# 易用性分析

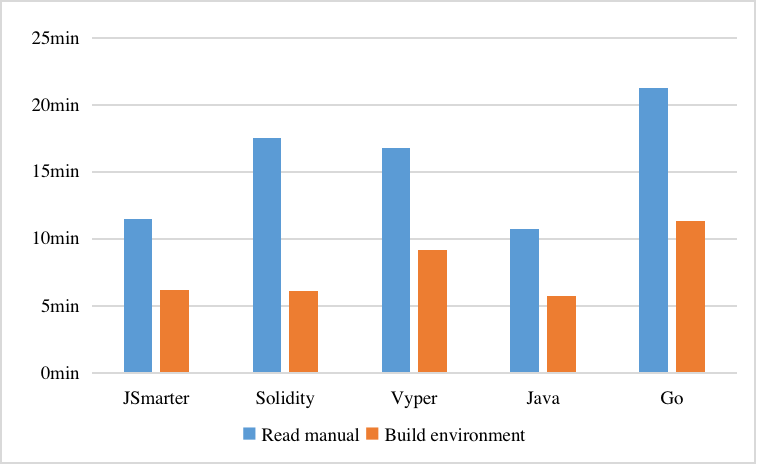
JSmarter的易用性通过完成时长、主观评分和代码量三项指标进行评估。

## 1.完成时长

完成时长由两部分组成，第一是参与者在阅读使用手册和搭建开发环境上所用的时间，第二是参与者编写测试合约的时间。

参与者对于五种语言在阅读使用手册和搭建开发环境上的平均用时如图1所示。参与者对于Java在阅读手册和搭建环境上的用时最短，这一定程度上得益于其本身的用户基础。JSmarter的用时仅次于Java，这从侧面反映了其学习成本较低。且据进一步统计，对于JSmarter而言，P1类型参与者阅读使用手册的平时用时比P2类型参与者短2分钟，搭建开发环境的平时用时短1分钟；对于其他语言而言，P1和P2类型参与者之间的用时差距明显更大。这说明JSmarter在总体学习成本较低的前提下，对智能合约开发的初学者相对更加友好。本工作后续将持续优化使用手册，完善配套生态，进一步降低使用者的入门门槛。

图1：

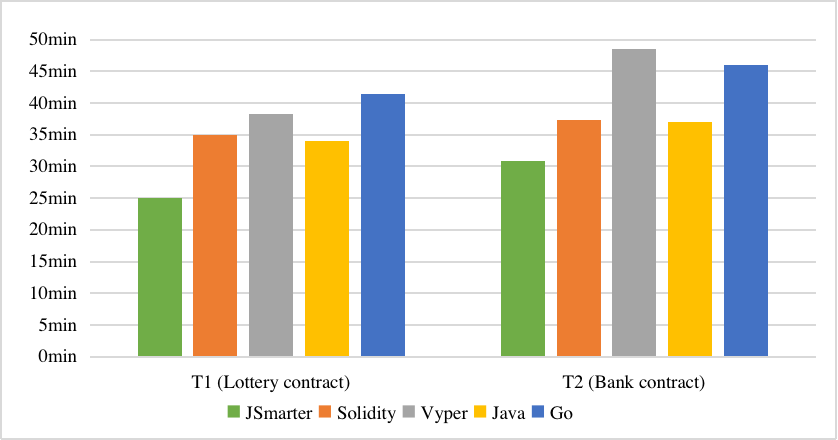


对于一门语言来说，阅读手册和搭建环境的用时属于一次性的指标，因此本实验更关注编写合约的用时。参与者使用五种编程语言编写两个测试合约的平均时间如图2所示。

从图中可以很容易观察到两点。第一，对于任一编程语言，实现合约T2的平均时间都要长于合约T1，这符合对于测试合约复杂度的预估。实际上，T1和T2除了存在复杂度上的区别，还存在应用场景的区别，Solidity和Java可能更适用于T2的应用场景，导致两者的实现时间差异并不显著，但这对实验结果几乎没有影响。第二，对于任一测试合约，JSmarter的平均实现时间在所有语言中均最短，这印证了JSmarter的易用性最高。且据进一步统计，P1类型参与者使用JSmarter编写T1和T2的平时用时比P2类型参与者分别短3分钟和4分钟，即对于具备智能合约开发经验的群体，可以进一步降低约10%的开发时间。

五种语言按照平均实现时间总体可以分为三个层级，JSmarter实现两个测试合约的时间明显均最短，Solidity和Java的实现时间不相上下，Vyper和Go的实现时间相对较长。对于这一结果，本文认为，首先如果排除参与者原本对于语言熟悉程度的干扰，使用通用编程语言开发智能合约的普遍难度理论上应当高于领域特定语言。而Java是最流行的通用编程语言，Solidity是智能合约最流行的领域特定语言，因此，具备相关编程基础的参与者相对偏多，他们的入门成本更低，而Go和Vyper恰恰与之相反。JSmarter则是以Java为宿主语言的内部DSL，既保留了Java庞大用户基础的优势，也融入了便于智能合约开发的语法设计，因此更加贴近开发者的使用习惯。

图2：

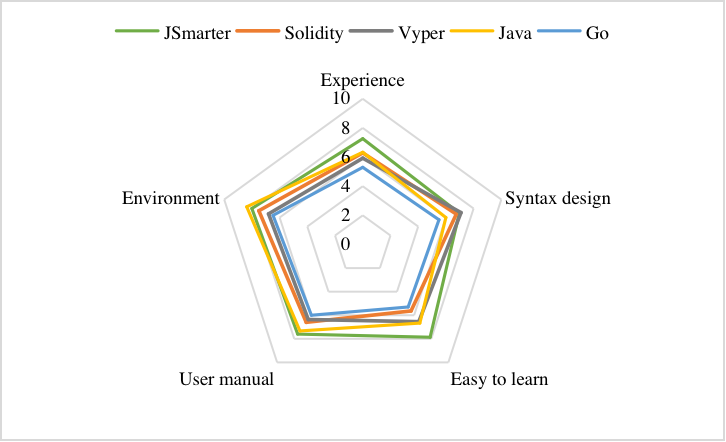


## 2.主观评分

通过合约编写完成后的评价问卷，实验收集了参与者对于所分配到的两种语言在开发体验、语法设计的简洁性和易用性、易学习性、使用手册的详细程度及易理解性、环境搭建的便捷性及易操作性方面的评分。1-2分最低表示非常差，3-4分表示较差，5-6分表示一般，7-8分表示较好，9-10分最高表示非常好。其中，使用手册和环境搭建这两个方面的评分与图1中参与者在相应阶段的用时统计可作为相互辅证。图3展示了五种语言在五个方面的平均评分。对于主观评分指标的分析可以分为两个维度：

1. JSmarter和领域特定语言的对比：总体来说，JSmarter和两门DSL在语法设计的简洁易用方面不相上下，在其他方面JSmarter均相对更优。其中，Vyper在语法设计方面的得分略高于JSmarter，事实上，Vyper确实是一门非常简洁的语言，其语法类似Python。但是从意见反馈中得知，有参与者表示如果没有接触过Python，上手Vyper会比较困难。Solidity在环境搭建的便捷性及易操作性方面和JSmarter差距不大，但需要说明的是，实验为Solidity选择的开发环境是在线编辑器，无需过多的本地环境。JSmarter在开发体验和易学习性方面明显优于Solidity和Vyper，这一定程度上也依赖于其宿主语言的优势。
2. JSmarter和通用编程语言的对比：总体来说，JSmarter和Java在环境搭建和使用手册方面相当，和Go相比在各个方面均显著领先。Java是JSmarter的宿主语言，因此两者在环境搭建上相差不多。但由于Java本身是通用编程语言，引入了很多第三方库来开发智能合约，因此在语法设计、开发体验等方面都逊于JSmarter。对于Go的评分结果，一方面Go作为通用语言不具备智能合约开发的领域特性，Go本身并没有Java那么庞大的用户群体，因此Go在各个方面的评分都明显低于JSmarter。

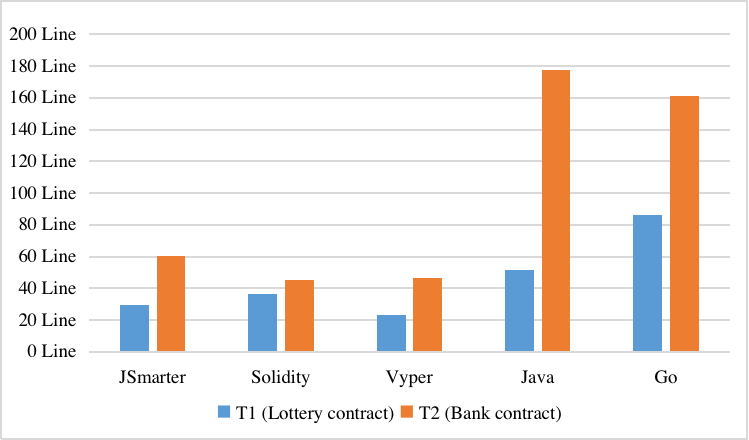
图3：



## 代码量

实验参与者使用五种编程语言实现两个测试合约的平均代码量统计结果如图4所示。且据进一步统计，P1和P2类型参与者之间实现合约的代码量差异并无明显规律，本文推测这更多取决于参与者的个人编程能力与习惯。直观来看，五种语言可以分为两个层级，在合约功能相同的情况下，JSmarter、Solidity和Vyper的代码量显著低于Java和Go，说明JSmarter编写的合约在程序复杂性上和DSL普遍属于一个层级。结合五种语言实现测试合约的平均时间，以及通过对相关合约样本代码的查阅，发现Java和Go合约的代码存在很多相似的重复逻辑，这也验证了通用编程语言在智能合约场景的冗余性。

图4：



# 安全性分析

本文对于实验中参与者实现的所有合约进行了漏洞检测。对于五种语言编写的合约中存在的漏洞类型及数量统计如论文表5所示，其中2/6表示6份合约样本中有2份存在该漏洞。

对于Solidity和Vyper合约，使用Mythril工具进行检测。Mythril是以太坊官方推荐的基于EVM字节码的智能合约安全分析工具，由Bernhard等人研发，集成了污点分析、符号执行等技术，可以检测整数溢出、时间戳依赖、重入等常见安全漏洞。经检测，在Solidity编写的6份T1合约中，均存在时间戳依赖漏洞，这来源于合约中随机数的生成依赖了区块时间戳；其中2份合约存在重入漏洞，该问题在Solidity中很容易产生，因为转账后总是会触发外部函数调用，从而导致存在重入调用的风险。在Solidity编写的6份T2合约中，有3份合约检测出重入漏洞。在Vyper编写的6份T1合约中，有2份检测出时间戳依赖漏洞，其他4份合约中的随机数通过random库实现，虽然没有依赖区块时间戳，但生成的仍是伪随机数，因此存在伪随机漏洞。在Vyper编写的6份T2合约中，没有检测出漏洞。

对于Go合约，使用HFCCT工具进行检测。HFCCT是一种将动态符号执行和静态抽象语法树分析技术相结合的HF平台Go链码漏洞检测框架，由本文作者研发，用于检测来自语言指令和外部访问的不确定性等多种漏洞类型。经检测，在Go编写的6份T1合约中，有2份存在链码字段声明漏洞，3份存在未检查的错误漏洞，所有合约均存在时间戳依赖漏洞。在Go编写的6份T2合约中，有3份存在链码字段声明漏洞，4份存在未检查的错误漏洞。对于上述各漏洞在实验样本中产生来源的概述如下：(1)链码字段声明漏洞：样本中的链码结构体存在字段声明，可以在Init和Invoke方法中访问该字段，由于HF区块链中每个节点不一定执行每笔交易，因此该字段值可能在各节点之间不一致。(2)未检查的错误漏洞：样本中存在通过赋值给“\\_”变量跳过对错误类型返回值的接收，从而可能会忽略错误的发生。(3)时间戳依赖漏洞：类似的，源自通过时间戳来实现随机数生成。

本文在学术界和工业界暂未搜集到漏洞种类覆盖面广且适用于Java智能合约的漏洞检测开源工具。因此，由两位具备智能合约漏洞研究基础的技术人员根据在前三种语言实现的合约中发现的漏洞类型及相关语句，对Java合约进行人工校验。经检测，在Java编写的6份T1合约中，均存在伪随机漏洞，其来源于Java的伪随机数生成器。在Java编写的6份T2合约中，未发现漏洞。

对于本文提出的JSmarter语言编写的合约，由于暂时没有合适的检测工具，本文尝试通过代码走查和理论论证相结合的方式进行人工校验。对于时间戳依赖漏洞，JSmarter将获取系统时间的相关函数作为黑名单，非必要不允许使用，否则无法通过预编译，同时在48份JSmarter合约样本中确实未发现使用。对于伪随机漏洞，所有T1合约样本均通过Java提供的Random类或Math.random()方法生成随机数，而JSmarter在预编译时将它们替换成真正的随机数生成器SecureRandom类及其方法。对于重入漏洞，JSmarter在预编译时引入了重入锁机制，避免函数重入调用，虽然只有14份JSmarter合约样本中使用了@NonReentrant注解对涉及账本操作的函数嵌入了这一机制，但其他样本经代码检查也未检测到重入漏洞。对于链码字段声明漏洞，实际上这一漏洞仅针对Go链码，在JSmarter中本身就不存在。对于未检查的错误漏洞，一方面，JSmarter及其宿主语言Java中没有类似Go中忽略处理函数返回值的机制，另一方面，该漏洞主要针对读取账本的返回值可能存在异常，而JSmarter强制要求在账本读取操作前判断该键是否存在，所有参与者在使用JSmarter时也均通过require内置函数进行了判断。

综上所述，在其他四种语言实现的合约中均检测出安全漏洞，其中Go合约样本中的漏洞数量最多，Solidity合约样本其次，而JSmarter合约样本中不存在上述漏洞类型，即漏洞数量为0。因此，从本次实验的结果来看，与其他四种语言相比，开发者使用JSmarter更不易产生安全漏洞。当然，本文并没有表明JSmarter绝对安全，只是相对而言安全性较高。

# 可移植性分析

随机抽取10份实验中功能实现完整且正确的JSmarter合约样本，其中5份T1合约和5份T2合约。接着在预编译器中接入向Solidity转换的平台移植模块，对这10份合约进行预编译，生成10份Solidity合约。首先，对这10份合约的实现逻辑进行人工检查，发现均和原JSmarter合约完全对等，验证了转换后功能的正确性。其次，将这10份合约在Remix开发环境中进行编译，均通过，如图5所示，验证了转换后语法的正确性。最后，对这10份合约使用Mythril工具进行漏洞检测，在5份T1合约中均检测出时间戳依赖漏洞。这是因为Solidity没有提供创建真正随机数的方法，若想避免这一漏洞，可以通过链下生成和存储随机数或者向分布式Oracle节点请求随机数的方式，但这实际操作起来较为复杂，暂时没有集成在平台移植模块中。除此之外，没有检测出其他漏洞，验证了转换后合约的安全性高于实验中原本的Solidity样本。

综上所述，以向支持Solidity开发智能合约的区块链平台如以太坊移植为例，本实验证明了JSmarter在可移植性方面的有效性。

图5：

