**🧾 简历 – 张杰**

**📌 个人信息**

* **姓名：** 张杰
* **性别：** 男
* **出生日期：** 1988年10月
* **出生地：** 中国上海
* **电子邮箱：** [sagibrant@163.com](mailto:sagibrant@163.com)
* **电话：** +86 136 1167 2073
* **领英：** [linkedin.com/in/sagi-zhang-2845451b](https://www.linkedin.com/in/sagi-zhang-2845451b/)

**🎓 教育背景**

| **时间段** | **院校** | **学位及专业** |
| --- | --- | --- |
| 2011年9月 – 2014年6月 | 华东师范大学 | 硕士/计算机软件与理论 |
| 2007年9月 – 2011年6月 | 华东师范大学 | 学士/软件工程 |

**💼 工作经历**

| **时间段** | **公司** | **职位** |
| --- | --- | --- |
| 2023年1月 – 2025年6月 | OpenText | 经理 I |
| 2017年7月 – 2023年1月 | Micro Focus | 经理 I |
| 2015年11月 – 2017年7月 | 惠普企业（HPE） | 高级软件工程师 |
| 2014年7月 – 2015年11月 | 惠普(HP) | 软件工程师 |

**🧠 技术专长**

* **编程语言：** C++、C#、JavaScript
* **技术领域：** 桌面应用开发、浏览器扩展程序、Web自动化等
* **管理技能：** 六顶思考帽、情境领导力II(SLII)、HBDI等方法论

**🌟 近期亮点**

* **混合文本识别解决方案：**  
  设计并实现了一种混合文本识别方案，显著提升了本地OCR在AI自动化测试中的识别准确率。
* **坐标转换算法：**  
  设计了一种屏幕坐标和浏览器内Viewport坐标的转换算法，兼容不同屏幕与浏览器缩放比例，有效解决了自动化软件在UI检查与高亮显示中长期存在的坐标偏移问题。

**🧾 专利**

* [**Automated Identification of Input Elements in a GUI**](https://patents.google.com/patent/US20220107883A1/en)  
  *发明人:* Peng-Ji Yin, Cheng Hua, Jie Zhang
* [**Recommending Programmatic Descriptions for Test Objects**](https://patents.google.com/patent/US11698851B2/en)  
  *发明人:* Peng-Ji Yin, Cheng Hua, Jie Zhang

**🚀 职业概述**

参与知名旗舰级商业测试自动化平台 UFT/QTP 的核心研发工作，担任资深软件工程师兼团队研发经理。在历经多轮组织重组与跨国并购的背景下，始终专注于产品与技术的深入研究，十余年来持续深耕企业级桌面应用开发、自动化测试技术、本地应用向 SaaS 架构转型、AI 功能落地及性能优化等领域。 擅长交付可扩展的复杂技术解决方案，聚焦大型企业的关键业务需求，主导核心功能设计与实现，积极推动产品创新与技术演进。具备出色的跨团队合作能力，长期与全球研发团队高效协作，确保项目高质量、按期交付并有效支撑业务目标。

**💡 核心优势**

* 🌐🧠 **跨团队协作：**  
  成功协同全球范围内的ALM、MC、SRF和AI团队，交付核心功能。
* ☁️⚙️ **云计算与DevOps：**  
  在SaaS测试执行平台（SRF和Digital Lab）中快速迭代开发新功能，并使用了TypeScript、AWS和Kubernetes技术。
* 🧹⚡ **代码库重构：**  
  通过分析数百万行遗留代码，设计并实现新的内部通信协议，在兼容旧系统的前提下，将UFT Web性能提升至原先的三倍。
* 🤖🔍 **AI功能开发：**  
  设计了一种混合文本识别系统，解决了本地部署中长期存在的OCR准确性问题，性能和准确率远超传统OCR引擎（ABBYY、Tesseract）。
* 📐🖱 **算法设计：**  
  设计了一种屏幕坐标和浏览器内Viewport坐标的转换算法，解决了自动化软件中长期存在的UI检查和高亮情况下的坐标偏移问题。

**🛠 项目经历**

**🧠📦 Unified Functional Testing (UFT/QTP) - *2014.7-2025.6***

作为 **Unified Functional Testing (UFT/QTP)** 的核心开发人员与团队经理，我带领团队持续交付新功能，积极响应客户反馈，并推动产品创新与技术演进。

**UFT**（原名**QTP**）是我所参与开发的旗舰级商业测试自动化平台：

* 🏛️ **历史与演进：**  
  最初起源于1998年的**Astra QuickTest (Mercury)**，在惠普和Micro Focus时期不断演化，发展为支持AI驱动的**UFT One**。现今，该产品更名为**OpenText Functional Testing**，具备AI识别能力和高度脚本自定义的UI自动化和UI测试功能。
* 🧩 **代码规模：**  
  UFT拥有上亿行代码，涵盖C++、C#、Java、JavaScript与VBScript等多种语言，并与Windows和各种主流浏览器的自动化技术深度集成。
* 🌍 **企业采用：**  
  受到全球2800多家企业的信赖，特别是在财富500强的金融、医疗和政府领域中得到广泛采用。
* 📊 **市场份额：**  
  在总规模达222亿美元的自动化测试市场中，UFT占有约2.14%的市场份额，年收入达数亿美元。

**🔹🤖🧠 UFT OCR改进 – *2024-2025***

UFT AI Add-in和UFT TextObject中大量使用了OCR引擎，但是UFT中使用的传统OCR引擎（如ABBYY和Tesseract）在许多客户的生产环境中准确率偏低，无法满足自动化所需的准确性。此外，大部分UFT客户受限于隐私政策或设备环境，大都无法使用基于云的OCR服务。

为应对这些挑战，我们团队评估了五种可能的解决方案，包括切换成EasyOCR或者基于Tesseract重新训练等方案。其中，混合文本识别的方案在初始概念验证(PoC)阶段，因其性能缓慢(每次扫描约20秒)和缺乏单词分段而被否决。

但是我敏锐的意识到该方案的前景，主动参与到了该方案的具体研发中，通过优化算法和查询逻辑，成功将处理时间从约**20秒**缩短到约**2秒**，并为其增加了分词分段功能。混合文本识别的PoC因而在后续的展示中，被一致选中为最终方案，并由我领导团队成功实施。

混合文本识别解决方案智能融合了原始OCR结果以及基于DOM的Web文本信息。该方案面临的主要挑战包括：处理复杂的CSS样式、修正嵌套iframe内文本位置信息，以及尽可能降低对UFT测试脚本执行性能的影响。该混合文本识别方案在网页上可以实现近 **100%** 的准确性，显著提高了离线环境中的自动化可靠性，为大量UFT客户解决了实际问题。

**🔹🌐🧠 UFT Cloud Browser 重构 – *2024***

上一版本的UFT Cloud Browser通过Digital Lab Connector与UFT Web Agent直接进行WebSocket通信。该简化设计虽便于实现，但限制了对UFT AI Add-in所需灵活性的支持，也难以满足客户日益复杂的业务需求。

最大的限制在于：UFT AI Add-in高度依赖COM对象与Windows消息机制，而Digital Lab使用基于Java的跨平台架构，与之不兼容。同时，Digital Lab团队在Windows原生技术方面经验有限，这进一步增加了集成难度。为解决以上问题，我带领团队进行了UFT Cloud Browser的重构工作。

主要贡献包括：

* **重新设计通信框架**：我们对原有通信协议进行了抽象，支持STD I/O、Windows消息和WebSocket三种协议，且保持与现有UFT通信格式一致，确保兼容性。
* **重新实现自动化模拟模块**：基于开源产品，实现新的跨平台的鼠标和键盘模拟模块。
* **重新实现自动化引擎模块**：接收来自UFT的自动化脚本消息后，将其拆解为具体逻辑，并根据任务类型分别交由UFT Web Add-in或UFT AI Add-in模块执行。

最终，我们实现了一个稳定、高效、易扩展的新UFT Cloud Browser，不仅无缝兼容上一版本的全部功能，还良好支持UFT AI Add-in模块，并具备拓展能力，可适配未来的其他平台与UFT Add-in。

**🔹☁️🧪 UFT Cloud Browser (Quick-Win) – *2023***

在StormRunner Functional (SRF)项目终止后，市场对**基于云的浏览器自动化解决方案**的需求依然强烈。然而，由于预算和资源限制，上层经理们开始寻求一种开发开销最小的轻量级替代方案。

作为技术负责人，我与Digital Lab团队合作，设计并交付了一个精简的UFT Cloud Browser替代方案，重用部分旧模块以降低开发成本并加快交付进度。

* **重用**现有Digital Lab的基础设施和UFT Web Agent，最大限度减少新实现开销。
* **重新设计**通信层，直接将基于NativeMessaging的通信改成基于WebSocket的通信，以更好的适配Digital Lab的基础设计。
* **重新实现**部分关键Web自动化功能，使用Extension API，消除了对Windows原生API的依赖，增加了跨平台支持。
* **采用敏捷开发模式**，与Digital Lab团队高频协作，确保每日协同研发、每周与经理同步进度，有效推动项目快速落地。

该项目仅用**4个月**和**4名**开发人员就交付了一个功能齐全的UFT Cloud Browser方案，在研发效率上远超此前历时**3年**、投入约**30名**工程师的SRF项目，充分展现了团队在敏捷开发和资源受限条件下高效交付高价值解决方案的能力。

**🔹🧩🧪 UFT MV3扩展迁移 – *2022–2023***

随着 Chromium 宣布废弃Manifest V2 (MV2)，我主导并完成了UFT Web Agent向Manifest V3 (MV3)的迁移，确保UFT Web Add-in能持续兼容Chrome与Edge浏览器。

主要技术贡献包括：

* **重新设计Extension内的消息架构**: 使用新的Extension API与CustomEvent替代了遗留的window.postMessage通信方式，从而降低了对被测试网页的影响，提升了安全性与可维护性。
* **使用异步操作以减少阻塞**：修改了部分同步代码实现，尽可能减少了UFT Web Add-in脚本执行对客户网页中JavaScript的干预，显著提升了UFT Web Agent的稳定性。
* **重新设计自定义用户脚本执行管道**：绕过了 MV3 更严格的Content Security Policy (CSP)限制，从而使UFT Web Agent能够在MAIN WORLD下执行自定义JavaScript代码，同时也解决了在MV3下点击href="javascript:"的DOM元素，但是其JavaScript代码在CONTENT下被阻止执行的问题。
* **完全向后兼容**：MV3 版本的UFT Web Agent被自动部署至超过**10万**个客户环境，升级后未出现任何重大功能故障，兼容性良好。

这项工作确保了UFT Web Add-in这一核心自动化能力在不断演化的 Web 测试与自动化生态中持续保持先进性与兼容性。

**🔹🧩⚡ UFT Web快速运行模式 – *2020–2021***

随着现代浏览器的演变，UFT Web Add-in的客户不断的从IE迁移向Chrome, Edge和Firefox。然而UFT Web Add-in最初针对IE浏览器进行了优先优化，在其他浏览器上存在明显的性能下降。尤其在处理复杂网页时，UFT Web Testing的执行性能出现明显下降。主要原因在于：IE使用基于C++的插件实现，而其他主流浏览器则依赖基于JavaScript的浏览器扩展程序，同时在具体内存对象上，IE允许直接的对象访问，而在其他浏览器中，则需通过大量额外通信来获取具体的DOM信息。

这个性能问题受到越来越多的客户关注，某大型企业客户曾反馈一个严重问题：在从IE迁移至Firefox后，部分包含Virtual Relation Identification (VRI)的测试步骤，执行时间从原本的**3秒**骤增至超过**3分钟**，严重影响了客户日常工作。

为解决此问题，我带领团队进行了大量的性能优化工作：

* 🔍 深入分析了超过**40万行**核心旧代码，追踪性能瓶颈，挖掘低效架构痛点。
* 🧠 确定关键问题：过多的DOM属性查询消息、冗余的存在性检查，从而导致I/O开销居高不下。
* 🛠️ 重新设计了UFT Web Add-in的通信协议，引入测试脚本上下文，将Query， Action与Report等消息合并为一个新的通用的AutomationStep，形成原子化消息单元，基本保证每一个测试步骤，只进行一次I/O操作，相比每个测试步骤需发送10–60个零散消息的旧方案，新架构显著降低了通信负载，平均提速了近30多倍。
* 🚀 在Chrome, Edge, Firefox上实现平均**3倍**的性能提升，在WebDriver方案中，测试执行性能提升近**5倍**（Baseline测试集运行时间从20分钟缩短至4分钟）。
* 💡 在VRI重度使用的场景中，执行时间从**3分钟**改善到**5秒**。
* 📘 除了技术成果，该项目也显著提升了团队整体技术水平能，使得团队对这部分历史超过15年的UFT Web Add-in代码有了更深刻的理解。

这次深度重构不仅提升了性能，还增强了团队的凝聚力，加深了代码理解，为团队日后处理UFT Web Add-in相关任务奠定了坚实基础。

**🔹🧠🔤 UFT TextObject功能 – *2019***

在UFT中，TestObject通常与特定的自动化技术（如Web、UI Automation、Java、.NET与WPF）紧密绑定。然而，许多大型客户由于安全限制，无法使用UFT所依赖的某些自动化技术，进而导致测试无法正常执行。

针对这种问题，UFT引入了InsightObject，基于一些图像对比的方式来识别UI控件，并通过Device Replay来模拟用户的交互行为。但这种基于图像对比的方案在面对不同屏幕分辨率与缩放比例时，识别准确率显著下降，最终导致其不适用于许多客户的真实环境。

为解决此问题，我提出并实现了一种新的TestObject类型 —— TextObject：

* 基于OCR进行文本识别，通过视觉文本定位UI元素。
* 基于OCR返回的文本在屏幕上的坐标，模拟用户交互操作（如键盘/鼠标），有效绕过传统元素层级结构。
* 在禁止使用Hook和Injection的高安全环境中，TextObject提供了稳健的替代方案，成功规避了传统自动化方式在安全机制下的限制。

这项创新使得许多安全要求高的大型组织能够以最低成本进行功能测试和自动化，无需在安全策略或基础架构要求上做出让步，填补了UFT在无脚本注入环境下的自动化能力的关键空白。

**🔹🧪⚙️ UFT Parallel Execution – *2018***

尽管UFT(QTP)自推出以来深受客户欢迎，但存在一个广为人知的局限——不支持并行测试执行。而这正是客户在加速自动化测试、提升生产效率过程中面临的主要瓶颈。UFT的内存开销（低于200MB）和CPU占用量都极低，但受限于早期架构设计，每个测试实例必须运行在独立的系统用户会话中。

转机出现在一次售前演示中，当时竞争对手展示了可在单台笔记本上并行运行十多个移动测试脚本，而UFT无法支持，此项关键能力的缺失，最终使售前团队未能赢得该笔业务。与此同时，由于SRF产品的回报较低，优先级被降低，借此契机，我对该项对市场拓展极具潜力的功能展开了深入研究。

**技术挑战：**

* UFT核心包含约400万行遗留代码，架构高度依赖单一COM对象与单实例模型。
* 许多模块未设计为支持并发执行上下文或隔离。

**主要贡献：**

* 利用UFT Cloud Execution的经验，成功实现了UFT Mobile Parallel Execution的POC。
* 重构遗留组件，移除不必要的依赖，同时将UFT Mobile Testing相关逻辑隔离到了更独立的模块中。
* 实现单用户会话中并行运行超过10个UFT移动测试脚本，同时保证了脚本运行速度、确保了测试结果的可靠性，并通过移除图形界面依赖，有效降低了UFT Mobile Testing并行运行时的资源消耗。

**成果与影响：**

* 该概念验证获得高级经理认可，并改变了当前大版本的产品策略：产品经理重新调整了路线图，优先考虑UFT Parallel Execution的实现。
* 带领团队，开发并发布了UFT ParallelRunner，支持了UFT Mobile Parallel Execution，后来将UFT Parallel Execution扩展到UFT Web Add-in和UFT Java Add-in。
* 使售前团队在后续交易中可以有更大概率竞争成功。
* QA团队的测试周期显著缩短，自动化测试效率大幅提升。
* 因该高影响力成果获公司认可，晋升为**Manager I**。

**🔹☁️🧱 UFT Cloud Execution on SRF - *2016-2017***

StormRunner Functional (SRF)是一个基于AWS的解决方案，用于在Web和移动应用程序上执行功能测试。该平台提供一个虚拟实验室，支持多种浏览器、操作系统与移动设备。团队可在无需管理底层基础设施的前提下，跨多个环境执行测试任务。

作为UFT后端团队的核心开发人员，我负责设计和实现UFT Cloud Execution解决方案：

* **初始AWS EC2实现**  
  交付了一个在AWS EC2实例上运行UFT测试的工作原型。虽然功能完善，但该方案成本较高，且难以与SRF基于Kubernetes的服务架构实现良好扩展。
* **Windows容器**  
  我们采用Windows容器技术重新设计方案，与SRF原生的容器化与Kubernetes基础架构保持一致：
  + 成本降低**75%**（降至EC2费用的四分之一）
  + 并与DevOps团队合作，在SRF的容器集群中成功实现系统水平扩展能力。
* **跨区域协作**
  + 跨区域协作，与多个国家/地区的工程与DevOps团队紧密配合
  + 在以色列现场向高级管理层演示并验证了方案原型
  + 与以色列DevOps团队直接合作，将Windows容器集成到SRF的方案中
* **业务影响**
  + 使UFT本地客户无缝过渡到SRF云平台进行Web和Mobile测试
  + 提升了平台可靠性和用户满意度
* **附加收入来源**
  + 通过在Docker Hub发布基于Windows容器的镜像，为公司带来了每年200–300万美元的意外新增收入来源。

此次革新不仅彻底改进了UFT Cloud Execution能力，也为未来容器化自动化测试方案奠定了坚实基础。

**🔹📱🔗 HP/HPE Mobile Cente集成 - *2016***

HP/HPE Mobile Cente (MC)是一款由公司内部孵化的移动测试平台，使QA团队能够在真实的iOS、Android及Windows设备上执行自动化测试。

我负责维护UFT与MC之间的集成并开发新功能以扩展我们在测试相关领域的功能。

我主导并参与了多个MC集成功能的开发，其中一项关键增强是引入MC Browser支持，使UFT Web脚本可直接在移动浏览器中运行。

在MC创始人Ameer来访上海期间，我与其深入探讨并协同解决了多个关键架构设计难题。

基于该设计，我实现了连接两个UFT Add-in的桥接服务，使单一的TestObject能够跨UFT Add-in协同运行。该设计也为未来支持Cross Browser Testing提供了良好的架构基础。

**🔹📄🧾 UFT HTML Report Logger - *2015***

UFT使用基于WinForms的应用程序 —— UFT Run Result Viewer (RRV)来展示测试运行结果。RRV在加载与处理大量测试步骤时效率较低，且难以共享测试结果，因为接收方必须安装RRV客户端才能查看。而QA经理及其他相关人员通常未安装RRV，限制了测试结果的可访问性。

为解决此问题，FA和PM决定开发一个新的测试结果报告工具-HTML Report，目标是设计为一个可在任何浏览器中以单个.html文件打开的，轻量、可分享的报告文件。

我与另一位团队成员合作共同实现了该解决方案。我负责开发HTML Report Logger，其主要功能为监控UFT ScriptEngine事件，收集必要的测试步骤信息，并将数据写入结构化JSON文件。后续这些信息会在独立进程中被合并，并生成最终的HTML报告文件。最终的HTML报告通过JavaScript解析JSON数据，并结合CSS动态生成UI，实现测试报告的完整渲染。

该项目使我对UFT Execution Engine有了更深入的理解，并逐步成长为团队内的核心后端专家。

**🔹🖥️🔌 UFT Remote Agent - *2014***

UFT Remote Agent是一个基于DCOM的组件，主要用于从ALM/QC远程触发UFT测试执行。该组件最初使用C++编写，随着时间推移演变为一个包含9000多行代码的巨型类，结构单一，存在大量全局变量且注释缺失，维护难度极高。由于代码结构混乱，系统难以维护，频繁出现客户报告的执行错误。

为解决上述问题，架构师决定以C#重构该组件，而我则承担了该项目的具体实现任务。

我深入分析了原始代码库，并与ALM团队密切合作，充分理解UFT与ALM的集成点。基于此，我根据UFT的功能领域分离职责，重新设计了类结构。同时，我采用WPF技术重构了Remote Agent的用户界面，界面风格更加简洁现代，用户体验显著提升。

该项目的一个关键挑战是性能：C#实现的初始版本在性能上逊于原始的C++实现。

我使用Visual Studio Profiler对其进行了为期一个月的深入性能分析，并实施多项优化措施，具体优化包括引入异步处理机制，以及将UFT AOM Agent动态加载为程序集，以显著减少跨进程通信开销。

最终，新的UFT Remote Agent不仅达到了与旧版相当的性能水平，还提供了更简洁的用户界面，并大幅提升了系统可维护性。该项目也为我赢得了职业生涯中的首个**团队之星**奖项，得到了管理层的高度认可。

**🌍 其他项目**

**🔹🍽️💳 Botion餐厅系统 • *2010年8月***

我在加入初创公司（xpfood.com）期间，参与研发了Botion线下餐厅系统，涵盖打印票据、会员卡识别等功能，提升了整体点餐与订单处理效率。