# Roadmap for Research in Software Services

Satish Chandra[*]
Samsung Electronics
San Jose, CA
schandra@acm.org

Saurabh Sinha
IBM Research
Bangalore, India
saurabhsinha@in.ibm.com

Vibha S Sinha
IBM Research
New Delhi, India
vibha.sinha@in.ibm.com

Krishna Ratakonda
IBM T J Watson Research
Yorktown Heights, NY
ratakond@us.ibm.com

## ABSTRACT

Abstract goes here.

## 1. INTRODUCTION

Software development is most visibly associated with companies that manufacture software products. Corporate names such as Microsoft, Google, IBM, SAP, Oracle, and so on are well-known software companies. These companies offer several software products to individual consumers and businesses.

In the broader information technology ecosystem, software development is also associated with "services companies". Instead of offering software products, these companies develop software for other businesses, i.e. they offer software development as a service.[1] Examples of service companies include Accenture, Cap Gemini, Infosys, Atos, and so on; in addition, companies such as IBM and HP have a large services business in addition to their software and hardware businesses.

There are two main drivers for demand for software development as a service. First, most businesses prefer to focus on their core competence, rather than develop and foster in house competence for software development. For example, every company in the financial sector relies heavily on information technology, but information technology is not the basis on which they differentiate themselves in that sector. They prefer not to run large in-house teams that have expertise in ever growing set of software technologies, when they can get easily procure IT services on demand from service companies. Obviously, some number of in-house staff is always needed, but the bulk of work can be outsourced to service companies.

Second, although a lot of business software functionality is now available in pre-packaged applications, it is not realistic to run an enterprise entirely with off-the-shelf software. Significant customization and systems integration is required to run any large enterprise, even if substantial building blocks are available off the shelf. Moreover, custom software must be maintained to incorporate new business needs or to accommodate legislative compliance, and continually upgraded to newer technologies such as cloud. Service companies have deep industry as well as technology expertise to cater to these needs. The fact that services companies can also deliver services cost effectively using a large pool of global resources is an added benefit.

Like software product companies, services companies employ software engineers in large numbers, and take on long running software projects. Given the scale of operations, many of the usual challenges in software engineering that are well known from product development context also apply in the context of services. However, the two kinds of businesses have some differences, due to which certain challenges become more pronounced for services companies.

The most salient of differences is this: product companies compete in the market they serve on the basis of the features offered in their product and how well it anticipate their customers' needs. Thus, their main focus is on innovation in identifying features that their customers would be willing to pay for, and bringing out a product that offer those features before their competition does. By contrast, services companies do not compete on the basis of the features of any product; rather they compete in terms of the "quality of service" that they offer to the business that purchase their services. Here, quality of service informally means several things that one may associate with any kind of services business, even outside of information technology: can the service provider deliver on their contract on time, with acceptable work quality, and at a competitive price. Therefore, the innovation in service companies has to do with techniques to carry out projects reliably, on time, with acceptable quality, and manage the cost.

The second important difference is that in product companies, copies of the same product is sold to a large number of customers, and typically the transaction is one time payment of license fee. A services company provides services to a much smaller number of clients, and the transaction is an ongoing relationship rather than a one-time payment of license fee. This has an important bearing in deployment of innovation. A product company must be extremely careful in choosing what innovations to productize, because they cannot afford to get it wrong; however, if successfully shipped, the innovation automatically makes its way to a large number of users. In a services company, the innovation is somewhat less risky, but it does not scale automatically: it might take just as much work in rolling out an innovation to the next client as it did for the first

[*]This author was formerly at IBM.
[1]Note to be confused with software-as-a-service, which is a way to deliver software products.

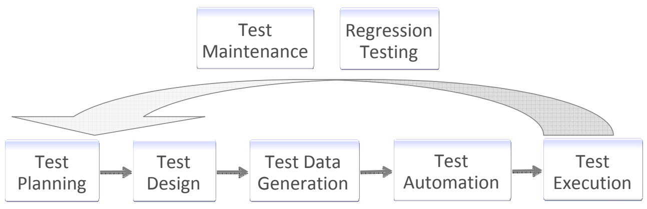

**Figure 1: Typical testing activities performed in delivery of testing services.**

client.

A third difference is in the timeline of engagement with the customer.

In this paper, we consider selected directions for research in software engineering that would impact how software services are delivered. The list of topics we have chosen is influenced by our work during the past few years at IBM. There are numerous other topics, e.g. legacy transformation and cloud migration that are very promising areas of research.

## 2. RESEARCH DIRECTION: TESTING

Within the increasing demand for software development as a service, *software testing as a service* has seen significant growth and adoption in its own right, often as a separate service that is independent of development activities. In fact, according to a 2006 survey,[2] software testing was the second largest outsourced software-engineering activity after coding: 81% of the 200 industrial practitioners, who participated in the survey, stated that they outsource software testing. Given this trend, services companies now routinely offer services that focus exclusively on testing activities.

The nature of the activities performed in a testing-services engagement can vary from client to client; but, typically, the scope of work includes activities, such as test design, test-data creation, test automation, test execution, and test maintenance. Figure 1 presents an overview of the commonly performed activities in the delivery of testing services. A delivery engagement might involve any subset of these activities. For example, delivery for a client might involve only manual test execution, or the scope of work may include test automation and test maintenance as well. In other cases, the scope of work may be even broader, encompassing test design and/or test planning. The nature of the test cases can also vary from functional tests to tests that focus on validating non-functional system properties, such as performance and security.

The activities shown in Figure 1, of course, pertain to testing in general (whether performed in-house or in an outsourced manner), but there are factors that can add unique challenges in the setting of testing services. For instance, many of these activities can require specific skills in testing techniques/tools or coding expertise, which the average testing practitioner involved in service delivery may not possess. Second, some of the activities may need to be carried out in strict time-bound cycles and client-controlled test environments. Finally, the myriad of technologies that must be accommodated in the contexts of the IT systems of different clients adds another layer of complexity for service companies.

### 2.1 Test Planning

### 2.2 Test Optimization

Optimization of test cases using techniques such as combinatorial test design.

[2]http://www.drdobbs.com/architecture-and-design/cheapers-not-always-better/184415486?requestid=247829

### 2.3 Test Data Generation

Enterprise applications are typically database-oriented, in that they make queries over databases and the program flow depends on the data retrieved as a result of a query. In order to test whether the application satisfies its requirements, sufficient amount of test data must be made available in the database. Consider, for example, a banking application, in which one of the requirement to exercise is that if a customer makes a certain fixed deposit, and the person is a senior citizen, and also holds a checking account in the bank, then an addition 0.25% is paid as interest. To test that this situation is handled correctly in the application, there should exist a customer in the database such that his or her age qualifies the person as senior citizen and the person also has a checking account. Notice that typically, customer data and account data would be maintained in separate database tables. Typically business requirements are complex, and there need to be specific entries in multiple tables for a scenario to work out.

When a client gives a testing contract, the client may expect the service provider to exercise all the business rules, but may not give enough sample data in the database to be able to exercise these rules. It then becomes the responsibility of the service provider to figure out the requisite test data to be present in the pertinent tables.

Randomly generated test data cannot be expected to suffice for enterprise applications with complex rules. Also, systematic test generation approaches based on program analysis cannot be expected to tackle enterprise application, which use a mix of multiple language and database technologies in their implementation. The database community has done some work in creating test data for SQL queries, and that may be relevant; however, as mentioned before, enterprise applications are implemented with a mix of technologies.

We see a big opportunity for research contributions in this subarea. How to take a specification of the application as testing criteria, and automatically populate database that would enable various scenarios that fulfill the test criteria to be exercised.

### 2.4 Test Automation

The functional test cases created during test design are often stated informally in natural language. These tests are known as *manual test cases*, which consist of a sequence of steps, intended for execution by a human on the application user interface. Test automation is the activity of converting these test cases into test scripts or programs that perform the test steps mechanically. After creation, the automated test scripts can be executed in regression cycles without any human intervention.

Although test automation is desirable (*e.g.,* for efficient and predictable test executions) and sometimes necessary (*e.g.,* to meet time-constrained regression schedules), creating *robust* test scripts that can be used repeatedly in regression cycles is expensive. In essence, creating such scripts is time-consuming, involving a significant amount of coding, and requires expertise in test automation tools (*e.g.,* [3, 4, 5]). In the context of service delivery, this poses the unique challenge: how can change-resilient test scripts be created efficiently by delivery personnel who may not possess good programming skills or deep tool knowledge?

While script robustness is certainly relevant in the context of evolution of the application under test or its persistent data, it is also pertinent when a script needs to be executed on different platforms, execution environments, or application variants. For example, functional testing of web applications needs to be performed on different browsers to detect potential cross-browser incompatibility defects [8, 10, 11, 17]. Similarly, in the context of mobile applications, tests for a native mobile application would need to run the variants of the application for different platforms (*e.g.,* Android, iOS, BlackBerry). Ideally, in such situations, a robust test script should be agnostic to browser/platform-specific differences unless, of course, those differences are symptoms of application defects.

### *Research Problem: Tackling Script Fragility*

The core problem that needs to be addressed in attempts to solve the script-fragility problem is how to locate user-interface (UI) elements in a reliable manner. In general, there are two broad categories of techniques for locating UI elements. The first category relies on the structure of the internal representation the UI [3] and on internal attributes, such as IDs or names, of UI elements/widgets. Sample test automation tools in this category include IBM Rational Functional Tester [4], HP QuickTest Professional (QTP) [3], and Selenium [5]. Over-reliance on internal structures and attributes can make scripts fragile (*e.g.,* if the structure changes often or the attributes are generated dynamically) and, in particular, unreliable for execution across browsers and platforms.

The second category of tools rely on image processing to locate UI elements; sample tools in this space include Eggplant [1] and Sikuli [7, 22]. These tools record a test script as a sequence of actions performed on images. During automation, the tester grabs a portion of the screen around the element of interest, specifies the "hotspot" in this portion (the location where, *e.g.,* a click action would be performed), and specifies the image-similarity threshold to be used during test execution to locate the element. By being independent of internal structure and attributes, such tools are resilient to changes in the internal structure/attributes, but they are fragile in the presence of differences in visual rendering, say across browsers or application variants.

A third, and less-common, alternative to these approaches is to associate with a UI element a label in the vicinity of that element, and refer to the element via its associated label. This approach, which is used in the ATA tool developed at IBM Research [19, 20, 21] and also supported by QTP [3], can overcome the limitations of approaches that rely on internal structure or image processing. But, the currently available implementations of these approaches perform simple label associations (based on visual proximity only) and are inapplicable when ambiguous labels exist. Development of more powerful and generally applicable techniques for label association would be a fruitful direction of research.

These approaches have different strengths and weaknesses; it is quite possible that no one approach turns out to be the best in all circumstances. For example, an image-based technique may outperform an internal-structure-based technique for cross-browser test execution; but, the converse would likely be true for test execution across internationalized variants of a web application. In practice, the choice of the technique would have to be tailored to the requirements of testing. Development of tools that combine the approaches, and rigorous empirical studies that evaluate the resiliency of these approaches for different types of testing would be useful.

### *Research Problem: Automated Test Adaptation*

Test automation in the space of mobile applications faces the formidable challenge of high diversity, which occurs along multiple dimensions: many platforms, devices, web browsers (in the case of mobile web applications), and application variants. For instance a mobile application can have native, hybrid, and web variants; moreover, in the case of native and hybrid applications, different platform-specific and device-specific variants can exist.

[3]In the context of web applications, this internal representation is the Document Object Model (DOM).

Efficiently testing an application across different combinations requires automation, but the automated test scripts must be self-adaptive. Native or hybrid mobile applications typically have variants that execute on different platforms (*e.g.,* Android, iOS, and BlackBerry) and different types of devices (phones or tablets). For some application variants, the UI layout and flow may be exactly the same, in which case a test script that has been recorded in a platform-agnostic script notation (*e.g.,* [2]) on one application variant can be executed, without modification, on another variant. But, in more complicated cases, the UI layout and flow for the same scenario may differ across variants—as an archetypal example, the phone and tablet variants of a mobile application would typically support many common scenarios that are realized via different sequences of UI events. In such instances, can a script automated on the tablet variant be automatically adapted for execution on the phone variant? We believe there is scope for the development of new techniques that perform such adaptation, while ensuring that the intent of the test cases are preserved.

### *Research Problem: Automated Test-Script Synthesis*

The problem of converting manual test cases to automated test scripts can be looked upon as an instance of program synthesis [15]. Automated program synthesis addresses the problem of discovering a program that realizes a user intent. There are three dimensions of the synthesis problem: the form of user intent, the search space of programs, and the search technique [15]. For instance, the user intent could be stated in different forms such as, natural language, input-output examples, logical relations between inputs and outputs, and demonstrations. In test automation, user intent is specified in the manual test steps, and the end goal is an automated script that realizes that intent.

Recent work [20] has attempted to automate the creation of test scripts from manual test steps, using a combination of natural-language processing and dynamic exploration of multiple flows via backtracking to search for the correct test script from the space of possible scripts. But, there are practical limitations to that technique. First, dynamic exploration of alternative flows in the application UI via backtracking can encounter problems, such as undoing persistent state updates during an exploration or finding disable UI elements that limit the scope of exploration. Addressing such problems is important for effective exploration of the space of possible test scripts. Second, test automation can sometimes require the coding of custom functions; for example, a verification step in a test might require checking that the values in a drop-down list are sorted or that some value exists in a particular cell of a table. Currently, such code must be written manually. Automatic synthesis of such code based on appropriate specification of user intent would go a long way in automating the overall creation of test scripts.

## 2.5 Test Maintenance

Although developing scripts in a change-resilient manner, as discussed in Section 2.4, can reduce test-maintenance effort, it cannot totally eliminate maintenance. In particular, refactoring of the application GUI, such as splitting a web page into multiple tabbed pages, can break the flow of a test [21], which would occur irrespective of how robust the mechanism for locating UI elements is. Such changes require the test flow to be repaired, involving addition or deletion of test steps—automatically performing such repairs is beyond the capability many of existing GUI test repair techniques [9, 14, 16]. Proposals GUI-refactoring-driven test repair [12] accommodate certain types of GUI refactorings (*e.g.,* replacing a radio box with a drop-down list), but not general flow repair. Recent research has seen the development of repair techniques for broken GUI flows [23], addressing some of the limitations of the older test-repair techniques, but there are opportunities for more research contributions on this topic.

## 3. KNOWLEDGE MANAGEMENT

Software development is inherently a knowledge intensive activity. Software designers and developers leverage their software development skills along with knowledge about the domain, past experiences and knowledge of team members, to solve the current problem at hand, be it implementing a new feature/application or solving a bug. Individuals are adept at managing their own knowledge and drawing upon it when a new situation comes in [6]. For example, a developer who has been assigned a bug on a system (s)he has implemented would typically know where to start investigation from because of prior knowledge about the system. A project manager working long enough in the product would know the primary risks associated with the project, fault prone areas and so on. In small, colocated teams too, knowledge management is not a big challenge. Who is an expert on what part of the system is typically known. New team members coming in would use informal channels of communication to figure out who are the experts on the system. If need be, individuals can tap into the knowledge of these experts to quickly address task at hand.

Knowledge management starts to become a challenge when team size increases and/or teams start getting distributed geographically. In large teams or distributed environments, the requisite knowledge of the system—expertise, best practices, insights, ideas and so on—is spread across multiple people, locations and even organizations (in cases such as outsourcing) [13]. In such scenario, even for a single project a formal knowledge management system needs to be in place that stores all data for a project—be it requirements, assumptions, solution documents, design rationale, bugs, code artifacts and so on. Any project member should be able to refer to this project database to understand how project evolved, who are the experts in the system and given a task at hand search for relevant artifacts and activites from past. Now consider the case for services organizations such as IBM, which are developing and managing software systems for multiple clients. Information on how a requirement to develop a payroll system for client A could be applicable for another client B also. Knowledge of how a leave related issue implemented on SAP HCM was resolved for client C, could we leverage when solving a similar issue for client D. A cross-projects/cross-clients knowledge repository is like the organization memory [18] for services organizations and provides oppurtunities to leverage learnings from the past to achieve greater efficiency and efficacy in present and future projects. Figure 2 illustrates how knowledge management needs in services organizations, evolve from personal knowledge management to intra-project and inter-project KM due to increased team sizes, geographic distribution and multiple client engagements.

The rest of this section is organized as follows. We first describe why knowledge management is needed in services organizations, followed by design considerations for building an effective knowledge management system. Finally, we present our experiences with deploying two knowledge management systems within IBM.

### 3.1 Need

- Do things faster
- Do things with less skilled resources
- Handle resource churn
- Promote reuse across engagements

### 3.2 Design Considerations for Effective Knowledge Management

- How and what to harvest: personalization versus codification, manually curated versus auto-harvested
- Features needed from KM system: search, topics, summarization
- How to promote usage: incentives, quality versus quantity

### 3.3 Examples

- Consultant's Assistant - KM at solution level
- Wisdom - KM for troubleshooting

## 4. TROUBLESHOOTING

One of the common forms of service engagements is application maintenance, where the expectation from the service provider is to take over a client's custom application and handle service requests for it. Service requests come in the form of trouble "tickets": users of the applications can raise a "ticket", logging a problem they have been experiencing. This is similar to bug repositories such as Bugzilla, in which users of an open-source software can enter the details of a problem that they experience. The main difference is that in typical software development in open-source communities, there is no obligation on the part of software maintainers to address the defects in a timely fashion. Likewise, even in a product setting, the development team can prioritize which defects they are going to address first. In service context, the service provider is supposed to resolve the ticket in a timely manner, often under a service-level agreement. For example, a critical bug must be resolved within 6 hours, at risk of financial consequences for the provider.

Clearly, the efficiency which the service provider can resolve these tickets is a differentiating factor. For this reason, troubleshooting is a important research area for software services.

Ticket resolution takes place in stages. At the front, customer facing level is what is called L1 support, which simple acknowledges the issue to the customer and assigns it to an internal queue. In simple cases, such as request for information, L1 support can provide "help desk" features. L2 support comes in when the ticket requires technical expertise to answer, but usually resolving the ticket entails no more than advising the customer of the right configuration, or a workaround, etc. Finally L3 support handles real defect, and resolving them typically involves fixing the code.

(We can use a picture here.)

L1 and L2 support can benefit from the techniques that we reviewed in the section on knowledge management.

Here we describe some of the challenges in L3 ticket resolution.

In the purest form, the troubleshooting problem (a.k.a. debugging) is this:

## References

[1] EggPlant. `http://www.testplant.com/products/eggplant`.

[2] Perfecto Mobile ScriptOnce API. `http://www.perfectomobile.com/articles/scriptonce-api's-page`.

[3] HP QuickTest Professional. `http://www8.hp.com/us/en/software-solutions/software.html?compURI=1172122&jumpid=reg_r1002_usen#.T-amcRfB-Gc`.

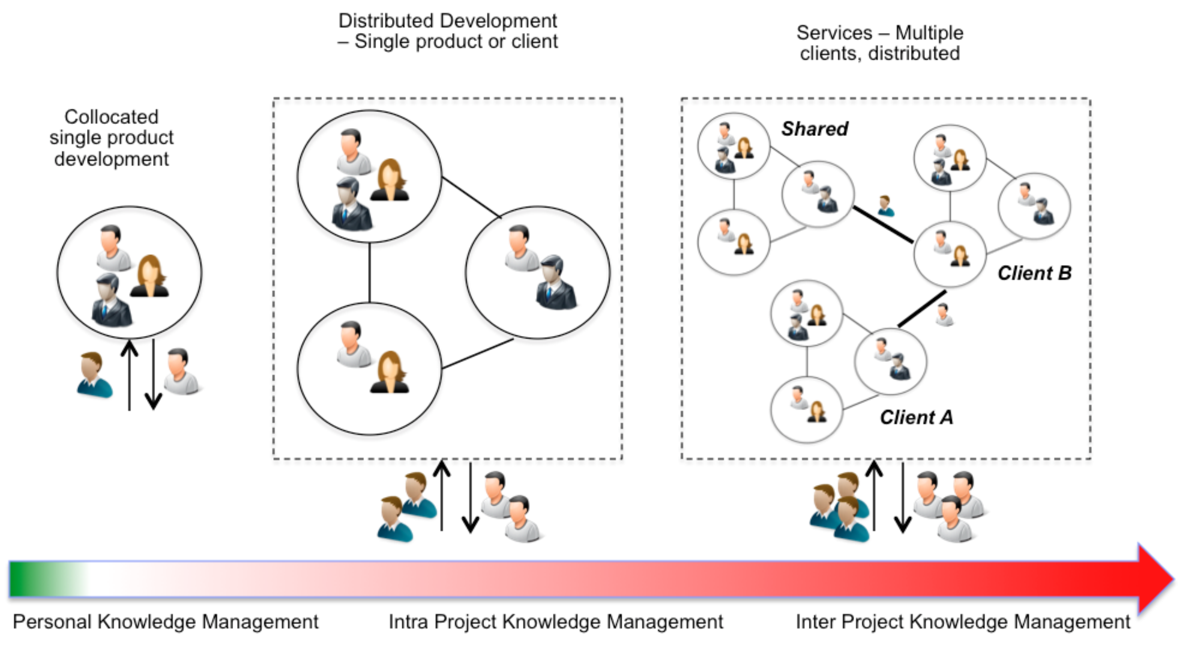
Distributed Development – Single product or client
Services – Multiple clients, distributed
Collocated single product development
Shared
Client B
Client A
Personal Knowledge Management
Intra Project Knowledge Management
Inter Project Knowledge Management

**Figure 2: Team structures and knowledge management needs**

[4] IBM Rational Functional Tester. `http://www-01.ibm.com/software/awdtools/tester/functional`.

[5] Selenium. `http://seleniumhq.org`.

[6] H. Bruce. Personal, anticipated information need. In *Information Research, 10(3) paper 232*, 2005.

[7] T.-H. Chang, T. Yeh, and R. C. Miller. GUI testing using computer vision. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 1535–1544, 2010.

[8] S. R. Choudhary, H. Versee, and A. Orso. WEBDIFF: Automated identification of cross-browser issues in web applications. In *IEEE International Conference on Software Maintenance*, pages 1–10, 2010.

[9] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso. WATER: Web Application TEst Repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, 2011.

[10] S. R. Choudhary, M. R. Prasad, and A. Orso. CrossCheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, pages 171–180, 2012.

[11] S. R. Choudhary, M. R. Prasad, and A. Orso. X-PERT: Accurate identification of cross-browser issues in web applications. In *Proceedings of the 35th International Conference on Software Engineering*, pages 702–711, 2013.

[12] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, and M. Pezzè. Automated GUI refactoring and test script repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, pages 38–41, 2011.

[13] K. C. Desouza, Y. Awazu, and P. Baloh. Managing knowledge in global software development efforts: Issues and practices. *IEEE Software*, 23:30–37, 2006.

[14] M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving GUI-directed test scripts. In *Proceedings of the 31st International Conference on Software Engineering*, pages 408–418, 2009.

[15] S. Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, pages 13–24, 2010.

[16] A. M. Memon. Automatically repairing event sequence-based GUI test suites for regression testing. *ACM Transaction Software Engineering and Methodology*, 18:1–36, November 2008.

[17] A. Mesbah and M. R. Prasad. Automated cross-browser compatibility testing. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 561–570, 2011.

[18] E. W. Stein. Organization memory: Review of concepts and recommendations for management. *International Journal of Information Management*, 15(1):17–32, 1995.

[19] S. Thummalapenta, N. Singhania, P. Devaki, S. Sinha, S. Chandra, A. K. Das, and S. Mangipudi. Efficiently scripting change-resilient tests. In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering, Tool Demonstrations Track*, 2012.

[20] S. Thummalapenta, S. Sinha, N. Singhania, and S. Chandra. Automating test automation. In *Proceedings of the 34th International Conference on Software Engineering*, pages 881–891, 2012.

[21] S. Thummalapenta, P. Devaki, S. Sinha, S. Chandra, S. Gnanasundaram, D. D. Nagaraj, and S. Sathishkumar. Efficient and change-resilient test automation: An industrial case study. In *Proceedings of the 35th International Symposium on the Foundations of Software Engineering, SE in Practice Track*, pages 1002–1011, 2013.

[22] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: Using GUI screenshots for search and automation. In *Proceedings of the 22nd Symposium on User Interface Software and Technology*, pages 183–192, 2009.

[23] S. Zhang, H. Lü, and M. D. Ernst. Automatically repairing broken workflows for evolving gui applications. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 45–55, 2013.