# **第18章 大数据时代的软件工程**

软件开发管理过程中产生了源代码、错误日志、测试用例、用户评论等大量数据，分析和利用这些数据，解决需求、编码、测试等阶段的问题，已经成为现代软件工程的一项重要活动。软件需求获取阶段，开发人员不再是完全凭借经验和直觉来进行软件的需求决策，而是通过采集大量的软件评论数据集，应用机器学习与数据挖掘方法，进行定量或定性分析来确定软件需求；软件编码阶段，开发人员的技术学习和优秀代码复用已经逐渐转向了开源项目，开源项目准确匹配和源代码辅助推荐已经是一个十分重要的发展方向；软件测试阶段，为了提高软件产品质量而进行的测试是一项艰巨和复杂的任务，数据驱动测试是提高测试效率，缩短软件开发周期的重要解决方案。本章将详细介绍大数据时代对软件工程各阶段带来的影响。

## 18.1 大数据与软件需求

传统软件工程的需求获取，一般是基于软件涉众（stackholder）的直觉和经验，以及一些基本原理（如标准、现有方案和论述）的指导，但直觉一般是主观的，可能具有不一致性，并且缺乏理论的支持。另一方面，基本原理会随着时间的推移而变化，因此需求分析人员很难捕捉到准确的用户需求并进一步细化。

大数据时代下，我们可以从用户社区、在线论坛、社交媒体等多种渠道，获得一系列支持软件开发需求决策的信息。同时，随着开源软件和应用商店的出现，用户能够轻松提交反馈、发表评论、报告漏洞，给应用和功能评分，或请求开发新的功能，与软件相关的反馈信息可以进一步帮助开发和运营人员进行需求获取和分析。图18.1展示了传统软件工程的需求获取与大数据时代软件需求获取的比较情况。

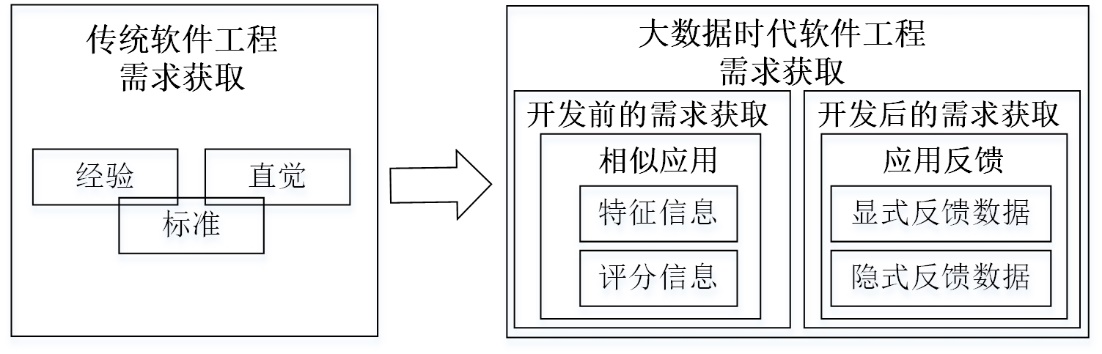


图18.1 传统软件工程和大数据时代软件工程需求获取的比较

### 18.1.1 数据驱动的软件需求获取

数据驱动的软件需求获取可以通过对相似应用的分析来获取一款新应用的需求。首先将相似应用的特征信息（如功能、历史版本、价格等）和评论评分信息相结合，把应用需求的捕获问题转化成一个优化问题，即根据类似应用的经验，确定新应用的最佳功能集，然后对新应用的功能吸引力进行评估，在综合了用户对新功能集的满意度之后，再选择新应用要实现的功能进行开发。

应用上线后，通过收集用户对应用的反馈，使用数据挖掘分析技术可以了解用户使用应用程序的总体情况（如错误报告或功能请求数量）。借助这些数据分析结果，需求分析人员和产品设计人员可以确定软件版本迭代的主要方向，也可以与类似的应用程序进行比较来辅助获取需求。

### 18.1.2 用户反馈数据挖掘

**1. 显式反馈数据**

随着移动设备的爆发式增长，移动应用程序通过应用商店直接分发给终端用户，用户可以在应用商店上直接对程序进行评论。软件应用商店的用户评论数据是一种比较典型的显式反馈数据。数据驱动的软件需求获取过程通常直接使用分析工具将收集到的用户评论进行分类，过滤掉不相关的评论，再对每个类别进行摘要总结，提取出有用信息，以此来帮助需求分析人员进行辅助需求决策。

以评论数据的分析为例，需求分析人员首先会根据数据对用户进行分类，根据使用方法、专业术语或角色任务的描述进行分析，挖掘并分类应用软件的使用人群，有时甚至可以获得意料之外的目标用户群体。例如，苹果应用商店中有放大镜和翻译类的应用广受护士们的好评，因为护士可以使用放大镜类的应用阅读药品上很小的说明文字，使用翻译软件和外籍病人熟练地交流，但这些应用其实并不是针对医药健康类型的专业应用[1]。通过对用户进行分类，能够捕捉到这些不同寻常的特征，支持应用软件的后续开发。其次，用户评论数据一般比较复杂，一条评论数据可能包含多种类别的信息（如漏洞报告、功能请求等），因此，需求分析人员可以使用自然语言处理等技术将评论以句子或段落为单位进行预处理，然后按照信息类别进行分类提取。

用户评论过滤主要是为了分类后的用户反馈数据可以被更高效地处理，直接过滤掉简短的评分型评论后，将分类数据交由项目团队中相应的人员进行处理。例如，报告漏洞的反馈数据交给开发人员和安全团队进行高优先级处理，而功能需求、用户体验相关的反馈数据则交给需求分析人员和产品设计人员来讨论以确定未来版本的迭代方案。

用户评论摘要生成方法主要包括基于应用功能的摘要和基于主题的摘要两类：基于应用功能的摘要生成方法主要使用自然语言处理和情感分析技术，将用户评论中每个应用功能进行提取并生成摘要，这种方法能够帮助开发团队了解用户更关心什么功能点，更喜欢或更讨厌哪些功能点，由此确定不同功能模块的重要性和未来版本迭代中开发的优先级；基于主题的摘要生成方法一般使用主题建模和情感分析方法从评论中提取出主题词并进行情感分类，主题可以是整个版本、应用程序、价格或支持服务的质量。

**2. 隐式反馈数据**

显式用户反馈数据在缺乏上下文内容的情况下通常对开发人员的帮助并不大，比如一些负面的显示反馈数据往往只包含一些情感宣泄的短评论。用户的使用数据以及交互历史通常被包含在隐式反馈数据中，能够帮助开发人员更好理解用户当时所处情境，这些数据相对于主观的显式反馈数据更加客观和合理。

隐式反馈数据也需要通过分析、过滤、摘要和可视化等过程才能够为开发人员提供可用的反馈信息。隐式反馈数据可以包括用户点击事件、用户接口交互等等，其中图形化的元素通常与特定应用功能、组件或需求有直接关联，在没有显式反馈数据的情况下，通过对这些隐式反馈数据的分析也能帮助相关人员做出需求决策。

### 18.1.3 软件需求预测模型

软件功能是否能够按计划完成迭代非常重要，在软件研发的迭代过程中频繁且稳定的软件功能发布能够给软件开发团队或个人带来更多价值。但是目前大量的研究发现高级需求（区别于bug修复等一般性需求）并没有按计划进行迭代，通常会被移到下一个版本中，或者返回到产品的待开发功能列表之中。基于机器学习与海量数据的软件需求预测模型具有很好的预测效果。

大型软件项目中有研究人员使用定性和定量相结合的方法分析高级需求的迭代变化情况，提出通过需求预测模型来预测高级需求是否会在其计划的迭代中完成，研究表明对于部分需求，其计划的迭代更改了至少三次以上。此外，研究者还通过建模预测开源项目问题（issue）的解决周期，研究人员收集了超过4000个代码托管平台上开源项目的问题数据集，过滤并分析了这些问题的生命周期，通过提取其静态特征（如问题的创建时间）、动态特征（如问题中的评论数量及针对问题采取的行动数目）和上下文特征（如相关项目最近的开发活动），构建和训练机器学习模型，预测问题完成时间，为软件后续的需求迭代提供指导[2]。

### 18.1.4 小结

针对传统软件工程中的需求获取问题，大数据相关技术能够很好地提高需求预测精度并在实践中达到令人满意的效果。开发人员不再是完全凭借经验和直觉来进行需求决策，而是基于数据挖掘技术和机器学习方法以及互联网中获取的大量数据集进行定量或定性分析，需求探索更关注用户真实体验与反馈，这往往比需求决策者自己的思考更具指导意义。从这些点来看，大数据技术更好地推动了软件需求工程的发展。

## 18.2 大数据与软件开发

软件开发过程一般包括软件需求分析、概要设计和详细设计、编码、测试、交付部署、维护等阶段。当对用户需求进行充分调研之后，需要针对用户需求进行详细的软件架构设计和模块设计，然后根据详细设计文档指导系统的编码实现。

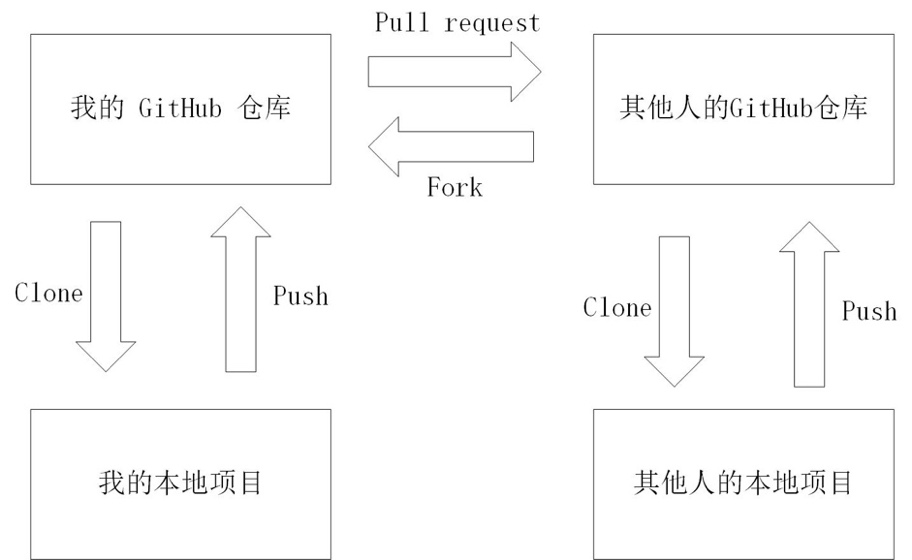


图18.2 Pull&Request开发模型

自从开源社区面世以来，它所固有的开放性、灵活性、协同性吸引了一批又一批优秀的国内外开发人员加入其中，活跃在开源社区的开发人员和优秀的开源项目每年都保持稳定的高速增长，以GitHub的拉取&请求（Pull&Request）开发模型为例，开发人员可以参与任何一个公开发布的仓库并将项目克隆到自己的仓库中，可以通过Pull& Request合入自己的更改，具体流程参考图18.2。但是，对于软件开发而言，降低每个开发人员的时间成本和精力，从海量的、质量参差不齐的开源项目中找到对自己有帮助的项目，并将自己的主要精力放在代码编写、技术成长上是软件开发成功的关键。

### 18.2.1 开源项目推荐

**开源软件**是指允许用户基于开放源代码促进会（Open Source Initiative，OSI）列出的开源协议，在协议许可的范围内自由使用、修改软件源代码，并且可以将软件源代码与其他软件代码结合使用的一种软件形式[3]。社交化开发平台GitHub是众多开源社区之一，是目前世界上最大的代码托管平台和开源社区，它不仅允许个人和组织创建、浏览代码库，而且还提供社区化软件开发的功能，包括关注其他用户、管理代码仓库动态、跟踪仓库代码的改动、Bug和评论等。此外，GitHub还提供代码仓库使用Wiki以及通过Git进行项目协同开发。

基于开源软件的软件开发模式所具有的灵活性、开放性、协同性使其在开源社区催生了一批又一批优秀的项目。但是，在海量的开源项目中找到开发人员真正感兴趣、需求相似度高的项目却需要花费大量的时间和精力，这无疑增加了开发人员的开发成本。因此，个性化的开源项目推荐方法对于大数据时代下的软件开发具有重大意义。

引入推荐算法为开发人员找到高质量且感兴趣的开源项目将有非常重要的作用，与开源项目相关的各种数据，如源代码，问题、拉取请求（pull-request）、评论、提交信息等均可用于构建推荐模型，但用于推荐算法的数据选择是最为核心的工作。推荐算法应该从开发人员的目标与关注点出发构建开源项目推荐系统。

大部分开发人员并不是开源项目的参与者，更多的是希望找到并复用需求相似度高的项目。在软件开发过程，如果开源项目与正在进行的项目有极大的技术相关性，那么开发人员很有可能对这些项目里包含的软件需求、架构、设计、代码以及文档进行复用，侧重于项目功能的评论，源代码等数据对项目相关性的分析更有用。

如果开发人员期望学习并加入开源项目，他们更多地考虑开源项目的质量和兴趣。在开源社区受到广泛关注，吸引众多外围开发人员参与的开源项目通常质量更高，以GitHub的fork、stars和watch机制为例，这些机制都可以有效地反映不同开发人员对一个项目的关注情况，fork主要反映贡献者的参与程度，stars反映一个项目的关注者个数，watch则更多反映项目受到关注的程度，开发人员一旦对某个项目抱有强烈的兴趣，那么就可以通过watch机制对项目进行实时跟踪，通过fork机制对项目进行本地开发，因此，可以将项目的watch数量，stars数量以及fork数量作为项目流行度和关注度的量化标准。

### 18.2.2 源代码推荐

随着软件开发技术的发展，软件系统具有高度的复杂性，用于提高开发效率实现功能复用的软件源代码模块（代码片段，构件等）不断产生，但简单的模块中经常包含数千个功能特性和大量有用的信息，掌握其中有价值的信息对开发人员来说具有很大的挑战性。直接通过自然语言检索可复用代码片段是一种非常好的解决方案，这种推荐方案通常以自然语言作为输入，输出为多个觉好满足自然语言功能描述的代码片段。Gu [4]等人利用代码段和自然语言的平行语料使用RNN训练了一个代码段和自然语言描述的联合嵌入模型，提供基于自然语言的代码块检索功能；Shen [5]等人则是从常见的库函数出发，构建自然语言描述到代码模板乃至代码块的工具NLI2Code。近些年，借助于大量的自然语言和代码片段的平行语料，代码领域的预训练模型也开始兴起，例如CodeBert [6]、GraphCodeBert [7]等，这些预训练模型从大量的语料库中学习到了广泛的知识，通常能更好地完成基于自然语言的代码片段检索。另一种更加便捷的方案是在开发过程中通过开发上下文直接主动向用户提供与任务相关的个性化代码片段推荐，这种推荐方法既不需要用户发起信息寻求过程，也不需要用户提供查询，而是根据开发人员的开发环境及上下文直接感知与任务相关的信息并反馈给开发人员，减少了寻求信息所需的时间，降低了开发人员的时间成本。Nguyen [8]等人提出挖掘和分析API调用模式，在用户开发项目时利用协同过滤技术直接推荐相应的API以及代码片段；He[9]等人通过分析Python代码中数据流、字符相似度等信息，利用机器学习技术实现了一个工具PyART，可以为开发者实时推荐Python API。

目前比较流行的智能化软件开发通过分析编程环境中正在开发的部分程序，不仅能够主动提供开发人员在当前编程任务中可能会使用的代码片段或构件，而且可以提供示例说明，生成注释。可重用代码片段或构件的推荐主要是基于深度学习在开源平台上通过使用海量代码训练实现。

### 18.2.3 小结

随着信息技术的发展，存在大量优秀的开源软件可供开发人员借鉴，并极大降低了开发成本，基于大数据相关技术的开源项目推荐与源代码推荐逐步成为未来软件开发的一种趋势。由于众多开源社区的项目质量参差不齐，技术栈多样化，开发人员面临的挑战也日益明显。如何实现开源项目、源代码等开源资源与开发人员以一种较高准确度的个性化匹配，如何通过尽可能多地提供自动化过程以减少开发人员所需要的额外工作，如何支持和鼓励开发人员提供可重用组件以便其他开发人员可以重用并确保修改后可重用组件的高质量，大数据时代下基于开源项目和源代码推荐的软件开发仍有很多的工作要做。

## 18.3 大数据与软件测试

软件测试是保障软件产品质量、稳定性和可靠性等问题的关键。软件测试是在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否能满足设计要求进行评估的过程，它是一种实际输出与预期输出之间的审核或者比较过程。

在大数据时代背景下，软件使用的复杂性不断增加，软件需求变更频率越来越快，传统的测试方法对需求的变化响应不及时且难于支持迭代开发，使得测试人员将同时面临着测试任务重，测试周期短的实际问题。

### 18.3.1 测试多元化

测试方法和测试主体需要多元化。复杂的软件演化过程中，软件测试的“杀虫剂免疫效应”突出。Boris Beizer 在1990年提出了软件测试的“杀虫剂免疫效应”，即软件测试的种类及数量越多，发现缺陷的数量越来越少但检测难度越来越大，这就像传统的农药杀虫剂一样，长期的利用一种农药进行杀虫，该地区的害虫会在体内形成抗药性，导致杀虫剂作用效果逐渐降低。在传统的软件测试工作中，测试主体多为软件公司内部的专职测试人员或是专门的测试团队，以测试为主的沟通协作多集中在公司内部，测试方法和过程容易单一化。

随着信息技术的快速发展，基于众包模式的众测平台逐渐成为软件测试的一个主流选择，凭借人力众包测试优势，较好地满足了测试多元化需求，开始得到越来越多软件开发公司的关注。众测平台上的测试主体是分布于各个行业、拥有各类设备的测试专家，具备行业、设备、方法多样性的特点。众测平台上的大量众包测试项目产生和积累了大量数据，既包含专业能力相关的各类数据，如从业经验，擅长行业等，也包含测试专家在平台上参与的每项任务的详细统计数据，如提交的缺陷数量和质量、客户的评价等，基于这些数据能够形成专业知识库和测试专家分级分类体系。众测平台基于大数据分析能力可以为软件测试需求方与测试专家进行智能匹配，更容易实现测试需求对接。众测平台通过对大量测试结果的精准分析可以得到精度较高和相对可信的测试报告，缩短测试成本和测试周期。

### 18.3.2 测试自动化

随着软件功能的迭代开发，测试人员需要对其进行回归测试，以验证新特性功能是否正常，代码修改是否将新的错误引入了之前测试的代码。**自动化的软件测试**主要指测试人员通过测试工具或其它手段，根据测试计划编写测试脚本，让机器代替手工测试，通过对软件的多次回归测试，达到改进软件质量加快测试进度的目标。在软件工程实践中，测试人员经常通过构建自动化测试框架以重用所有的测试用例来测试引入新特性后的软件系统版本，但这种重新运行所有测试用例的方法的代价是昂贵的。虽然回归测试在维护软件后续版本的质量方面非常关键，但它会带来巨大的人力成本，占软件开发成本很大一部分[10]。

目前存在很多提高回归测试成本效益的方法：回归测试选择技术是通过从现有测试套件中选择测试用例的子集在修改后的软件系统上执行从而降低测试成本；测试套件最小化技术是通过识别和消除冗余测试用例来减少测试套件的大小。这两种技术都通过减少测试时间和维护时间来降低测试成本，但使用这两种技术需要验证其安全性，否则可能会遗漏测试场景，对软件系统的安全造成影响。

测试用例优先级技术则提供了一种优先运行优先级更高的测试用例方法，以便更早地检测错误或提供反馈，从而降低缺陷泄漏到已发布系统的可能性。测试用例优先级技术会重新为测试用例排序，利用软件测试中反馈的各类数据进行分析，更早发现在软件系统中造成缺陷的原因，使开发人员能够更早地开始调试和修复。

测试用例优先级技术主要利用代码覆盖率信息来实现测试用例优先级划分，即可以根据它们在软件的前一个版本上执行的代码语句、基本块或方法的数量，对测试用例进行优先级排序，提高回归测试的有效性。此外，Mirarab和Tahvildari提出了基于贝叶斯网络（Bayesian network，BN）的优先级技术，该技术采用了带有代码修改信息，错误倾向的单变量度量，以及测试覆盖率信息的概率推理算法；Leon和Podgurski提出了结合抽样方法的优先级技术，该方法从基于测试执行文件而分布形成的集群中选择测试用例，在测试用例优先级中利用聚类算法并从聚类中随机选择测试用例进行优先级排序；Yoo通过利用专家知识对测试用例进行两两比较来提高优先级技术的有效性，并将测试用例聚类到相似的组中以方便比较，有效减少人为判断过程中成对比较的数量。

### 18.3.4 测试敏捷化

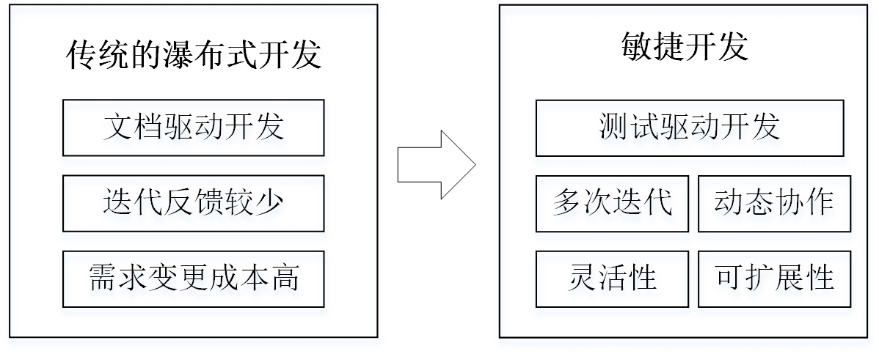
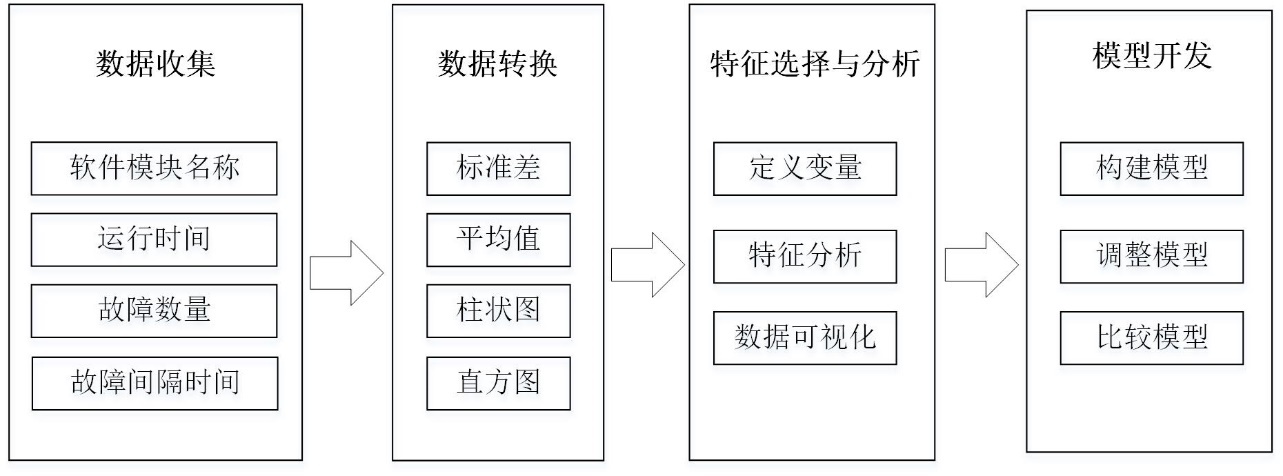


图18.3 传统开发模型与敏捷开发模型比较

对于软件工程来说，开发流程已经从传统的瀑布模型演变为包含数据分析的敏捷开发模型，如图18.3所示。传统的瀑布式开发中测试要晚于需求和开发，发现缺陷的过程较晚，敏捷开发过程中强调迭代、增量的开发过程，以满足软件需求的不断变更，强调测试驱动开发，随时可以交付软件是常态，因此敏捷开发模型相较于传统的瀑布模型更容易控制软件早期的缺陷数量，质量风险不易堆积到最后阶段，同时发现和解决问题缺陷的平均成本也更低。遵循敏捷开发的理论和方法，软件测试同样需要敏捷化和效率化，需要在敏捷开发过程中建立合适的软件测试模型和测试方法。

当前的软件开发过程经常是敏捷和动态的，这会导致软件开发过程中产生并记录大量数据，为了解决软件系统中的故障定位和缺陷预测等问题，软件开发人员和测试人员通常使用数据挖掘技术和统计建模技术建立学习预测模型，组织和分配测试资源。

图18.4 敏捷开发过程中的数据分析流程

敏捷软件开发过程注重项目相关数据收集、数据转换、特征选择与分析、模型开发过程，如同18.4所示。数据收集通常是一个耗时的过程，目标是确保分析所需的数据就位。数据转换是为了改进数据集质量，将使用的变量转换为标准形式，排除不必要的数据，生成快速汇总统计信息，构建柱状图和直方图来帮助理解数据的基数和分布。特征选择与分析定义了用于模型开发的变量，并对不同的特征进行分析，并使用热图、气泡图或其他不同形式来完成数据可视化。模型开发主要包括构建多个模型，比较模型结果和质量。

开发过程中可以使用工具收集每个软件功能模块的名称、运行时间、故障数量在内的重要数据，记录所有软件故障并测量故障间隔时间，进而通过线性回归公式和收集的数据建立回归预测模型，预测系统的稳定性。数据分析人员可以通过分析这些数据来帮助进行产品决策。

### 18.3.5 小结

更敏捷的软件开发，需要减少编码时间，测试时间，需要更快的响应新功能增加和代码变更，在大型的复杂软件上为所有变更代码上执行测试变得较为低效，找到缺陷的概率变的极为困难。通过众测平台通过众包方式降低测试成本，寻求测试方法和主体的多元化是一种比较好的选择；复杂软件在长期的版本迭代过程中通常需要构建自动化框架以支持多轮回归测试，利用回归测试中反馈的各类数据进行分析，可以更早发现在软件系统中造成缺陷的原因；敏捷开发过程中全方位地通过数据收集与建模分析，有助于合理分配测试资源和产品决策。

## 18.4 案例研究

本小节我们将通过一个。。。案例介绍。。。。。

Android等移动设备操作系统的迅速发展使得移动应用迅速发展，目前在Android和苹果的应用商店中有数以百万计的应用。这些应用通过应用商店直接分发给用户，这些用户可以随时对这些应用进行评论，而评论的好坏和评论的数量是衡量应用是否成功的重要参考之一。从业者已经在应用商店中挖掘数据，以解决软件工程中出现的功能、隐私、错误等问题。

Spotify 是最大的音乐流媒体服务提供商之一，截至2022年3月，每月活跃用户超过 4.22 亿，包括 1.82 亿付费用户。Spotify App数据集包含了2022年1月1日至2022年7月9日在Google Play Store上的61594条评论。部分用户通过评论分享其使用经验并给出相应的评分，以此衡量对该App的满意度。App的评分范围为1星到5星。

每条评论主要包括如下几个字段：

①Time\_submitted：该条评论的提交时间，格式为yyyy-MM-dd hh:mm:ss。

②Review：该条评论的具体内容。

③Rating：评分，范围为1-5。

④Total\_thumbsup：点赞数。

⑤Reply：评论的回复。

首先，将评分数据分为正面评论和负面评论两类，可以直接观察用户正负面评论的比例。将评分为1，2，3的评论处理为负面评论，将评分为4，5的评论处理为正面评论，基于评分的简单统计处理完成后得到正面评论和负面评论的比例如图18.5所示，负面评论和正面评论的占比接近。

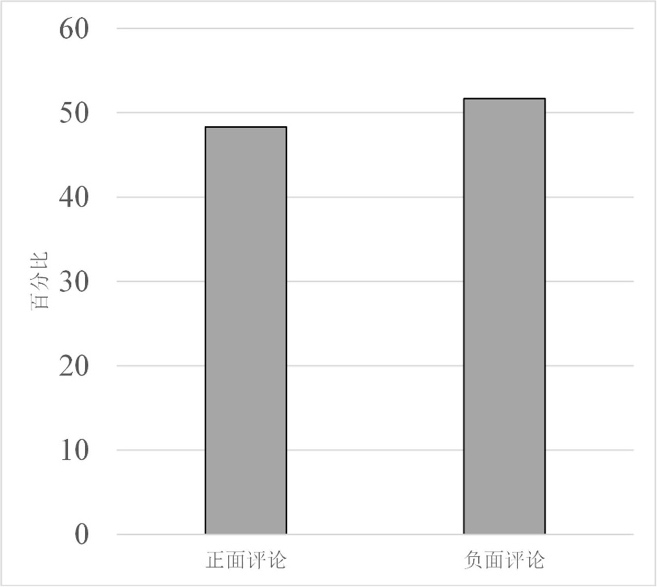


图18.5 正面评论与负面评论占比

通过自然语言技术处理非结构化的用户评论数据，分析和找出用户在正面和负面评论中涉及的主要评论类型，并通过词云将其可视化分析，可以简单获取这款应用在市场上的用户反馈。对用户评论中出现的涉及情绪和类型的关键词进行词频统计，如图18.6所示，在正面的评论中，用户更多地体现出对应用的情绪表达，出现如“喜爱”，“好”，“非常棒”等关键词，同时可以发现用户对播放品质、易用性、播放列表等软件功能有较强的关注度；根据图18.7所示，在负面的评论中，用户对费用，异常，奔溃等有较多的投诉，费用和程序错误占比接近30%以上。观察到这些问题，可以帮助开发人员改善现有的应用。



图18.6 正面评论中的关键词可视化



图18.7 负面评论中的关键词可视化

## 18.5 本章小结

## 习题

1.软件回归测试涉及测试用例（单元测试用例，集成测试用例，系统测试用例）代码、测试用例执行情况记录、测试失败记录等数据，这些数据非常复杂，如何设计一个可视化工具帮助软件测试管理人员方便地分析测试用例执行的数量、测试通过情况、代码覆盖率等指标？

2. GitHub社交化开源开发社区中，存在着大量优秀的开源项目，然而这些项目非常分散，一些优秀的能为开发者提供直接帮助的项目仅仅吸引了很少量开发者的关注。思考如何基于GitHub上用户的个人数据设计一种推荐算法，该算法可以给用户推荐Top-N的感兴趣的项目，同时能够通过GitHub上用户的实际行为记录来验证推荐算法的合理性。

3.在软件开发过程中除了代码还涉及大量的文档，通常这些代码是非结构化的，是否能用自然语言处理技术来解决一些实际问题？

4.软件工程中一般会用到哪些数据采集方法，遇到数据缺失时应该怎么办？

5.从数据科学的视角出发，在软件开发活动中，基于收集的数据可以做哪些预测工作？在软件错误预测过程中，包含错误的软件模块比不包含错误的模块更少，这使得错误标记样本明显少于非错误标记样本，这样的数据不均衡问题如何解决？

## 参考文献

1. W. Maalej, M. Nayebi, T. Johann and G. Ruhe, "Toward Data-Driven Requirements Engineering," in IEEE Software, vol. 33, no. 1, pp. 48-54, Jan.-Feb. 2016.
2. R. Kikas, M. Dumas and D. Pfahl, "Using Dynamic and Contextual Features to Predict Issue Lifetime in GitHub Projects," 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), 2016, pp. 291-302.
3. DiBona C, Ockman S. Open sources: Voices from the open source revolution[M]. " O'Reilly Media, Inc.", 1999.
4. Gu X, Zhang H, Kim S. Deep code search[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 933-944.
5. Shen Q, Wu S, Zou Y, et al. From api to nli: A new interface for library reuse[J]. Journal of Systems and Software, 2020, 169: 110728.
6. Feng Z, Guo D, Tang D, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages[C]//Findings of the Association for Computational Linguistics: EMNLP 2020. 2020: 1536-1547.
7. Guo D, Ren S, Lu S, et al. GraphCodeBERT: Pre-training Code Representations with Data Flow[C]//International Conference on Learning Representations. 2020.
8. Nguyen P T, Di Rocco J, Di Ruscio D, et al. Focus: A recommender system for mining api function calls and usage patterns[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 1050-1060.
9. He X, Xu L, Zhang X, et al. Pyart: Python api recommendation in real-time[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 1634-1645.
10. Carlson R, Do H, Denton A. A clustering approach to improving test case prioritization: An industrial case study[C]//2011 27th IEEE International Conference on Software Maintenance (ICSM). IEEE, 2011: 382-391.