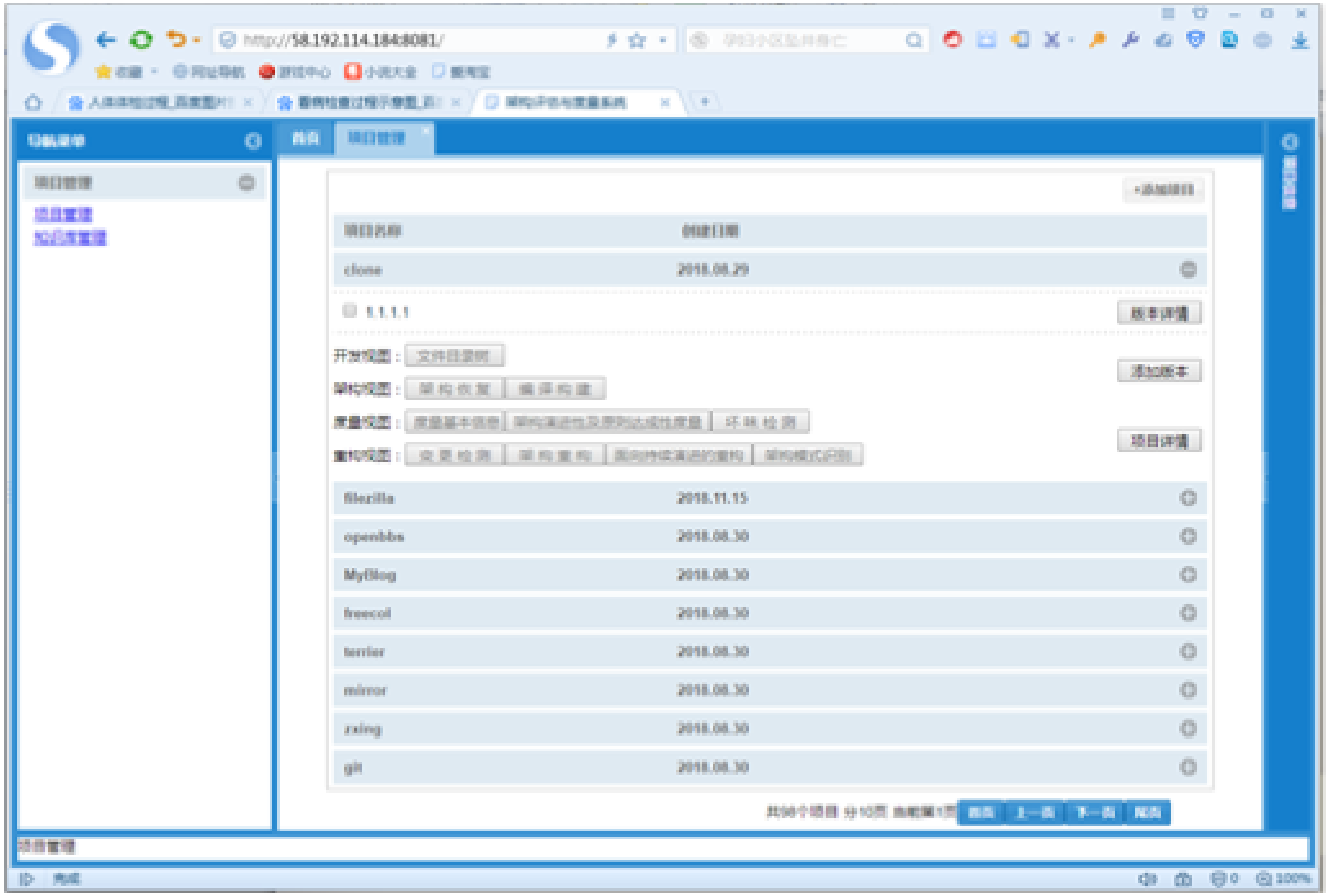
顺序图时间仿真	verification	metric	scenarioEvaluate	simulation
最大运行时间：	35138.0	28.0	60050.0	307.0
平均运行时间：	35138.0	28.0	47551.5	302.8
最大响应时间：	35000.0	10.0	35010.0	20.0
平均响应时间：	2928.17	2.33	2971.97	9.9
时间消耗最多模块名：	CallSpin	MaintainabilityAnalysis	callSubsystem	returnSDResourceSimulati...
模块消耗最大时间所占比例：	1.0	0.36	0.74	0.33

顺序图资源仿真	verification	metric	scenarioEvaluate	simulation
累计内存利用量：	110486.0	28.0	155054.5	135.5
内存利用平衡率：	742773277.53	8.61	622248566.75	108.39
最大内存利用量：	110108.0	15.0	111018.0	53.0
影响最大内存利用的步骤：	CallSpin	MaintainabilityAnalysis	callSubsystem	returnSSDResourceSimul...
单位时间内存利用量：	3.14	1.0	3.26	0.45
累计CPU占用量(%)：	92.9	7.1	92.95	20.95
CPU占用平衡率(%)：	177.75	2.0	221.05	8.88
最大CPU占用量(%)：	60.0	4.0	62.0	2.0
影响最大CPU占用的步骤(%...	CallSpin	MaintainabilityAnalysis	callSubsystem	returnSDResourceSimulati...
单位时间CPU占用量(%)：	0.0	0.25	0.0	0.07

图simulation资源仿真中最大资源消耗的详细执行信息：

内存仿真步骤名	步骤开始需要内存	步骤结束释放内存
startSDSimulation	5.0	3.0
returnSDTimeSimulationResult	6.0	4.0
returnSDResourceSimulationResult	10.0	6.0
startSDSimulation	5.0	3.0
returnSDTimeSimulationResult	6.0	4.0
returnSDResourceSimulationResult	10.0	6.0

4.2.2 自底向上(BOTTOM-UP WAY)：从代码恢复软件架构



```
Never claim moves to line 5     [(1)]
proc  1 = Actor
proc  2 = TaskNode1
proc  3 = TaskNode2
proc  4 = TaskNode3
proc  5 = JobClient
proc  6 = HDFS
proc  7 = JobTracker
q\p   0   1   2   3   4   5   6   7
  1   .   c_InputJob!InputJobm
  1   .   .   .   .   .   c_InputJob?InputJobm
  2   .   .   .   .   .   c_SubmitJob!SubmitJobm
  2   .   .   .   .   .   .   .   c_SubmitJob?SubmitJobm
  3   .   .   .   .   .   .   .   c_JobInfo!JobInfom
  4   .   .   .   .   .   .   .   c_LaunchTask1!LaunchTask1m
  5   .   .   .   .   .   .   .   c_LaunchTask2!LaunchTask2m
  3   .   .   .   .   .   c_JobInfo?JobInfom
  6   .   .   .   .   .   c_SaveJob!SaveJobm
  6   .   .   .   .   .   .   c_SaveJob?SaveJobm
  5   .   .   .   c_LaunchTask2?LaunchTask2m
  7   .   .   .   c_LoadJob2!LoadJob2m
  4   .   .   c_LaunchTask1?LaunchTask1m
  8   .   .   c_LoadJob1!LoadJob1m
  8   .   .   .   .   .   .   c_LoadJob1?LoadJob1m
  9   .   .   .   .   .   .   c_GetJob1!GetJob1m
  9   .   .   c_GetJob1?GetJob1m
 10   .   .   c_OutputResult1!OutputResult1m
 10   .   .   .   .   .   .   c_OutputResult1?OutputResult1m
  7   .   .   .   .   .   .   c_LoadJob2?LoadJob2m
 11   .   .   .   .   .   .   c_GetJob2!GetJob2m
 11   .   .   .   c_GetJob2?GetJob2m
 12   .   .   .   c_MapTask!MapTaskm
 12   .   .   .   c_MapTask?MapTaskm
 13   .   .   .   c_OutputResult2!OutputResult2m
 13   .   .   .   .   .   .   c_OutputResult2?OutputResult2m
 14   .   .   .   c_ReduceTask!ReduceTaskm
 14   .   .   .   .   c_ReduceTask?ReduceTaskm
 15   .   .   .   .   c_OutputResult3!OutputResult3m
 15   .   .   .   .   .   .   c_OutputResult3?OutputResult3m
 16   .   .   .   .   .   .   c_OutputData!OutputDatam
 16   .   .   .   .   .   c_OutputData?OutputDatam
spin: _spin_nvr.tmp:3, Error: assertion violated
spin: text of failed assertion: assert(!(<>(!(<>(LaunchTask1&&LaunchTask2)))&&!(!(
OutputData)))))
```

```
proc 0 = :init:
using statement merging
proc 1 = ResourceManager
proc 2 = HDFS
proc 3 = JobClient
proc 4 = AppMaster2
proc 5 = TaskNode2
proc 6 = AppMaster1
proc 7 = TaskNode1
proc 8 = Actor
proc 9 = TaskNode3
q\p   0   1   2   3   4   5   6   7   8   9
  1   .   .   .   .   .   .   .   .   c_InputJob!InputJobm
  2   .   .   .   .   .   .   .   c_ApplyForJob1!ApplyForJob1m
  3   .   .   .   .   .   c_ApplyForJob2!ApplyForJob2m
  1   .   .   .   c_InputJob?InputJobm
  4   .   .   .   c_SubmitJob!SubmitJobm
  4   .   c_SubmitJob?SubmitJobm
  5   .   c_JobInfo!JobInfom
  5   .   .   .   c_JobInfo?JobInfom
  6   .   .   .   c_SaveJob!SaveJobm
  6   .   .   c_SaveJob?SaveJobm
  2   .   c_ApplyForJob1?ApplyForJob1m
  3   .   c_ApplyForJob2?ApplyForJob2m
  7   .   c_LaunchTask1!LaunchTask1m
  7   .   .   .   .   .   .   .   c_LaunchTask1?LaunchTask1m
  8   .   .   .   .   .   .   .   c_LoadJob1!LoadJob1m
  8   .   .   c_LoadJob1?LoadJob1m
  9   .   .   c_GetJob1!GetJob1m
  9   .   .   .   .   .   .   .   c_GetJob1?GetJob1m
 10   .   .   .   .   .   .   .   c_startAppMaster1!startAppMaster1m
 10   .   .   .   .   .   .   c_startAppMaster1?startAppMaster1m
 11   .   .   .   .   .   .   c_ReduceNode1Task1!ReduceNode1Task1m
 11   .   .   .   .   .   .   .   c_ReduceNode1Task1?ReduceNode1Task1m
 12   .   .   .   .   .   .   .   c_OutputNode1Result1!OutputNode1Result1m
 13   .   .   .   .   .   .   c_ReduceNode2Task1!ReduceNode2Task1m
 14   .   .   .   .   .   .   c_ReduceNode3Task1!ReduceNode3Task1m
 14   .   .   .   .   .   .   .   .   .   c_ReduceNode3Task1?ReduceNode3Task1m
 15   .   .   .   .   .   .   .   .   .   c_OutputNode3Result1!OutputNode3Result
1m
 13   .   .   .   .   .   c_ReduceNode2Task1?ReduceNode2Task1m
 16   .   .   .   .   .   c_OutputNode2Result1!OutputNode2Result1m
 12   .   .   c_OutputNode1Result1?OutputNode1Result1m
 16   .   .   c_OutputNode2Result1?OutputNode2Result1m
 15   .   .   c_OutputNode3Result1?OutputNode3Result1m
 17   .   .   c_OutputData!OutputDatam
 17   .   .   .   c_OutputData?OutputDatam
 18   .   .   .   c_Result!Resultm
 18   .   .   .   .   .   .   .   .   c_Result?Resultm
spin: trail ends after 122 steps
```

ISEU与软件架构相关成果

- 李必信，廖力，王璐璐，孔祥龙， 周颖 编著. 软件架构理论和实践，机械工业出版社，2019年3月.
- 李必信，樊玉琦，廖力. 一种基于演化的软件架构评估方法。ZL 2016 1 01926069
- 李必信，樊玉琦，廖力.一种架构静态成熟度度量方法。 ZL 2016 1 0112423.1
- 李必信，樊玉琦，廖力.一种架构动态成熟度度量方法。 ZL 201610112152X
- 李必信,张心悦; 王楠; 孔祥龙; 廖力，一种软件架构可持续演进原则达成性度量方法. 申请号：2018102215237
- 熊壬浩; 李必信. 一种基于本体的架构行为模式识别方法. 申请号：2018100502720
- 李必信;陈艺; 司静文; 孔祥龙; 苗意盈. 一种基于原型仿真的架构评价方法. ZL2014102686928
- 李必信; 王楠; 孔祥龙; 张心悦. 一种基于软件架构的可演进性度量方法. 2017105072654
- 熊壬浩; 李必信. 一种基于本体的架构模式建模方法. 2017111434934
- 王璐璐;张叶涛; 孔祥龙; 李必信. 一种基于多规则聚类的组件识别方法. 2018102215256
- 李必信; 司静文; 俞祈聚; 孔祥龙; 樊雨琦. 一种基于度量和预测技术的软件架构 评估方法. ZL 2014102712636
- 李必信; 俞祈聚; 陈艺; 孔祥龙; 王璐璐. 一种基于UML模型的软件架构正确性验证方法. 2014103190169
- Renhao Xiong ; **Bixin Li**. Accurate Design Pattern Detection Based on Idiomatic Implementation Matching in Java Language Context. 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)，2019，pages 4-14.
- Tong Wang ; Dongdong Wang ; Ying Zhou ; **Bixin Li**. Software Multiple-Level Change Detection Based on Two-Step MPAT Matching. 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)，2019，pages 4-14.
- Xianglong Kong, **Bixin Li**, Lulu Wang, Wensheng Wu: Directory-Based Dependency Processing for Software Architecture Recovery. IEEE Access 6: 52321-52335 (2018).
- **Bixin Li**, Li Liao, Ximeng Yu. A Verification-Based Approach to Evaluate Software Architecture Evolution, Chinese Journal of Electronics, 26(3): 485-492 (2017)
- **Bixin Li**, Li Liao, Jingwen Si: A technique to evaluate software evolution based on architecture metric. SERA 2016: 273-280
- **Bixin Li** ; Li Liao ; Yi Cheng. Evaluating Software Architecture Evolution Using Performance Simulation. 2016 4th Intl Conf on Applied Computing and Information Technology，2016，pages 7-13.
- Chuanqi Tao, **Bixin Li**, Jerry Gao: Testing Configurable Architectures for Component-Based Software Using an Incremental Approach. SEKE 2013: 356-361
- Chuanqi Tao, **Bixin Li**, Jerry Gao: Regression Testing of Component-Based Software: A Systematic Practise Based on State Testing. HASE 2011: 29-32
- Pengcheng Zhang, Henry Muccini, **Bixin Li**: A classification and comparison of model checking software architecture techniques. Journal of Systems and Software 83(5): 723-744 (2010)
- **Bixin Li**, Ying Zhou, Yancheng Wang, Junhui Mo: Matrix-based component dependence representation and its applications in software quality assurance. SIGPLAN Notices 40(11): 29-36 (2005)

ISEU与可信相关的成果

- **李必信**, 张新程. 组合服务建模与验证. 科学出版社, 2014.
- **李必信**; 宋锐; 吴晓娜; 刘翠翠; 齐珊珊; 孔祥龙. 一种基于用户诚实度的动态的Web服务信任评估方法. 201210279364
- **李必信**; 刘翠翠; 齐珊珊; 吴晓娜; 宋锐; 刘力. 一种基于数据依赖的组合服务可信性计算方法. 2012102686986
- **李必信**; 谢蓉丽. 一种基于SBG的组合服务可靠性的动态预测方法. 201210209600X
- **李必信**; 齐珊珊; 刘翠翠; 吴晓娜; 宋锐; 李伟. 一种基于XCFG的组合服务可信性演化影响分析方法. 2012102703430
- **李必信**; 吴晓娜; 刘翠翠; 宋锐; 齐珊珊; 李超. 一种基于信任的组合服务优化方法. 2012102098927
- **Bixin Li**, Li Liao, Hareton Leung, Rui Song: PHAT: A Preference and Honesty Aware Trust Model for Web Services. IEEE Trans. Network and Service Management 11(3): 363-375 (2014)
- **Bixin Li**, Shunhui Ji, Dong Qiu, Hareton Leung, Gongyuan Zhang: Verifying the Concurrent Properties in BPEL Based Web Service Composition Process. IEEE Trans. Network and Service Management 10(4): 410-424 (2013)
  Zecheng Li, Li Liao, Hareton Leung, Bixin Li, Chao Li: Evaluating the Credibility of Cloud Services. Computers & Electrical Engineering 58: 161-175 (2017)
- Dong Qiu, **Bixin Li**, Shunhui Ji, Hareton K. N. Leung: Regression Testing of Web Service: A Systematic Mapping Study. ACM Comput. Surv. 47(2): 21:1-21:46 (2014)
- **Bixin Li**, Dong Qiu, Hareton Leung, Di Wang: Automatic Test Case Selection for Regression Testing of Composite Service based on Extensible BPEL Flow Graph. Journal of Systems and Software 85(6): 1300-1324 (2012)
- Li Liao, ShanShan Qi, **Bixin Li**: Trust Analysis of Composite Service Evolution. SERA 2016: 15-22
- Chunli Xie, **Bixin Li**, Hareton Leung: SRM: a Staged Reliability Model for Web Service. ISSE 10(2): 137-154 (2014)
- **Bixin Li**, Shunhui Ji, Li Liao, Dong Qiu, Mingjie Sun: Monitoring Web Services for Conformance. SOSE 2013: 92-102
- **Bixin Li**, Rui Song, Li Liao, Cuicui Liu: A User-Oriented Trust Model for Web Services. SOSE 2013: 224-232
- Rui Song, **Bixin Li**, Xiaona Wu, Cuicui Liu, ShanShan Qi: A Preference and Honesty Aware Trust Model for Web Services. APSEC 2012: 61-66
- Xiaona Wu, **Bixin Li**, Rui Song, Cuicui Liu, ShanShan Qi: Trust-Based Service Composition and Optimization. APSEC 2012: 67-72
- ShanShan Qi, **Bixin Li**, Cuicui Liu, Xiaona Wu, Rui Song: A Trust Impact Analysis Model for Composite Service Evolution. APSEC 2012: 73-78
- Cuicui Liu, **Bixin Li**, ShanShan Qi, Xiaona Wu, Rui Song: Data Depedency Based Trust Evaluation for BPEL Processes. APSEC 2012: 857-866