第1章

探寻有力的证据

Tim Menzies
Forrest Shull

是什么使证据精确、合理、有用并有信服力？这正是本章作者在过去20年中所探寻的问题。我们两人在实证软件工程这个话题上发布的文章数目加起来都超过了200篇，组织了无数次专家小组讨论、讲习班、会议以及杂志特刊的发布。

在这篇文章中，我们想要稍微反思一下。我们回顾了迄今为止软件工程的进程，并自问："这到底有什么好处？""我们该走向哪里？"以下是我们的发现。

- 对有力证据的探寻比我们开始想的要复杂得多。
- 软件工程研究人员需要更多地工作在一个团体中，分享更多的证据。
- 我们需要承认证据多少是与背景相关的。那些使我们信服的证据并不一定使你们信服。因此，我们需要准备好，一旦听众改变，就需要重新解释和重新分析。

1.1 起步阶段

几十年前，当被问起什么是我们认为的"完美的证据"时，我们会提到下列这些特征。

- 研究的精确性

许多软件工程的研究包含了人为因素。因为很多软件技术的有效性很大程度上依赖于使用它们的人[①]。但是处理人的可变性是很具挑战的。通过精妙设计来最小化这些混杂效应的研究能使别的研究者倍感羡慕。比如，Basili和Selby的一个研究[3]使用了部分析因设计[②]，在这个设计中，每个开发人员使用的都是有待审查的技术，并且所有技术都被使用在实验中的程序片段。

- 统计的强度

① 在研究了161个项目之后，Boehm等人于2000年发现最好的项目人员的产出率是最差的3.5倍。

② Fractional Factonial Design，选择析因实验（Factonial Experiment）的一部分来研究。——编者注

随着数学复杂度的增长，对统计学显著性结果的关注也随之增长。以此，研究者能对他们的理论在现实世界中的作用有一些信心，即使在随机背景的干扰下也能识别出来。

- 结果的可重复性

如果结果可以在许多不同背景下被重复发现，也就是说，结果不止局限于一种背景或一组实验条件，这样的结果会更有说服力！在其他学科中，复制增强信心，出于这个原因，很多人花了很多精力，试图使软件工程实验可简单地被其他研究者在其他背景下重复运行[4]。举一个可重复性的例子，Turhan证明了在其他站点习得的软件缺陷预报器可以成功运用于新站点[31]。

1.2 当今证据的状态

回顾以往，我们现在才知道当时对有力证据的定义是多么幼稚。精确、统计性强、重复证据，这些都最终被证明比我们想的要难找得多。另外，它们并不能满足与研究更相关的目标。

1.2.1 精确性研究的挑战

我们发现精确的研究可能对那些接受过足够科研培训的人来说很有说服力，但对那些普通从业者却很难解释。因为这样的研究通常会有所限制和简化，从而使得研究背景不是以代表真实的开发环境。比如，Basili和Selby的研究仅针对“迷你”问题运用了研究中的技巧，该问题只在虚拟环境中有不超过400行的代码。这项研究经常被引用，虽然它是被经常复制的对象，但似乎没有一项复制使用了更大规模的或者更具代表性的应用[4]。虽然这项精确的研究对我们理解铲除代码缺陷的不同方法的优劣有重大的贡献，但如果我们在这一主题上的大部分思维来自相对小的代码段，这并不理想。

1.2.2 统计强度的挑战

什么构成了对现实世界问题的“强有力”统计数字？对于这个问题的共识出奇得少。首先，有一个关于外部有效性的问题：经过充分测试的度量方法是否能反映我们所关注的真实世界的现象。比如，Foss等人证明了常用估计工作量的评估方法根本就是有问题的[14]。更糟糕的是，他们指出，没有单个的解决方法：

> “寻找‘圣杯’是徒劳的：单个的、简单易用的、普遍适用的、使用起来可以方便地（和不同方法）做比较的度量方法，根本不存在。”

此外，不同的作者使用完全不同的统计分析法，没有一个方法被广泛地接受为“最强有力的”方法。

- ❑ Demsar记录了在一个著名国际会议上看到的巨大数量的统计方法，那个会议专注的是从数据中学习经验教训[11]。
- ❑ Cohen讨论了使用标准统计假设测试来做科学结论的方法。他尖刻地描述这种测试是“有

力但毫无结果地耙平了智力成果，剩不下任何可生长的科学种子”。[10]

为了支持Cohen的论题，我们再列举一个有益的教训。在营销领域中，Armstrong[1]使用显著性测试复查了一个研究，该研究的结论指出，从多样本来源产生的估计并不比单样本来源产生的估计要好。他通过列出了31项研究结果推翻了这个结论。在这31项研究中，多样本来源预测的效果始终超过单样本来源预测3.4%～23.4%（平均12.5%）。在Armstrong的每项调查中，这些超越都与显著性测试所预计的结果正好相反。

基于这些发现，我们改变了对统计分析的看法。现在我们会尽可能使用简洁的可视化方法来给出论点，并且把统计的显著性测试降级至“合理性测试”的角色，来验证从可视化方法中做出的结论）。①

1.2.3 结果可复制性的挑战

可复制性已被证明是一个很难捉摸的目标。在某些主题上，很少有证据证明一个项目的结果已经或者可以被推而广之。

- Zimmermann研究了629对软件开发项目[34]。在只有4%的情况中，从一个项目中习得的缺陷预测模型对另一个有用。
- Kitchenham等人的一项调查声称第一个项目的数据对第二个项目的工作量估计是有用的[18]。但他们发现现有的证据没有说服力，甚至是自相矛盾的。

在其他主题上，我们可以发现特定现象对不同环境都奏效的证据。比如，像软件检查这样成熟的技术可以找到软件产品中大量的现存缺陷[27]。然而，如果一个新的研究提出了反面证据，那也很难确定新的（或旧的）研究是有缺陷的，或者研究其实只是在一个特殊的环境下运行的。鉴于软件开发环境的多样性，两个结论通常都是有可能的。

事实上，虽然直觉上我们可能愿意相信可复制的结果，但要么关于特定的软件工程问题的研究就很少，要么它们并不全面。

作为对研究匮乏的例证，Menzies研究了不同组织提出的100个软件质量保证方法（如IEEE1017和NASA IV&V内标），发现没有实验能证明任何一个方法比别的方法更划算[21]。

现有研究不全面的例子包括如下所列。

- Zannier等人随机抽取了所有ICSE发布文章中的5%作为研究对象[33]，ICSE是自评为排名第一的软件工程大会。他们发现那些自称为“实证”的文章中，只有极少数（2%）比较了不同研究者的方法。
- Neto等人汇报了一项，针对基于模型测试（MBT）方法的文献的调查结果[23]。他们发现有85篇文章描述了71个独特的MBT方法，却只有很少数的研究有实验的部分。这显示了研究人员的总体趋势是持续创新和汇报新的方法，而不是理解可比的现有方法中的实际利益。

---

① 但我们仍然每月驳回大约两篇申请杂志发表的文章，因为他们只汇报平均值结果，没有任何平均值附近多方差的统计上或可视化的表达。最起码，我们推荐Mann-Whitney或者Wilcoxon测试（分别针对非配对结果和配对结果）来证明明显不同的结果真的可能是不同的。

为了看看这些文章反应的到底是一种个别情况还是一种更普遍的趋势，我们审阅了所有PROMISE[①]大会上做过的关于可重复软件工程实验的演讲。从2005年起，在PROMISE大会上。

- 一共有68个演讲，其中48个尝试了对旧数据的新研究或者以Zannier[33]的形式做了汇报，即一个对特定项目生效的新方法。
- 9篇文章对之前研究结果的有效性提出了质疑（如 Menzies 的报告[22]）。
- 4篇文章主张软件工程模式的通用性是不可能的（如 Briand 的报告[7]）。
- 只有很少的研究者（68个演讲中的7个）汇报了从一个项目到另外项目的推广：
  - 4篇文章汇报了从一个项目中习得的软件质量预警器在一个新的项目中成功使用（如Weyuker等人[32]和Tosun等人[30]的报告）；
  - 3篇文章举了特定案例说明通用性是可行的（如Boehm的报告[5]）。

这些发现稍微引起了我们的一些警惕，我们与欧美领先的实证软件工程研究者进行了讨论。我们的谈话可以总结为以下几点。

- Vic Basili在过去30年中是实证软件工程（SE）的先驱。在断言现在的实证软件工程已经比20世纪80年代健康许多后，他承认了两点。第一，迄今为止的研究结果都是不全面的。第二，几乎没有例子能举出有多个项目可用的方法[2]。
- David Budgen和Barbara Kitchenham一起，是欧洲提倡“循证软件工程”（EBSE）的领军者。在EBSE中，软件工程师的实践应该基于有文献充分支持的方法。Budgen和Kitchenham试图探明：“循证软件工程是否在实践和政策上已经足够成熟？”他们的回答是“不，还不是”：软件工程领域需要大幅调整，只有在那之后，我们才能证明研究结果是可以在不同项目中复制的[8]。它们主张软件工程中不同的报告标准，特别是使用“结构式摘要”来简化对SE文献的大规模分析。

1.3 我们可以相信的改变

至今，我们还未实现对精确、统计充分、可复制证据的梦想。而且，即使找到了符合单个特征的证据，也没有我们想象中的有影响力。也许，我们需要重新审视对“有力证据”的定义了。根据前几节的反馈，是否有更实际的对有力证据的定义，以此激励研究者？

（如果谦虚地说）更可行的定义是：有力的证据*激发改变*。我们猜想，这本书中的很多作者都是因为看到真实软件开发的第一手问题和困难，才开始了对有力证据的搜寻。这些研究者寻找的“圣杯”应该是那些能够创造真实世界进步的研究结果。

为了激发改变，一些有影响力的读者不得不去相信证据。有一种处理不够严格的经验报告或案例分析的方法，是对每件证据赋予一个“信心指数”。这个指数试图反应每篇报告在一定信心范围内的位置，从传闻性的证据，或者说有问题的证据，到非常可信的证据。Kitchenham建议对系统性评论和软件工程研究使用新的流程，而这样的评估正是该流程的必要部分[17]。在Feldmann

① 查看http://promisedata.org。

的一篇文章中，也可以找到一种帮助从业者交流信心指数的简单刻度[13]。

生成真正令人信服的证据也需要对传播结果的途径做出改变。也许，科技出版物不再是传播证据的唯一技术。现在的出版物存在很多问题，其中之一就是，这些出版物通常不提供原始数据，其他研究者就不能重新分析、重新解译并验证。另外，虽然出版物的作者非常努力地对背景做出详尽的描述，但也不可能列出所有相关因素。

更有希望的一种做法是，创造软件工程数据知识库，用来存储跨环境的研究成果（至少是跨项目的），然后可以从这个仓库的数据中展开分析。如下所示。

- 内布拉斯加大学的软件构件基础设施研究（SIR）中心[①]

  这个知识库存储了软件相关的构件，研究者可以把它用在有程序分析和软件测试技术严格控制的实验法中。教师也可以用它在可控实验中来训练学生。这个知识库包含了很多Java和C的不同版本软件系统，也包含了支持构件，如测试套件、缺陷数据和脚本。这个知识库中的构件已经在上百个出版物中被使用。

- NASA软件工程实验室（SEL）

  NASA实验室（详见第5章）是一个巨大的成功。它有广泛的影响力，它的数据和结果也持续被引用。有一个涉及背景的重要经验：实验室的领导者要求想用他们数据的研究者花时间做实验，从而正确理解他们的研究背景，不曲解他们的分析结果。

- CeBASE

  这是一个数据和经验的知识库，由NSF资助，用于与其他研究者共享并重新分析。我们利用了SEL的经验，探索把数据用较少的开销置于不同背景的方法，如用一套丰富的元数据标记所有的数据集，元数据描述了数据的出处。这些数据经验回过头来成为另一个“经验教训”知识库的支柱，这个知识库由美国国防部的国防采办大学维护，可以让最终用户指定他们自己的背景参数，比如项目的大小、重要性或者领域，以此找到在相似环境中被证实的实践。

- PROMISE项目

  另一个活跃的软件工程知识库来自于PROMISE项目，这个项目在本章的前面也被引用过。PROMISE探寻可重复的软件工程经验。这个项目分为三个部分。

  - ❑ 在线公共域数据知识库。在写此文时，该知识库包含91了个数据集。其中一半关于缺陷预警，剩余的数据集探索了工作量预估、基于模型的软件工程、SE数据的文本挖掘以及其他问题。
  - ❑ 每年一次的年会，强烈鼓励作者不仅发布文章，也在使用数据知识库做出他们的结论后，为知识库做贡献。
  - ❑ 特别期刊，发表年会中最好的文章。[19]

知识库在提供链接至数据贡献者时特别有用。我们必须承认，数据很少独立存在。作者应该把回答问题以及与试图解读他们工作的人对话视为常态，而非反常情况。这种接触对理解得出数

① 查看http://sir.unl.edu。

据的背景很重要。也能帮助用户找到与他们的问题和他们的环境最相关的数据。巴尔的摩市马里兰大学（UMBC）的Gunes Koru在PROMISE的一些数据集中找到了一个系统错误。他通过PROMISE知识库附属的博客软件张贴了这个错误。然后，通过PROMISE的博客[①]，他联系到了原始数据的制造者，这些制造者对这些错误的原因以及避免方法提供了大量的评论。

1.4 背景的影响

PROMISE这样的知识库虽然是找到有推动性和有说服力证据的必要因素，但它们也只是解决方法的一部分。我们在之前章节中描述了为什么证据的收集总是与上下文相关的。最近我们开始认识到，解读证据也是与上下文相关的，甚至是与读者相关的。为了更好地认识这个问题，我们需要离题一会，去讨论一点理论。

许多软件工程的问题存在于一个解空间内，这个解空间就像随意扔在地板上的地毯一般任意扭曲。想象一个蚂蚁正在搜寻这块地毯的高峰和低谷，试图找到最低点，比如在这个点上，软件开发的工作量和缺陷的数量最少。

如果问题足够复杂（软件设计和软件流程的决定也确实非常复杂），就不存在找到最佳解决方案的最佳途径。也就是说，蚂蚁可能被困在错误的低谷中，自认为是最低其实却不是（比如，一些山脊阻碍了它的视线，使之看不见相邻的更低谷）。

优化算法和人工智能算法使用各种试探性方法来搜索这个选择空间。一种方法是根据特定观众的目标对问题的上下文建模，然后把搜索引导至一个特定的方向。想象那只蚂蚁在一根皮带上，而这根皮带向着目标方法被轻轻地拉动。

现在，令人吃惊的来了。这些试探性搜索方法虽然有用，但却对研究结果施加了一个偏差。如果你改变上下文，也改变了偏差，那这些算法就会找到不同的“最佳”方案。例如，在对软件过程模型的人工智能搜索实验中，Green等人使用了两种不同的目标函数[15]。一个目标代表了安全性至关重要的政府项目，试图同时减少开发工作量和软件交付的缺陷数。另一个目标代表了更标准的商业情况，急于发布软件至市场，并一直努力不要插入过多缺陷。此研究用人工智能优化器搜索了4个不同项目。每次搜索在每个目标下重复。一个惊人的结果是，一个目标产出的建议通常在另一个目标下会被否决。比如，使用一个目标的人工智能搜索推荐增加交付前的时间，而另一个则建议减少时间。

对任何试图找到证据来说服人们改变现有软件开发流程和工具的人来说，这个结果都有重要意义。我们需要根据观众调整证据，而不是假设所有证据都能说服所有人。站在台上像变戏法似的拿出看似动人的证据并不足够。即使是从精确研究中取得的有统计强度的、可复制的证据，如果证据与观众问题不相关，那也不能激发任何改变。换句话说，观众可能会自问：“这到底有什么好处？”而我们需要尊重他们的“业务偏差”。

① 查看 http://promisedata.org/?p=30#comments。

1.5 展望未来

虽然软件工程研究已经进行了几十年，但是至今我们只看到了极少的有力证据，能引导软件项目运行方式的改变。我们推测这是由于背景的问题：研究者制造了关于A的证据，而观众却关注B、C、D等。我们推荐研究者在搜寻软件工程证据时多一点谦卑，至少和现在持平，并愿意结识并倾听软件从业者，他们可以帮助我们更好地领会B、C、D究竟是什么。我觉得我们的领域需要在至少一段时间内，停止搜寻对所有项目所有情况都适用的结果的证据，找到有影响力的局部结果已经足够有挑战性了。

Endres和Rombach提出了一个，关于如何构建软件和系统工程知识的观点[12]。

- 在特定环境下对实际开发工作进行观察，这在任何时候都能进行。（“观察”在这里的定义同时包含铁的事实和主观印象。）
- 重复的观察会引出一些规则，帮助理解事情将可能会如何发生。
- 规则可以被理论所解释，解释为什么这些事件发生了。

鉴于当前经验性研究处理问题的复杂性，我们不应该急于进行理论建设。一个更多产的知识构建模型基于我们现在看到的所有数据和研究，是一种“观察”和“规则”的双层方法（使用Endres的术语），由我们之前描述的知识库来支撑。

对第一层来说，研究者会对局部观众的目标建模，然后收集关注这些目标的证据。已有的现代化数据挖掘工具①，可以帮助工程师们学习局部的经验教训。

在第二层中，我们把不同项目和不同背景的结论抽象出来。现在可能必须满足于只在一个领域中抽象出重要因素或基本原则，而不是提供“唯一”的解决方法对所有不同背景的子集都适用。

比如，Hall等人试图回答什么能激励软件开发者。[16]他们查看了92个实验过此问题的研究，每个都有不同的背景。在试着弄清楚这些研究和它们的不同研究结果时，研究者在寻找看似能激励开发人员的因素，即使不太可能量化这些因素的贡献。于是，伴随着一定的可信度，在多个研究中所找到的有作用的激励因素被包含在这个模式中。

最终结果不是一个预言性的模型，如X因素是Y因素重要性的两倍。相反，它是一个重要因素的列表，管理者可以在自己的环境中用它来确保他们没有忽视或忘记某些东西。也许，在许多问题上，我们能做到的最好情况是，提供需要考虑的因素来装备那些带着问题的从业者，让他们找到自己问题的解决方案。

要做到这点，我们需要扩大第一层观察所能接受的“证据”的定义。一些研究者一直争论说，软件工程研究存在对定量数据和研究的偏好，而定量的工作也可以同样严格并提供对相关问题的有用回答[26]。构建更强大证据集合的开始，是真正拓展可接受证据的定义，以此混合定性和定量的资源。也就是说，更多关于技术为什么能行或者为什么不行的文本或图片数据，以及技术影响力多少的定量数据。

然而，真正有影响力的证据实体走得更远，并且接收完全不同种类的证据。不止是试图找到

① Weka：http://www.cs.waikato.ac.nz/ml/weka; R: http://www.r-project.org; YALE: http://rapid-i.com/。

具有统计意义的研究，也包含能提供更多关于技术实践应用信息的调查经验汇报。这种经验汇报目前被低估了，因为它们被认为不如现有的文献严格。比如，不总能确保各方面利益都被精确衡量，不能保证混淆的因素已经排除，或者不能证明流程一致性因素已经避免。然而，这些报告应该是任何软件开发技术“证据线索”的一个显而易见的部分。

正如Andres把“观察”定义地足够广泛来包含主观印象，如果我们不根据他们的标记去创建无根据的规则，即使不那么严格的输入模式也可以帮助我们找到有效的结论。对这些证据源赋予信心指数非常重要，这样，方法论的问题就可以被凸显出来。（当然，即使最严格的研究也不会完全没有任何方法论的问题。）

经验汇报可以通过证明实践约束下达成的结果不总是与我们对给定技术的期望相匹配，从而使研究有根有据。同样重要的是，他们对于需要如何剪裁或修改技术来应对日常软件开发的实践约束提供了见解。在技术转移领域，我们经常发现，一个描述技术在实践中正面经验的案例分析，对从业者来说，比大量额外的研究报告更有价值。如此令人信服的证据体，也能通过提供对不同用户有用的证据来帮助引导改变。Rogers提出了一个经常被引用的模型，用钟形曲线刻画了一些创新产品（如研究结果）的消费者：最左边的尾巴包含了创新者，钟形的突出部分代表了大多数人，他们随着这个创新想法逐渐流行而采纳，最右边的尾巴包含了抵触改变的落后者[24]。了解自己观众的研究者能从真正健壮的数据集中选择适当的子集来构建他们的案例。

- 早期采纳者可能找到的是包含相对低可信度的可行性研究的一个小集合，或者一到两个“精确”研究，这已经足够使他们采纳改变。特别是当这些研究的背景显示出证据是在一个与他们相似的环境中收集的。
- 大部分采纳者需要看到不同上下文中的多样研究，才会相信研究的想法是有价值的，被证实不止在一个合适的环境中可行，并且已经开始成为被部分接受的行事方式。

落后者或者后期采纳者可能需要压倒性的证据，这可能包括大量的高可信度研究，以及在大范围不同背景下得到的有益结果。

一些混合这些不同类型证据的方法已经被提出[29]。但最重要的是，定性报告和定量数据之间的相互影响。我们经常看到从业者在混合不同数据类型的数据集后收到更好的效果。例如，一个同时包含硬数据和真实团队正面经验的丰富集合，帮助了软件检查技术在不同NASA中心的散播[28]。

从良好设计的经验性研究中得出的证据可以帮助结果达到统计意义，但实施可能仍然不切实际，或者不能及时回答新问题（特别在技术快速改变的领域，如软件工程）。实际运用中的证据可能对益处更有说服力，但是通常不太严格。经常需要同时结合两种证据，相互巩固，才会最具说服力。

1.6 参考文献

[1] [Armstrong 2007] Armstrong, J.S. 2007. Significance tests harm progress in forecasting. *International Journal of Forecasting* 23(2): 321-327.

[2] [Basili 2009] Basili, V. 2009. Personal communication, September.

[3] [Basili and Selby 1987] Basili, V., and R. Selby. 1987. Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering* 13(12): 1278-1296.

[4] [Basili et al. 1999] Basili, V.R., F. Shull, and F. Lanubile. 1999. Building Knowledge Through Families of Experiments. *IEEE Transactions on Software Engineering* 25(4): 456-473.

[5] [Boehm 2009] Boehm, B. 2009. Future Challenges for Software Data Collection and Analysis. Keynote address presented at the International Conference on Predictor Models in Software Engineering (PROMISE09), May 18-19, in Vancouver, Canada.

[6] [Boehm et al. 2000] Boehm, B.W., C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. 2000. *Software Cost Estimation with Cocomo II.* Upper Saddle River, NJ: Prentice Hall PTR.

[7] [Briand 2006] Briand, L. 2006. Predictive Models in Software Engineering: State-of-the-Art, Needs, and Changes. Keynote address, PROMISE workshop, IEEE ICSM, September 24, in Ottawa, Canada.

[8] [Budgen et al. 2009] Budgen, D., B. Kitchenham, and P. Brereton. 2009. Is Evidence Based Software Engineering Mature Enough for Practice & Policy? Paper presented at the 33rd Annual IEEE Software Engineering Workshop (SEW-33), October 13-15, in Skövde, Sweden.

[9] [Carver et al. 2008] Carver, J., Juristo, N., Shull, F., and Vegas, S. 2008. The Role of Replications in Empirical Software Engineering. *Empirical Software Engineering: An International Journal* 13(2): 211-218.

[10] [Cohen 1988] Cohen, J. 1988. The earth is round ($p < .05$). *American Psychologist* 49: 997-1003.

[11] [Demsar 2006] Demsar, J. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7: 1-30.

[12] [Endres and Rombach 2003] Endres, A., and H.D. Rombach. 2003. *A Handbook of Software and Systems Engineering*. Boston: Addison-Wesley.

[13] [Feldmann et al. 2006] Feldmann, R., F. Shull, and M. Shaw. 2006. Building Decision Supportin an Imperfect World. *Proc. ACM/IEEE International Symposium on Empirical Software Engineering* 2: 33-35.

[14] [Foss et al. 2003] Foss, T., E. Stensrud, B. Kitchenham, and I. Myrtveit. 2003. A Simulation Study of the Model Evaluation Criterion MMRE. *IEEE Transactions on Software Engineering* 29(11): 985-995.

[15] [Green et al. 2009] Green, P., T. Menzies, S. Williams, and O. El-waras. 2009. Understanding the Value of Software Engineering Technologies. *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering:* 52-61. Also available at *http://menzies.us/pdf/ 09value.pdf.*

[16] [Hall et al. 2008] Hall, T., H. Sharp, S. Beecham, N. Baddoo, and H. Robinson. 2008. What Do We Know About Developer Motivation? *IEEE Software* 25(4): 92-94.

[17] [Kitchenham 2004] Kitchenham, B. 2004. Procedures for undertaking systematic reviews. Technical Report TR/SE-0401, Department of Computer Science, Keele University, and National ICT, Australia Ltd.

[18] [Kitchenham et al. 2007] Kitchenham, B., E. Mendes, and G. H. Travassos. 2007. Cross VersusWithin-Company Cost Estimation Studies: A Systematic Review. *IEEE Trans. Software Eng.* 33(5): 316-329.

[19] [Menzies 2008] Menzies, T. 2008. Editorial, special issue, repeatable experiments in software engineering. *Empirical Software Engineering* 13(5): 469-471.

[20] [Menzies and Marcus 2008] Menzies, T., and A. Marcus. 2008. Automated Severity Assessment of Software Defect Reports. Paper presented at IEEE International Conference on Software Maintenance, September

28-October 4, in Beijing, China.

[21] [Menzies et al. 2008] Menzies, T., M. Benson, K. Costello, C. Moats, M. Northey, and J. Richardson. 2008. Learning Better IV & V Practices. *Innovations in Systems and Software Engineering* 4(2): 169-183. Also available at *http://menzies.us/pdf/07ivv.pdf.*

[22] [Menzies et al. 2009] Menzies, T., O. El-waras, J. Hihn, and B. Boehm. 2009. Can We Build Software Faster and Better and Cheaper? Paper presented at the International Conference on Predictor Models in Software Engineering (PROMISE09), May 18-19, in Vancouver, Canada. Available at *http://promisedata.org/pdf/2009/01_Menzies.pdf.*

[23] [Neto et al. 2008] Neto, A., R. Subramanyan, M. Vieira, G.H. Travassos, and F. Shull. 2008. Improving Evidence About Software Technologies: A Look at Model-Based Testing. *IEEE Software* 25(3): 10-13.

[24] [Rogers 1962] Rogers, E.M. 1962. *Diffusion of innovations.* New York: Free Press.

[25] [Runeson et al. 2006] Runeson, P., C. Andersson, T. Thelin, A. Andrews, and T. Berling. 2006. What do we know about defect detection methods? *IEEE Software* 23(3): 82-90.

[26] [Seaman 2007] Seaman, C. 2007. Qualitative Methods. In *Guide to Advanced Empirical Software Engineering*, ed. F. Shull, J. Singer, and D. I. K. Sjøberg, 35-62. London: Springer.

[27] [Shull 2002] Shull, F., V.R. Basili, B. Boehm, A.W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M.V. Zelkowitz. 2002. What We Have Learned About Fighting Defects. Proc. *IEEE International Symposium on Software Metrics (METRICS02)*: 249-258.

[28] [Shull and Seaman 2008] Shull, F., and C. Seaman. 2008. Inspecting the History of Inspections: An Example of Evidence-Based Technology Diffusion. *IEEE Software* 24(7): 88-90.

[29] [Shull et al. 2001] Shull, F., J. Carver, and G.H. Travassos. 2001. An Empirical Methodology for Introducing Software Processes. *Proc. Joint European Software Engineering Conference and ACMSIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE):* 288-296.

[30] [Tosun et al. 2009] Tosun, A., A. Bener, and B. Turhan. 2009. Practical Considerations of Deploying AI in Defect Prediction: A Case Study within the Turkish Telecommunication Industry. Paper presented at the International Conference on Predictor Models in Software Engineering (PROMISE09), May 18-19, in Vancouver, Canada. Available at *http://promisedata .org/pdf/2009/09_Tosun.pdf.*

[31] [Turham et al. 2009] Turhan, B., T. Menzies, A.B. Bener, and J. Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 14(5):540-557.

[32] [Weyuker et al. 2008] Weyuker, E.J., T.J. Ostrand, and R.M. Bell. 2008. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empirical Software Engineering* 13(5): 539-559.

[33] [Zannier et al. 2006] Zannier, C., G. Melnik, and F. Maurer. 2006. On the Success of Empirical Studies in the International Conference on Software Engineering. *Proceedings of the 28th international conference on software engineering*: 341-350.

[34] [Zimmerman 2009] Zimmermann, T., N. Nagappan, H. Gall, E. Giger, and B. Murphy. 2009. Cross-Project Defect Prediction. *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*: 91-100.