The Development of an Automated Testing Framework for
Data-Driven Testing Utilizing the UML Testing Profile

James Edward Hearn

Dakota State University
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK

Table of Contents

1. Introduction .................................................................................................................................................. 3
  1.1 Background of the Problem ...................................................................................................................... 5
  1.2 Statement of the Problem .......................................................................................................................... 6
  1.3 Objective of the Project ............................................................................................................................ 7

2. Literature Review ........................................................................................................................................... 8

3. System Design and Methodology .................................................................................................................. 20
  3.1 Action Design Research ............................................................................................................................ 20
  3.2 Test Framework Development ................................................................................................................... 23
  3.3 Data Collection and Gathering Techniques ............................................................................................... 26
  3.4 How ADR and the Test Framework Development Principles Were Applied ....................................... 27
  3.5 ADR Iteration by Iteration ........................................................................................................................ 29
    3.5.1 Iteration I: Crafting the artifact .......................................................................................................... 29
    3.5.2 Iteration II: Crafting the artifact ......................................................................................................... 51
    3.5.3 Iteration III: Crafting the artifact ....................................................................................................... 68

4. Formalization of Learning and Generalization of Outcome ........................................................................ 79

5. Project and Methodological Reflection ....................................................................................................... 83

6. Discussion ....................................................................................................................................................... 86

7. Conclusion ...................................................................................................................................................... 89

Appendix A ...................................................................................................................................................... 92

Appendix B ...................................................................................................................................................... 106

Appendix C ...................................................................................................................................................... 149

References ....................................................................................................................................................... 162
Abstract
The development of increasingly-complex Web 2.0 applications, along with a rise in end-user expectations, have made the testing and quality assurance processes of web application development not only an increasingly-important part of the SDLC, but have also made these processes more complex and resource-intensive. One way to effectively test these applications is by the implementation of an automated testing solution. The difficulty, though, lies in how to best go about creating such a solution. The use of a framework is shown to help, by decreasing the amount of duplicate code and maintenance required, and increasing the amount of separation among the various elements of the testing solution. This research examines the use of the UML Testing Profile (UTP), including the use of UML diagrams, in the creation of such a framework. Using an Action Design Research methodology, a framework is developed, used in an organizational context, and evaluated, over the course of multiple iterations. Design principles are distilled from these iterations, and their impact on the larger context, the delivery of a robust application that meets end-user expectations, is examined.

Keywords: testing, automation, web applications, UML, Testing Profile, UTP, frameworks
Acknowledgement

I would first like to thank my thesis advisor, Dr. Cherie Noteboom, of the College of Business and Information Systems at Dakota State University, for giving me both the structure required to complete this research but also the flexibility for this research to be my own experience and work. Additional thanks to Dr. Surendra Sarnikar for guiding my initial planning activities and assisting me with defining the scope and intent of this project.

Many thanks also to Dr. Ronghua Shan and Dr. Chris Olson, both of the College of Business and Information Systems at Dakota State University, for their help and assistance throughout my time as a student at Dakota State University, as well as serving on the faculty committee for this research. Through them I have learned many valuable lessons that go well beyond the classroom.

Finally, I would like to express sincere gratitude to my wife and child for providing me with constant support and encouragement throughout my years of study and research. This goal would have never been reached without them.

James Edward Hearn
1. Introduction

1.1 Background of the Problem

Web-based, enterprise-level applications that provide both flexibility and reliability are increasing in usage and complexity in order to meet increasing customer demand. Customers of software development organizations are asking more from the applications developed for them than ever before: more data to collect, validate, and process, higher levels of system performance required, and increasing amounts of customization and user interface flexibility are being expected out of modern applications. Scalable, instantly-accessible multi-user, multi-component, multi-tier systems are being required to perform mission-critical tasks in an environment of continuous delivery and operation, with an expectation of zero down time and easy maintenance and administration.

These advanced applications, like any advanced system, do not survive the development cycle without the introduction of defects. In order to ensure these modern applications meet user requirements while also being as defect-free as possible, software development teams are turning to automated systems for testing the software they develop. The use of automated testing allows for more testing to be done, increasing the amount of code covered and scenarios tested, as well as finding defects in the application sooner in the development process, and also allows manual testing operations to focus on particular problem areas and more in-depth coverage of system performance. For these test automation platforms, a framework for these platforms is suggested. The use of a test automation framework has been shown to improve the maintenance effort of the tests performed, making it quicker and easier to update a test when a change in the application under test is made, resulting in automated tests that are robust and consistent, and thereby more valuable to the organization.
How this framework should be constructed is a crucial question, as the answer will dictate the entire basis of the testing platform and future development of the tools. A good reference for the construction of this framework is the UML Testing Profile (UTP). As a UML profile, the UTP extends the functionality of UML into a particular domain, and provides customization options for the problem domain. The UTP is language- and implementation-independent, and supports the design, visualization, specification, analysis, construction, and documentation of testing artifacts.

1.2 Statement of the Problem

A proper test automation solution reduces the time and resources required to adequately test software, while also increasing the level of quality of the application under test (Dustin, Rashka, and Paul, 1999). Bad test automation designs result in additional time spent maintaining the automation code, which is a detriment to modern, agile processes used in developing software, and, more importantly, lead to circumstances where testing is inadequate, increasing risk for system end-users. Knowing the possible gains from good design and the possible losses from poor design, it is important to look at the process of test automation construction in order to determine the best course of action as to ensure that all goals are met. One technique proposed by the Object Management Group is an extension of the Unified Modeling Language (UML), known as the UML Test Profile (UTP). The UML Test Profile provides extensions to the original UML specifications that support test documentation, design, specification, and construction. This project will therefore answer the question: How can a test automation solution designer use the UTP to design test automation systems that effectively test their systems, thereby increasing the robustness of the applications being tested?
1.3 Objective of the Project

The ultimate purpose of this research is to help identify how test automation engineers can implement the UML Testing Profile during the process of creating a test automation framework in order to ensure that the overall automated test solution

- assists in finding and preventing defects,
- ensures that the software delivered meets business requirements, user requirements, and specifications.
- helps end-users gain confidence in the quality of system.

In order to achieve this purpose, this research is undertaken with several goals in mind. One goal will be to use the concepts found in the UTP to formulate a set of design principles for test automation framework creation that reflect the current theory and research intent. Another goal for this research will be to demonstrate that using the UTP for creating an automated testing framework results in a measurable level of test coverage and testing efficiency. Meeting these goals will be done by meeting the following objectives:

- Build an automated test framework solution, incorporating the concepts found in the UTP, as shaped by organizational use
- Concurrent evaluation of the resulting ensemble artifact in an organizational setting for value and utility in an authentic environment
- Identify the areas of automation framework creation where the concepts found in the UTP can be positively implemented
- Draw out design principles and knowledge gained through the research proves, via the lens of the design research outcomes
2. Literature Review

Web 2.0 Challenges

The advent of Web 2.0 has altered the way users interact with enterprise-level software and the way enterprises incorporate software into their business. Tim O'Reilly's definitional article (2005) describes the shift that had commenced around that time, and points to a number of factors that have led to the proliferation of Web applications for business use.

The Internet has become its own platform, upon which developers are developing next-generation Web applications and rich Internet applications. The evolution of the Web as a platform allows developers to think about applications as browser-based services, independent of user operating systems. Some even argue that the Internet browser is becoming the next operating system, capable of delivering applications and extending architectural capabilities in much the same way traditional, desktop operating systems have worked for many years (Wayner, 2013; Garaizar, P., Vadillo, M. A., López-de-Ipiña, D., & Matute, H., 2012). Using the Web as a platform, instead of a specific operating system or specific browser, frees developers from the constraints imposed by utilizing operating systems and browsers, allowing the large amounts of interoperability organizations require from their Web applications. Thinking of the Web as a platform also highlights the network effects possible with Web applications, where the value that the Web application provides the organization using it increases as the number of individuals using the application increases.

The use of Web applications, distributed via the platform of the Internet, has also increased because organizations have discovered how Web applications can harness the collective intelligence of their user bases. One of the primary hallmarks of modern Web applications is the movement from a "read-only" mindset to a "read/write" mindset, allowing and encouraging the same users that consume a site's content to also add content of their own
(DiPietro, 2012; Nielsen, 2012). The use of Web applications to facilitate collective intelligence has been used to improve decision-making, through participative social interactions, and Web applications can create synergy among individuals, as the source of experience and ideas, and computers. This trait not only allows individuals to connect with each other, but it also provides a means for storing and searching vast amounts of data, increasing the collective intelligence of the user community (Deed, C., & Edwards, A, 2013).

In order to harness these vast amounts of data and expand the use of Web applications for collective intelligence, specialized databases have to be implemented. Next-generation Web applications cannot function without next-generation databases, and recent trends in big data highlight the centrality of databases in the upward trend of Web application use. The advent of NoSQL brings with it an increase in scalability not seen in traditional relational databases, built to take advantage of cloud computing and distributed technologies, as well as provide an increase in speed and reliability (Raj and Deka, 2014). Database implementations like Cassandra and MongoDB are designed to handle large volumes of data that can process tens of thousands of features quickly and seamlessly (Kozielski, 2015). These design implementations are required for handling the large amount of user data that can be generated by a virtual community of users-as-creators, and extend the capabilities of Web applications beyond users, to include additional technologies, as seen by the rise in the Internet of Things (IoT) (Simon & Covic, 2015).

The environment in which Web applications are developed and released has changed the fundamental nature of application delivery. The advent of cloud technology has created a paradigm where services can be increasingly delivered online, and has enabled the growth of Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Structure (IaaS), and other service-oriented technologies (Ranajn, 2015). These technologies lead the way for IT
businesses to shift from delivering products to delivering services to their customers. These technologies also shift the focus of development teams away from traditional processes. It is no surprise that the shift to delivering services, where the user expects instant access and engagement, has also shifted the methodologies used to create these services, including the implementation of agile practices such as Scrum, Extreme Programming (XP), and Test-Driven Development (TDD). The use of these practices also introduces new processes not seen in the traditional approach of product development. For example, the daily operations of development teams have shifted to include more maintenance tasks (Uikey & Niten, 2015) that have been shown to add overhead to both source code development as well as test development (McIntosh, Adams, Nguyen, Kamei, & Hassan, 2011). The amount of system administration required to keep these always-on services up-and-running has also increased, as networking and load balancing become critical issues.

Nothing embodies the changes in application development and delivery like the concept of perpetual beta. The increase in agile processes, continuous integration techniques, and slipstreamed features all but demand productive dialog among development teams, quality assurance teams, data providers, and end-users, and require the use of best practices and user-focused policies in order to be effective (Shadbolt, O’Hara, Berners-Lee, Gibbins, Glaser, Hall, & Schraefel, 2012). Web applications are unique in that new code can be introduced into production with no effort on the users' part, and no downtime in the production environment. This allows development teams to introduce new features quickly, and also allows for quick feedback from users so that unpopular updates can be rolled back just as quickly as they are rolled out. Doing this successfully also requires a strict monitoring of the production environment in order to ensure that the services delivered are kept at a high quality. This increase
in velocity and decrease in traditional software release cycles also means that as Web applications move from hypermedia delivery to service and application delivery, business logic and data storage take on more important roles, as users access Web content new ways, including REST services and APIs (Keith & Schincariol, 2013).

In order to develop these quickly-delivered, consistently-modified Web applications, lightweight programming languages and models are required. JavaScript, for example, has become one of the foundational elements of modern Web 2.0 applications (Toledo & Tanter, 2011). Because of JavaScript's easy-to-implement features and small memory footprint, it is included as part of many frameworks, such as ASP.NET MVC. Accordingly, REST Services, XML, and APIs have become standard service-oriented architecture elements (Datta, Dutta, Liang, & VanderMeer, 2012; Li, Chou, Zhou, & Luo, 2016; Pan, Zhang, Wu, Li, Yang, Lin, & Shi, 2014). The ability these tools have given developers to separate the concerns of data from display properties (using the "loosely-coupled" model) allow for new Web browser functionalities not previously seen. Additionally, these services can be provided in scalable, cost-effective manners, taking advantage of other modern technologies (such as cloud computing), important for defining the business aspects of the system as well as the technical details for implementing these solutions. These lightweight, decoupled models have also allowed the proliferation and segmentation of the mobile marketplace, making Web sites and Web applications more accessible to users with varying smartphone and tablet screen sizes.

The proliferation and splintering seen in the mobile marketplace highlights a key component of modern Web applications, namely the fact that these Web applications are not limited to single-use devices. The use of Web applications has become largely device-independent, so that the end-user elements of the applications need to be available on many types
of end-devices (Chmielewski, 2014). This ability to unhinge Web applications from their standard desktop/laptop access models is mainly achieved by the separation of application layers, so that the user interface (UI) is processed separately from the data and business logic. Service-Oriented Architecture patterns embody this device symphony, where the application elements, data elements, and presentation elements can operate on separate layers (a la multi-tier architecture) and separate machines, combining multiple servers via load balancing with client-server architecture over networks, as a service synthesis compiled from different hardware elements (Immonen & Pakkala, 2014). The movement of services to cloud technology platforms only reiterates this, where the focus is on the services provided, not the Web applications themselves (Dikmans & Van Luttikhuizen, 2012), evolving the Web application into an "architecture of participation" (Nath, Dhar, & Basishtha, 2014), harnessing the power of networks, collaboration, and content, while exposing Web application functionality that results in a variety of rich user applications and experiences.

This advancement in device fragmentation, away from the traditional desktop and laptop presentations, and towards a more participative engagement on behalf of the users, has spurred the desire to deliver rich user experiences. On a basic level, the development of rich user experiences centers around designing and delivering services that are more intuitive and convenient than before. Technologies such as XHTML, AJAX, DOM, CSS, and HTML5 give developers the ability to deliver Web applications that are more useful, usable, desirable, scalable, and, ultimately, more valuable (Mohan, Agarwal, & Dutta, 2012). However, the use of this myriad of complex technologies also creates an environment that can be very hospitable to a host of vulnerabilities that must also be addressed (Ofuonye & Miller, 2013). This focus on user experience (UX) and user interface (UI) design incentivize the use of Web applications, making
the users want to use the application even more. These ideas must also be weighed against the need to develop Web applications that are sometimes required to manage complex states, provide easy accessibility to users, and ensure that proper authorization and authentication are used for accessing the elements of the application.

**Testing Challenges**

The above discussion highlights many of the difficulties encountered when testing modern, browser-based applications. Choudry, Prasad, and Orseo (2014), for example, have experimented with developing prototype tools, testing cross-platform Websites for compatibility with the World Wide Web Consortium's (W3C) One Web Principle, a doctrine that stipulates equal access to information and services regardless of browser and device. These types of testing tools have become essential for ensuring the continued quality of Web applications, as these applications become more distributed, and as applications are increasingly required to perform equally-well across a variety of browsers.

The Web, as a platform, seeks to deliver four essential types of information:

- Data, rendered as HTML, XML, or similar file
- Style, rendered as CSS and other technologies
- Client-side code, including JavaScript
- Binary files, such as images, documents, and executables files

All four of these information types must also work in interoperable environments (Zao and Xia, 2014), and must do so effectively in order to drive operational excellence and competitiveness by coordinating and reconfiguring digital connections, while also managing updates to these connections and maintaining services with little downtime.
The Internet's ability to harness collective intelligence by storing large amounts of data highlights additional complications in modern Web application testing. Reflective of Moore's Law, an observation made in 1965 that states the number of transistors on a computer chip doubles roughly every two years (Moore, 1998), ever-increasing volumes of data are being created, stored, and retrieved by Web applications. The shift of the Internet from a place to retrieve data, to a place to publish data, means that modern testing solutions must also keep pace. Deed and Edwards (2013) have noted that a primary issue in harnessing collective intelligence lies in the transition from simple communications to complex, open conversations where individuals can assess and interact with the data.

Testing the databases that serve as the repositories for the information used in building this collective intelligence involves several important steps. Kaur and Sehra (2015) identify key points for ensuring proper testing of databases, including:

- The number of arguments being passed
- The data type of arguments being passed
- The order of the arguments passed
- The return value
- The data type of the return value

Additionally, their research examined issues surrounding data integrity, namely testing for blank and default value retrieval and ensuring that all values can be entered, modified, saved, and deleted successfully. Recent research also shows that the validation of the conceptual models upon which modern databases are designed, often built using CASE tools, is an important, but often inadequate step in the software testing process (Aljumaily, Cuadra, & Martínez, 2012).
These new technologies, resulting in SaaS, PaaS, IaaS, and other online offerings, require new ways of testing. For example, Matthew and Spratz (2009) have demonstrated the use of an integrated, on-demand, automated testing infrastructure included with the Platform-as-a-Service Force.com. This PaaS is built upon a MVC framework, and includes Apex Code, a Java-based object-oriented language for developing the business logic for the platform, as well as developing unit and user interface (UI) tests.

One of the most important advances in software development, design, and testing is the introduction of Test Driven Development (TDD). Formally introduced by Beck in 2002, this agile practice places testing as the driving force behind code development for quick development of code that is also higher in quality and easier to test. Since then, much research has been conducted on the effectiveness of Test Driven Development practices, with mixed results. Recent research by Dogsa and Batic (2011) suggests that TDD can improve quality and increase maintainability, but did not find that productivity was increased, compared to traditional software development practices. Other research has shown similar results. Interestingly, TDD practices are also being assimilated into new development strategies, along with other testing and development methodologies, expanding the software development and testing boundaries. For example, 2012 research by Sadeghi and Mirian-Hosseinabadi combined TDD with model-based testing, and found improvements in software quality by expanding the use of TDD to all phases of software development and being able to include non-technical team members in the TDD process, as well as improving requirement traceability by connecting tests together in a hierarchy, and improving the testing process by being able to compose some of the tests automatically and independent of any particular platform.
The tools used to test Web applications that rely on lightweight programming models are being developed almost as quickly as the methods and tools for developing the applications themselves. For example, Karam and Safa (2013) developed a visual collaborative testing tool and methodology for testing state-based workflows of MVC-based applications built with Ruby on Rails. Research conducted by Jensen, Moller, and Su (2013) focuses on making automated testing easier for Web applications that use JavaScript, XML, and AJAX technologies by implementing server interface descriptions, following a design-by-contract model of software development, allowing for more flexibility in test data maintenance.

Modern testing means testing mobile platforms as well. To that end, much of the current research is focusing on how to appropriately test Web application use on mobile devices, with an understanding that users expect these applications to perform as well as their desktop or laptop counterparts, but also with the knowledge that the abundance of mobile devices, with their varied screen size, resolution, processing power, abilities, and connectivity makes this a difficult task. Ivan and Popa (2014) searched for answers by breaking mobile testing elements down into three elements: the proprietary server, the mobile client, and the third party services the be tested. Furthermore, they recognized that the situation is even more obfuscated by the fact that additional technologies, such as cloud computing, are increasing in popularity. Heitkötter, Hanschke, and Majchrzak (2103) have evaluated the development of stable, cross-platform Web applications, based on several criteria, including supported platforms, long-term feasibility, and application speed, which can also be applied to the testing of the application in question as well.

Testing is also made more complicated by the use of technologies designed to offer rich user experiences, such as AJAX and JSON. Dhote and Sarate compare the classical Web application model, where synchronous client requests and server responses are made, creating a
new Web page for each request for the client browser, to modern AJAX Web application models that are capable of asynchronous calls, posting partial page data to servers and loading only particular data elements from the page and not entirely new content, incrementally, on an as-needed basis (2013). They note that many of the traditional testing methods used may not necessarily apply to Web applications that implement AJAX, since the tester may not necessarily take any actions, yet there still may be many calls to the server for updated data. Additionally, the AJAX engine that exists in an AJAX application serves as something of an intermediate client-side server, sending data to the client without having to use the full capabilities of the server and reducing the load placed on the server.

Mesbah, van Deursen, and Roest (2012) also speak to the difficulty of testing modern AJAX-based applications, citing their event-based nature and the ability of the client to manipulate the browser Document Object Model (DOM). They note that static analysis, while useful for traditional Web applications, is unable to expose the dynamic dependencies used in modern, rich internet applications, and that classical page request/server response models of Web testing do not take client-side processing into account. Their research focuses on the use of fault models for probable AJAX application errors, such as violations of the DOM and error messages, the use of invariant types, such as XPath and JavaScript expressions, for expressing invariants in Web applications being tested, and the use of algorithms for achieving transition coverage of state-flow graphs for the application being tested.

**The Importance of Automated Testing**

Software quality assurance and testing are essential parts of risk management strategy, producing risk-related information that, when done correctly, determines whether or not the system under test (SUT) meets required results. Understanding that all applications have defects,
good quality assurance and testing practices result in software failures, displayed as external incorrect behaviors, that point to an internal software error that must be corrected in order for the SUT to meet required results. This is a process that can and should be performed throughout the life cycle of the software product, as a way to ensure that the needs, demands, desires, and preferences of the stakeholders of the project are satisfied (Black, 2011).

While software testing can and should be a hybrid of manual and automated testing, manual testing has its limits. Test cases can be complicated, so that manual input becomes time-consuming and complicated. End-to-end testing via the user interface can involve complex values, with multiple paths taken, multiple states that need to be maintained, and multiple calls to the server for data that must be validated. Automated testing can result in a decrease in execution time required, which can be of great benefit during regression testing, where all constructed functionalities are tested to ensure changes made do not negatively impact other parts of the system (Just & Schweiggert, 2011). This is also especially useful when using continuous integration (CI) techniques, where numerous code check-ins can occur on a daily basis, modifying and adding system functionality, in order to verify that no new defects were introduced between automated testing sessions. Automated testing is considered one of the main resources for improving software testing process efficiency (Swartout, 2012), but is also not to be used in place of manual testing (Mesbah, et al., 2012).

Donegan, Bandira, Matos, Luciana da Cunha, and Maia (2007) also noted that automated testing is important because it allows manual testers to focus on advanced issues, such as exploratory testing, knowing that the automation can assist with basic user interface tests. Test coverage can not only increase, but an automated system allows tests to be repeated, exactly the same, so a test that reveals the presence of a defect can be ran again after the defect is fixed, with
an understanding that the same inputs will be applied and evaluated, bringing a uniformity in testing that is difficult to achieve via manual testing.

The Use of Frameworks

A software framework has been described as a "well-defined support structure for organizing and developing a software project" (Arpaia, Inglese, & De Matteis, 2015, p. 33). This can include support structures, libraries for access to common controls and data storage and retrieval, and scripting languages for run-time execution of tests. Test automation frameworks should be designed as independently as possible from the details of the target application in order to facilitate flexible use, as well as to accommodate new elements to be tested. Additionally, test automation frameworks can be identified, separate from simple class libraries, due to their focus on code reuse and reduction, minimizing the amount of code required to achieve the goals of the testing solution. Frameworks designed for flexibility and modularity can provide the most benefit, as elements that can be rapidly reused reduce the costs and time required for test creation.

Research by Wand and Du (2012) demonstrated that the use of software frameworks can improve the extensibility and reusability of automated tests, in a productive and efficient manner. The use of frameworks in test automation solutions results in the separation of concerns, so that the various elements of the solution can all be modified and updated independently from one another: the data, the tests, the framework, the tools used to manipulate the browser, and the actual browsers themselves. This independence provides the flexibility and independence needed in Web application testing, so that the maximum amount of code reuse is achieved, which reduces overhead in the coding of the solution, allows for users of the solution to better understand the elements that make up the tests and how they interact with each other, and greatly
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK

reduces the amount of maintenance and re-work in order to keep the test automation system running well (Pluralsight, 2013).

The UML Testing Profile

The UML Testing Profile (UTP) represents an important step forward in the design, visualization, specification, analysis, construction, and documentation of software tests (Object Management Group, 2013). The UTP is specifically devised for test system design and configuration, including the "identification of the System Under Test (SUT) and its boundaries, the derivation of test components, and the identification of communication channels between the SUT and test components" (Object management Group, 2013, p. 1).

The use of UML and UTP concepts has been shown to help software designers manage complex designs, by being able to refine layers of abstraction into actual components of implementation (Wehrmeister, Ceron, & Silva, 2012). The use of modeling and CASE tools has also shown to aid in ensuring that the right system is built with fewer defects, quicker and with less expense, by detecting errors earlier in the development process, displaying the cost of change tendency, where changes made later in the development process take more time and money than changes made earlier in the process.

3. System Design and Methodology

3.1 Action Design Research

A primary goal of Action Design Research (ADR) is to combine aspects of theoretical contributory research with the desire to investigate issues faced by current practitioners in the industry (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011). ADR recognizes that, at its core, the discipline is based upon the development and use of artifacts in an organizational setting. ADR also recognizes that a shortcoming of pure design research, however, is the fact that current
design research methodologies do not typically view artifacts as emergent from an organizational context. ADR seeks to generate "prescriptive design knowledge through building and evaluating ensemble IT artifacts in an organizational setting" (Sein, Henfridsson, Purao, Rossi, & Lindgren, p. 40, 2011). Through the use of seven principles and four steps (three of which are iterative), ADR addresses the two separate challenges of undertaking a specific issue faced by a specific organization, by means of intervening and evaluating, and developing and evaluating an artifact to address an entire class of problems exemplified by the issue.

Figure 1: The ADR Process Stages

The first stage in the ADR process involves a reaction to a perceived, in-practice problem, that serves as the basis for the research effort. Through the review of prior research, study of existing technology and practices, and by input from current practitioners and researchers, issues of practical matters are found, inspired by, and wedded to theory (Hilpert, Beckers, Kolbe, & Schumann, 2013). The two primary guiding principles in this first stage are...
the importance of practice-inspired research, using field problems as knowledge-creation tools, and the generation of ensemble artifacts as informed by theories. It is important to structure problems in such a way that solutions can be identified, and so that the theories examined may manifest themselves into an initial form that can be used, evaluated, and reshaped by organizational practice.

Stage two involves the iterative building, intervention, and evaluation of the artifacts. These iterations can be placed somewhere on a spectrum of research design, with one end of that spectrum being IT-dominant, meaning the focus is on the creation of innovation from the beginning of the build process, so that alpha versions of the artifact are conditionally implemented by the organization, and a more developed beta version of the artifact sees use in a wider organizational context, so that the artifact can be continually scrutinized and developed by use in-context. The other end of this spectrum is an organizational-dominant process, that focuses on the organizational use of the artifact as the primary driver of design contributions. This process attempts to deploy the artifact as early as possible in the design process, so that the design knowledge is developed by the organizational participants who either, ultimately, decide to adopt or reject the artifact.

Stage two utilizes three principles: reciprocal shaping, mutually influential roles, and authentic and concurrent evaluation. Reciprocal shaping refers to the fact that, when performed correctly, it is impossible to separate the influence of the produced artifact on the organization and the influence of the organization on the artifact. The iterative process provides the opportunity for each element to impact the other, in order to not only produce a more viable artifact, but ultimately produce useful, prescriptive design knowledge. Mutually influential roles refers to the importance of creating a mutual learning environment between project participants.
The researchers are available to disseminate knowledge of theory and current technology, while organizational practitioners provide practical information. Authentic and concurrent evaluation involves ensuring that the evaluation of the artifact occurs at the same time as the building of the artifact. This differs from traditional stage-gate models where evaluation is conducted only after the artifact is built and implemented in the organization. The iterative process seeks refinement of the needs of the organization as well as the needs of the artifact.

Stage three is an ongoing process throughout the first two stages, reflection and learning. This stage ensures that as understanding of the problem domain increases, that the research process is adjusted accordingly, via conscious reflection on the domain and artifact. The primary guiding principle for this stage is guided emergence, where two seemingly disparate ideas, design (as intentional) and emergence (as organic) act upon each other, recognizing that the produced artifact is not only a product of initial design, based on theoretical constructs, but is also inspired and shaped by continual organizational use.

The final stage of the ADR process is the formalization of learning, where the specific issue and resolution investigated is cast to a wider class of field-level problems, realizing the first guiding principle of the ADR process (practice-inspired research). This stage relies on the principle of generalized outcomes in order to apply the accomplishments of the designed and tested artifact to other situations. Both the problem originally addressed and the solution to overcome the problem can be generalized, and design principles can be derived from these concepts in order to formalize results and make them applicable to other environments.

### 3.2 Test Framework Development

The UML Testing Profile (UTP) was developed with an understanding that testing is often not integrated with other design and development phases (Object Management Group,
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK

2013), and the means of developing a complete testing tool that can accurately gauge the "fitness" of a system to its intended, specified use, that meets the requirements for the purposes of testing software, including the ability to provide programmers relevant information for the prevention of defects, provide management the ability to determine the amount of risk involved when implementing a particular software package, achieve as "bug-free" of an environment as is warranted, and to validate (ensure the system meets customer requirements) and verify (ensure the system meets specifications) a software package (Beizer, 1995), is, at best, difficult. The UTP includes the ability to create, document, visualize, and specify a testing solution that is domain-independent, and provides a basis for specifying test architecture, data, and deployment strategies.

The development of a testing framework should be considered, by an organization developing such framework, a development process on par with the development process employed when developing the system under test. "Automation development requires the same discipline as software development" (Graham & Fewster, 2012, p. 533). The development and successful execution of automation utilizing a framework is as difficult, if not more so, for several reasons. Fewster and Graham (1999) speak to these concerns and mention at least three primary issues that warrant addressing, and can be resolved, with proper implementation of an automation framework.

The process of automation development generates a large number of individual artifacts that must be properly stored and accessed in order to be effective. So, test data, test scripts, shared information, output results, and all other data required must all be organized in a way that is scalable. Proper automation is capable of being used by multiple individuals on the project, and having a single person as the sole responsible party for development and effort is a recipe for
disaster when that individual moves on to another project or organization. Finding an architecture that is understandable, modular, and scalable, from the commencement of the framework development process, is key.

Real progress in automation is obtained by reuse, and good automation seeks to minimize the amount of code duplication and maximize reuse. "The ideal Automated Software Testing Framework is developed for reuse" (Dustin, Garrett, & Gauf, 2009, p. 148). Elimination of duplication results in increased implementation velocity when creating new tests and maintaining existing tests. Reuse partially depends upon the above concern, in that, even if a reusable element of the solution exists, it must also be able to be found by the practitioners when required, because if it not found, another version of the same element will be developed by the practitioner.

Maximum reuse depends upon a framework and design that is easy to understand and navigate. When reusable components are developed, future reuse must be designed into the element, which takes a knowledgeable developer. Additionally, reuse, in our context, means the ability to reference the original data in some way, not making copies that can then be mutated or edited and saved in some way. A reusable library of components is crucial to the self-sustenance of any automation solution.

Finally, the ability to build platform and environment independence into an automation solution will help ensure that modern, browser-based applications are tested across multiple browsers, versions, and operating systems for assurance of quality. It is understood that the expected outcomes may be different, based upon the environment and tools used, so that the data may display differently when looking at it utilizing Google's Chrome browser versus Microsoft's Internet Explorer, for example. The rules for what elements should be tested, and the amount of
acceptable variance between platforms and environments, should be part of any automation framework, reducing the amount of repeated code when possible.

In order to address these issues, the development of any automated solution should be based off the abstraction of layers (Graham & Fewster, 2012). Using concepts related to application design patterns such as Model-View-Controller (MVC) design patterns, the elements of the solution that manipulate the browser's user interface need to be separate from the elements that will actually test business functions and requirements. Doing this allows for a separation of concerns, so each element may be edited and implemented independently of the others. This also allows for easier maintenance of the overall solution, so that one element may be changed without affecting the other elements. Additionally, future changes to one part of the solution (for example, using a different toolkit to manipulate the browser) will not affect the other elements of the solution. Data-driven testing involves the separation of the data used in the testing from the code that executes the tests, which provides additional flexibility in test management.

### 3.3 Data Collection and Gathering Techniques

In order to form a more complete measurement of the effectiveness of the artifact, the data points for this research included a mixture of qualitative and quantitative elements. All iterations of the building, intervention, and evaluation (BIE) process included quantitative measures, including measurements of time taken to run the tests, compared to a baseline of manual testing using the same data and process. Total source lines of code (SLOC) were also measured for each BIE iteration.

Qualitative data was also elicited, in the form of open-ended, written questions asked of participants in the quality assurance process of the organizational SDLC, that included questions of ease-of-use of the framework, reusability of the artifact, and readability of the code for the
framework and the resulting tests. These questions were asked in the second iteration of the BIE process, with follow-up questions asked as warranted. Close attention was paid to emerging data that seemed to contradict initial thoughts, in order to learn how these data points could be applied to future processes.

### 3.4 How ADR and the Test Framework Development Principles Were Applied

![Activity diagram that demonstrates how ADR and the test framework principles were applied.](image)

*Figure 2:* Activity diagram that demonstrates how ADR and the test framework principles were applied.
The ADR process was applied to the principles of test framework development by starting with the Problem formulation stage (Stage 1), where the need for a reliable test framework, based on the principle of practice-inspired research, and the need to a generalizable and repeatable structures, based on the principle of theory-ingrained artifact, were discovered. The building, intervention, and evaluation (BIE) stage occurred in three cycles, with each cycle containing one building, one intervention, and one evaluation stage.

The initial BIE stage introduced UML testing profile concepts to framework creation, molded by researcher and practitioner viewpoints, and evaluated internally by the researcher, in order to define and generate an introductory framework. The second BIE stage sought to introduce additional design principles to test framework creation in order to generate domain-independent framework specifications and models. The artifacts improved and created in the second BIE stage were used in the final BIE stage to contribute to the testing of a specific system under test (SUT), evaluated for usability, and served as the basis for a discussion of the evolution of design principles via organizational use. The result is a compound artifact that serves as a method of test development framework creation suitable for an organizational context.
3.5 ADR Iteration by Iteration

3.5.1 Iteration I: Crafting the artifact

3.5.1.1 Problem formulation.

*Discovery of immediate or anticipated problem.* The initial impetus for this research project came about as a result of the implementation of an automated testing program for the information technology company for which the researcher works. The organization had, before this time, no prior experience with implementing automated testing for this particular project.

Before the commencement of this research, a test automation solution, including framework, had been initiated in related modules. However, throughout the continual use and development of this initial framework, it became clear that there was room for improvement. This initial framework was being used for writing black-box, user interface, end-to-end regression tests, that mimic users' steps taken to complete different scenarios.

This current automation solution system was not being implemented in all modules being developed for the organization’s clients. Additionally, no long-term planning was being introduced into the framework, and the framework was not being built using any strict methodology, which made it complicated for engineers writing tests to understand how to effectively use the framework.
Identification of a class of issues to which this specific issue belongs. This is a common issue that is growing in complexity as time progresses. As the researcher was able to speak to current automation practitioners, as well as read, listen to, and watch various media sources, and complete the literature review, he was able to receive insight as to the importance of building a test automation framework the right way from the beginning. Additionally, the struggle of being able to accurately model the existing system under test as a usable and helpful tool was found by the researcher, by attending user groups for both general quality assurance testing and tool-specific practices. In the flow of several conversations (and several "lean coffees"), it became clear that while many other automation engineers were able to focus on the tools being used and good programming design principles, there was not much talk revolving around how to build the right framework that serves to accurately model the system under test and provides not only for flexibility in testing but also the robustness required for efficient and repeated testing that can be trusted.

Additionally, during the initial exploratory stages of the research, it was discovered that the Object Management Group (OMG) has developed a Unified Modeling Language (UML) extension, the UML Testing Profile (UTP), as way to encompass testing design and specification. It was also discovered that many of the concepts contained in the UTP have not been fully researched and tested, as not many practitioners were aware of the UTP or its benefits.

For practitioners, this means that there is not any real source of inspiration for the "ideal" framework, off of which any "best practices" or guidelines can be gleaned. For the researcher, this relates to a class of problems revolving around the best way to build an initial automated testing framework for reliably representing the system under test.
**Initial research question formulation.** Based upon the initial and continued discussions with current practitioners, as well as a survey of the initial framework used for the organization's black-box user interface testing, a draft research question of how the UML Testing Profile can be used to improve test framework design, was drafted, with the organization's automated test framework as the test case.

This original research question guided efforts to complete a literature review in order to better understand the particular challenges facing those that test modern, Web-based applications, and what kinds of elements constitute a high-quality test framework. Having completed this, two research questions were formulated:

**Q1:** How can the UML Testing Profile be implemented in order to create a robust automated test framework that addresses the concerns of testing modern, Web-based applications?

**Q2:** How can this framework be implemented in-practice?

**Identification of theoretical bases that contribute to resolution and prior technology advances.** The literature review was conducted to review the concepts that contributed to test framework creation. Through this process, the picture of what needs to be included in a successful test framework became clear, as the benefit of using the UTP was shown. The inclusion of Web 2.0 concepts was noted, and the construction of a framework that is independent of the SUT, where code reuse and reduction principles are applied for maximum flexibility and modularity was shown.
3.5.1.2 Building, intervention, and evaluation.

**BIE process selection.** As the initial overview of the existing automation framework was being completed, the researcher saw that there was an opportunity to completely redesign the framework from scratch. After meeting with the quality assurance manager and discussing the research aims and goals, it was made known that the researcher could have full control over this redesign. Because of this, the decision was made to use an IT-dominant BIE process.

This decision was made for several reasons. First, the researcher is also the senior engineer for the project’s quality assurance team, so keeping the initial, alpha-version of the ensemble artifact internal to the research team also, in essence, involves current practitioners and end-users, as the ADR team and end-user team are completely separate. While there are certainly additional end-users for this artifact, the researcher is also, ultimately, an end-user of this artifact as well. This also denotes a change from the typical ADR process, where end-users are considered wholly-separate from the ADR team. Testing the alpha-version of the ensemble artifact with just the researcher team allowed baselines in performance and "fitness of use" to be set, before revising the ensemble artifact for more widespread testing and use.

**BIE cycle execution.** The researcher started with an evaluation of the UML Testing Profile, in order to understand the embedded design principles contained in the profile and how they might apply to test automation frameworks. The UTP, as an extension of UML, is similarly designed as a graphical language capable of supporting design and implementation details for object-oriented systems, and uses existing UML concepts when it is possible to convey those details accurately. The organization agreed that the use of such a tool would assist in the generation of an appropriate framework, and so the goal of the initial BIE process was to develop a framework that accurately reflects the SUT.
About the system under test. The System Under Test (SUT) is defined as an Electronic Marriage Registration (EMR) System. The purpose of this system is to allow for the creation, retrieval and updating of official, state-government records, primarily the marriage license required when two parties wish to be married, as well as the marriage certificate issued after the two parties have been officially wed. This SUT is currently being implemented by a client for the organization, and, whereas this particular module has received quality assurance testing prior to its release, all testing done before this research project was manual.

In order to effectively manage the project, and make the resulting ensemble artifact as thorough as possible, it was decided that three particular use cases would receive focus:

1. The ability to create a new marriage record
2. The ability to issue a marriage license
3. The ability to approve a marriage certificate filing

These use cases represent the core functionality of this particular module that differ from other modules (the Electronic Birth Registration System or Electronic Death Registration System, for example). The first use case tests whether the user, as an authorized, local registrar, is able to create a new marriage record, by entering the data on each of the marriage registration pages, and saving all of the data entered. This use case is the primary step in the process of completing a marriage registration and issuing a marriage license and subsequent certificate.

The second use case tests the system's ability to produce the correct output for a marriage license, once the proper data has been entered for the marriage record. As this use case produces an official document, this use case also includes the use of the Order Processing module, since the two parties to be wedded will purchase the marriage license from the local official's office once the new marriage record has been entered and verified.
The last use case for this research tests the system's ability to approve a marriage certificate filing once a proper marriage license has been issued and the information from the wedding ceremony is added to the record. This use case is performed by an authorized, local registrar, and makes the record available to the state registrar's office for final approval and registration.

Together, these three use cases represent the core functionality of the EMR System, and demonstrate the primary uses of the system. The use cases include the ability to create, retrieve, and update data in the database, core functions of any database application. It should also be noted that this system does not include the ability to delete cases from the system, due to various legal mandates on how vital records are treated by state agencies.

Building the framework. The use cases defined above represent system-level testing, identifying aspects of the testing to be performed that verify the SUT conforms to requirements. A primary goal of this system-level testing is the production and implementation of tests that exhaustively test the system, under normal use and under exceptional use, as specified by the requirements. The framework for these tests should be no less thorough. In order to reach this primary goal, two qualities of the framework that must be met are correctness and completeness. Correctness refers to the framework's ability to represent sufficient aspects of system behavior, while completeness refers to the lack of error, inconsistency, and ambiguity in the framework, with respect to the specifications provided. These notions of correctness and completeness are advocated for by the implementation of UML specifications, in order to promote precision of understanding and meaning.

The UTP defines, as the basis for all testing to be performed on a system, a test architecture, that defines all structural concepts required for the performance of tests (Figure 3).
This architectural package imports the system model package as well as the testing profile package where all UTP concepts are defined. For this research, The Electronic Marriage Registration (EMR) System Test architecture requires the use of the EMR System as the System Under Test as well as the Testing Profile, in order to gain access to the pre-defines UTP concepts and framework. In the figure above, the package demonstrates that the EMR System Test package relies on the TestingProfile package with its pre-defined UTP concepts and the EMR System model package as the SUT.

Use of the UTP also includes a context (Figure 4). The Test Context acts to define and group test cases together with stereotype <<Test Context>>, and lists all attributes for the test cases defined in the context, and defines all test cases as public class operations, where the type of value returned is the verdict (outcome) of the test.

The verdict is a predefined, enumerated data type, that specifies possible test results. This data type will be provided for the framework using NUnit. NUnit provides for four verdict types, Pass, Fail, Inconclusive, and Skipped. The Pass verdict records a test as successful, and is returned when a test method completes. The Fail verdict can be returned under four scenarios: failure of an assertion in a test (as a failing test), an unexpected exception (as a test error), an
attempt to run an invalid test (as not runnable), and upon cancellation of a test by a user (as a canceled test). Inconclusive tests represent a test where the results could not be verified as passing or failing. For example, a newly-created, un-ran test will have the verdict of Inconclusive since no testing has been performed. The Skipped verdict is returned on tests either explicitly marked as tests to be ignored.

For this research, the Test Context includes the attributes of the marriage record to be entered, as a new case in the system, as well as the marriage license service to be ordered and printed. The three test cases included, Create_Case, Issue_License, and Approve_Certificate, are also included, where each test case includes its own verdict.

A primary method of application of the UTP for system-level testing largely involves the use of existing UML specifications that represent the requirements of the overall system. The use of existing UML specifications for the creation of an automated testing framework gives the framework several advantages, including:

- easier communication, due to the use of graphical specification techniques, improving communication and agreement on framework construction
● a more complete framework, due to the additional thought and effort put into the initial visualization, construction, and setup of the framework

● earlier use of the framework for testing, based on the fact that the rigorous requirements implemented by UML can allow for earlier introduction of testing of the system, so that vague elements and errors in the framework can be discovered and cleaned up sooner

● reuse of requirements, between the creation of the requirements, the creation of the framework, the creation of tests that use the framework, and the creation of the SUT, achieving a higher level of consistency among all of the elements

_UML elements used in building the framework._ Use Case Diagrams. Use case diagrams are important specifications that can be used for describing the functional capabilities the system should include, and how the system's capabilities apply in relation to external stakeholders. Use case diagrams provide important information for test case coverage, and include data points for expectations for user capabilities. For example, Figure 5 below describes the use cases included in this research, and demonstrates how the actor, the Internal: Marriage Registrar, interacts with the system. Use case diagrams set the scope of the test context, and provide crucial information in regards to the SUT, as the use case subject, and users involved in the tests. Use case diagrams also represent an obvious arboreal branching structure that can be implemented in the test context, where the various use case nodes, extensions, and inclusion relationships can provide details for test structure and control flow that needs to be present in order to accurately test the system.
Additionally, use case descriptions, although not officially part of UML, can also provide
great detail in regards to how tests should be constructed, and point to key elements that should
be included in the framework. These textual descriptions serve to emphasize essential behavior
patterns associated with each use case. These textual descriptions also highlight the actor(s)
involved, any preconditions that must be present before the use case is executed, triggers that
cause the use case to begin, and end conditions that signify success and failure, that can assist

Figure 5: Use case diagram for the Electronic Marriage Registration System.
with determining a valid verdict of tests performed based upon use cases. The Test Context for our system-level testing described above was obtained by use case analysis of our EMR System, and the test cases discovered are contained in the text context EMRSystemTestContext. This context serves as the base of our test structure, off of which additional tests can be gleaned and produced.

**UML elements used in building the framework.** Interaction Diagrams. Interaction diagrams such as sequence diagrams and communication diagrams are useful for development of a test automation framework. Gaining an understanding of the test cases that need to be present by studying the use case diagram, it is important to understand how the elements of these tests will interact with each other, what messages are sent between elements, and what behavior is expected from the system. Sequence diagrams demonstrate the objects created by the system, as instantiations of a class, attributes that describe the information included for each object created, and the operations that each object can perform. Sequence diagrams can be particularly helpful since they, as dynamic models, display the specific sequence of messages that are passed between objects for a particular scenario. In order to realize framework objects from sequence diagrams, the interactions between the elements of the sequence diagram must be explored, where framework components can be extracted from the diagrams.

For example, in the sequence diagram for the first use case below (Figure 6), there are messages being sent from the user (the Internal: Marriage Registrar) to the system, via the boundary objects (Start New Case and Marriage License Pages roll-up object). Additionally, the boundary objects communicate with the Marriage Case Controller and the database, the Marriage Case Controller sends messages to the Marriage Case object itself, which, in turn, send messages to the validator database. What this means is, for our test framework to be successful, each of
these objects must be faithfully represented in our framework. Each of the messages sent by the objects in the diagrams must be accounted for, all elements that accept data need to be included, and all data sent in the messages should be represented.

Figure 6: Sequence diagram for Use Case 1.

UML elements used in building the framework. Activity Diagrams. Activity diagrams depict basic activities and the relationships among the activities in a process. The activities depicted in an activity diagram are also associated with use cases, and can be divided in the activity diagram into separate swim lanes for activities performed by actors and activities performed by the system. As these activities typically transform objects in the system, activity diagrams can be used to glean information regarding framework behavior. Additionally, control flows in activity diagrams are used to model the paths of execution through business processes,
connecting activities to one another as the activities modify the objects in the system. The control nodes in an activity diagram also provide information regarding the various paths of execution that must be accounted for in a test automation framework.
Definitions for how actors stimulate the SUT, the modeling of flow control and manipulation of objects in the system, and judgments on observations made during the tests will
need to be included in the framework. The primary way this is done, using the Selenium toolkit, is via the Page Object Model. Use of page objects results in a separation of concerns between framework objects that contain object-specific code and the tests that implement those framework objects. This results in reduced test maintenance and code duplication, two primary concerns in creating modern test automation frameworks. A page object is "an object-oriented class that serves as an interface to a page of your [SUT]" (Selenium Project, 2016, para. 1). Typically, each Web page is given its own page class, where each class locates the elements of the Web page to be interacted with, and also contains methods for interaction with the page.

Appendix B contains the page object model class files built for this iteration. The genesis of these pages is a base class, called BasePage. This base class contains all functionality that can be found on all (or at least most) pages, including:

- the ability to visit a URL (Visit method)
- the ability to find elements on the page with which to interact (Find method)
- the ability to type text into an element (Type method)
- the ability to type text into an element only if it is enabled (EnabledType method)
- the ability to type text into an element only if the element is displayed on the page (DisplayedType method)
- the ability to replace typed text with other text (Replace method)
- the ability to select an option from a dropdown (Select method)
- the ability to select an option from a combo box (SelectComboBox method)
- the ability to select a checkbox (Check method)
- The ability to select all checkboxes on a page (SelectAllCheckboxes method)
- the ability to determine whether an element is displayed or not (the overloaded IsDisplayed method with one parameter)
- the ability to determine whether an element is displayed or not, within a given time frame (the overloaded IsDisplayed method with two parameters)
- the ability to determine whether an element is enabled or not (the IsEnabled method)
- the ability to switch iframes (the SwitchToFrame method)
- The ability to switch to the default frame (the SwitchToDefaultFrame method)
- The ability to return the page's title (the GetTitle method)
- the ability to return the text of an element (the GetText method)
- The ability to return the Case Id (the single unique database key for a record) for a case (the CaseId method)
- The ability to return the status of a case (the CaseStatus method)

Additionally, the ability to select common buttons found on case registration pages have been identified, and for each button, a separate method for selecting that button has been created (such a search button, save button, next button, and validate case button).

Common elements on all (or at least most) pages has also been identified and added to the base page class as locators. These common elements include a system wait message, that is displayed when the system is attempting to send data to the database and return additional information, such as a new page to display, or validation results for a case. Elements such as the Case Id element, case status text, and common buttons are all included in the base page in order to draw, as much as possible, from a single code source. In this way, we achieve a primary goal of a test automation framework: reduction in maintenance costs through the elimination of duplicate code.
The base page object also contains a variable for the IWebDriver. The Selenium WebDriver (or, as is the C# convention for interfaces, IWebDriver) is an object-oriented API that allows a coded test to manipulate a browser, directly and natively, just as a user would when using a keyboard, mouse, or other input device. WebDriver, as an interface, contains only signatures for the various methods and properties contained therein. As an interface, the implementation of the interface is left up to the class instance created by each test, where the driver for the specific browser is assigned to the WebDriver for implementation. So, when testing using the Chrome browser, for example, a new ChromeDriver class is created and assigned to the WebDriver variable. The implementation of the methods and properties specific to the Chrome browser are defined there, so that each major browser type may have its own specific implementation, all while using the same WebDriver interface.

Each of the page objects extends the base page class in order to inherit all of the methods defined in the base page class. These page objects gain access to the base page's methods and locators, and include additional locators for the elements found on that specific page. For each page included in the tests, methods are used to interact with the system in ways as indicated by the sequence diagrams and activity diagrams. The methods created for each page object represents elements of the sequence diagrams and activity diagrams.

*Intervention of the artifact in the organization.* The framework was used to create tests for the three stated use cases, and these three tests were run, once a day, for five days, in order to determine how well the framework met the stated goals of accurately reflecting the SUT, correctly and completely, with minimal duplication of code and maintenance. The tests were run using the Chrome browser each time, with the same test data (incrementing that dates each day)
for each set of tests that were run, in order to control for independent variables and external factors as much as possible. For each test run, the time taken to run the test was noted.

Additionally, this testing was performed using data-driven methodologies. Data-driven testing includes the use of a table of inputs to the SUT, and can include a table of expected results as well. This means that there is a separation of concerns between the data used for the tests and the tests themselves. The separation of variable data from the invariant tests provides the ability to edit the data without having to edit an entire test. This separation of concerns makes the testing more variable and easier to maintain.

For this framework, the data is stored in an Excel spreadsheet. Each data element to be used in the test is designated with a unique column header, and all data elements for a single test are located on a single row of the spreadsheet. When a test is initialized, the data is read from the spreadsheet to a data table for use in the test, and when an element of the test required data, the arguments that are passed indicates the row and column where the data is located in the data table, which is passed to the object's method.

 Evaluation of artifact. For this BIE cycle, the total number of lines of code, between the framework and the tests, amounted to 855 total SLOC. The framework accounted for 678 of those, and the test components accounted for 177 lines of code. Additionally, the Visual Studio code metrics rated the code's maintainability index at 70, on a 0 to 100 scale, with a score of less than 10 being poor, 10 to 19 being fair, and above 19 being good. The framework individually scored as a 71 and the test components were scored at 55. The time per test, as well as average time for each of the tests, is indicated in Table 1, in order to establish a baseline to use for comparison.
<table>
<thead>
<tr>
<th>Test Date</th>
<th>Use Case 1</th>
<th>Use Case 4</th>
<th>Use Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/04/2016</td>
<td>1:23.3</td>
<td>1:20.7</td>
<td>0:40.3</td>
</tr>
<tr>
<td>10/05/2016</td>
<td>1:15.9</td>
<td>1:15.8</td>
<td>0:41.1</td>
</tr>
<tr>
<td>10/06/2016</td>
<td>1:15.9</td>
<td>1:15.8</td>
<td>0:38.2</td>
</tr>
<tr>
<td>10/07/2016</td>
<td>1:25.1</td>
<td>1:30.2</td>
<td>0:37.3</td>
</tr>
<tr>
<td>10/10/2016</td>
<td>1:18.2</td>
<td>1:30.2</td>
<td>0:40.0</td>
</tr>
<tr>
<td><strong>Average Time</strong></td>
<td><strong>1:19.7</strong></td>
<td><strong>1:22.5</strong></td>
<td><strong>0:39.2</strong></td>
</tr>
<tr>
<td>Manual Testing</td>
<td>06:11</td>
<td>01:14</td>
<td>02:17</td>
</tr>
</tbody>
</table>

*Table 1:* Time, per test, per day tested, average time taken per test, and the time spent manually performing each test.

### 3.5.1.3 Reflection and learning.

*Reflection of design.* The initial phase of this research was dedicated to generating an initial framework with which tests could interact with the UI of out SUT. The concept of creating an entire automated testing solution seeks to include a separation of test elements, between the data used in the tests, the tests themselves, the framework that the tests use to interact with the UI, the Selenium toolkit that performs the actual browser manipulation, and the browser itself.

This initial design and building process took place with a previous understanding of the SUT, as the researcher has been a quality assurance engineer for this project team and application before this research began. Also, there is an existing automated testing framework in place, for other modules in this application, for the researcher to compare and study as well. This BIE cycle was a first attempt at using UML and UTP concepts to design a framework for this particular module, as this module has been, until this BIE cycle, untested, except for some manual testing efforts put forth by the QA team.

Upon reflection of this initial BIE cycle, it appears as if this initial building of the framework was a longer process than first imagined. As this framework design is a new design,
with no previous framework elements to use as a guide, all of the elements included in the page objects themselves were newly-created. This is a positive result of this cycle, as no existing framework elements were available to inform the researcher as how to proceed with the current research, but that also made the process of creating the framework using UML and UTP design principles more difficult to implement and slower.

An important lesson learned from this BIE iteration is to not only let UML and UTP principles be a primary guide, but also ensure implementation of solid programming and software design principles. As an example, the locators in the page objects for each page can be made protected and read only. The use of the protected keyword limits access to the class where the locator is contained and subclasses of that class, and making each locator read only helps to make the locator immutable, further ensuring that each locator is safe from unwarranted changes to the locators during testing.

Adherence to principles. Principle 1: Practice-Inspired Research. This composite artifact was constructed with an understanding that practitioners need a reliable, sturdy framework for their automated testing solutions. This organization, during the time of this research, was in the middle of establishing an automated testing program as part of their quality assurance testing, so this research was timely and necessary.

Principle 2: Theory-Ingrained Artifact. Understanding that the automated testing framework needed to incorporate Web 2.0 principles was an important first step in this process, and the literature review helped to make these principles known. By examining the concepts behind Web 2.0, the researcher was able to understand the complexity in the Web application that needed to be represented by the framework, and that UML and the UTP would assist in ensuring that these principles were faithfully and accurately represented.
Principle 3: Reciprocal Shaping. A lot of the time spent in this initial BIE cycle was spent attempting to articulate how the SUT is used in practice, in order to accurately reflect that use in the framework. The organizational goals of attempting to automate the use cases and major functionality of the system helped guide this initial BIE cycle, so that a genuinely useful artifact would be created for use by the researcher.

Principle 4: Mutually-Influential Roles. This initial BIE cycle was IT-focused, so this alpha version of the ensemble artifact was kept inside the research project. However, since the researcher is also the senior engineer for the project team, the researcher was able to discover new areas of design theory and technology, as well as capture important information regarding practical elements of the design and implementation of an automated testing framework.

Principle 5: Authentic and Concurrent Evaluation. The evaluation of the framework occurred as the parts of the framework were being built. An example of how this took place was the employment of the YAGNI principle during framework construction. The YAGNI principle, taken from Ron Jeffries' views on Extreme Programming (as one of the co-founders of XP), is based upon the acronym that stands for "You Aren't Going To Need It", meaning, do not implement features until they are necessary to complete the functionality required from the system. For the researcher, this involved writing each use case test to be implemented, and, as an additional piece of the test was written, also adding the necessary framework components with the UML diagrams and UTP elements as guideposts. This ensured that, as the artifact was being built, the focus remained on the initial research questions, and that the principles behind those research questions were being investigated by the researcher.

Principle 6: Guided Emergence. The preliminary design for the framework in this initial BIE cycle centered around the use of UTP architecture, the test context, the use of UML
diagrams, including activity diagrams and sequence diagrams, as well as focusing on the meta-design principles of use of abstraction layers (separating the data from the tests as found in data-driven testing, tests from the framework, the framework from the toolkit, and the toolkit from the browser) and use of page objects to represent the way the application is displayed to the user.

The framework also received benefits from the ongoing, authentic, concurrent evaluation, while the framework was being constructed. For example, the researcher found that it was necessary to include, for each page object, a locator that was unique for each page in the application, and to also include a method, in the base page object, by which a test can verify that the unique locator is displayed before proceeding with the test. If the locator is not displayed on the page after ten seconds, the test verdict returns as fail. This had to be included in the design of the framework during the creation, and was not included in any initial design elements, due to the fact that the concurrent evaluation of the artifact showed that the test would simply attempt to continue with the next test element, which, often-times, led to false verdicts of fail, due to the load time of the next page to be used in the test. Other, additional and substantial changes to the design of the base page occurred as the artifact was being formed and evaluated as well, including the inclusion of several more methods for interacting with elements on each of the pages. Not only were methods for typing and clicking built into the base page, but the ability to type data into fields only if they were enabled, select all checkboxes on a page, and the ability to select a choice from a JavaScript dropdown element, filtered based upon the value entered in the dropdown element, were required for consistent, positive outcomes of tests. The researcher had to be sensitive to the signals present while creating the tests and the framework, most importantly when it concerned the recording of false verdicts for a given test. Every attempt was made to ensure, in the framework, that when a test returned a verdict of fail, it was due to the SUT itself,
and not due to a defect in the ensemble artifact. In this way, the two separate ideas of intentional
design and organic emergence were able to act upon each other, producing an artifact based upon
an initial design informed by theory, but also shaped by continual use by the researcher in the
organization.

Principle 7: Generalized Outcomes. The problem instance in this initial BIE cycle can be
generalized thusly: It is difficult to create a test automation framework that can faithfully
represent the application to be tested, that also ensures the correct elements are being tested, and
that tests built upon that framework will run reliably. A generalized solution that comes out of
this instance includes the use of the UML Testing Profile in order to define the objects that
should constitute the framework. By using the UTP, a practitioner is able to define the overall
architecture and context for their testing environment. Additionally, by the use of UML
diagrams, a practitioner is able to understand the requirements of the application to be tested,
assisting in the creation of the framework elements that are most helpful to testing the SUT. The
design principles successfully applied here include the creation of page objects, that represent the
various pages of the SUT, based upon UML diagrams. By using UML diagrams, a practitioner is
able to model the framework in the same way the application developer models the application to
be tested. This includes the use of a base page object, from which all other page objects are
extended, in order to reduce the amount of duplicate code and make maintenance on the shared
methods and fields found in the base page object.

3.5.2 Iteration II: Crafting the artifact

3.5.2.1 Problem formulation.

The second BIE iteration consisted of identifying elements in the first iteration that
resulted in moving the project closer to its stated goal, and building upon those elements. The
reflection and learning phase of the last BIE cycle showed that much of the work in the first
cycle was relevant, and most of the work for the second BIE cycle would consist of improvements and enhancements to the current framework. 

The first BIE iteration consisted mainly of ensuring that the framework was built in accordance with the primary elements of the UML and UTP specifications. This included page objects for all pages used in the use cases tested, proper locators for each page object, and ensuring that each page object had appropriate methods with which to interact, as defined by the UML diagrams. These objects, as the ensemble artifacts for this cycle, represent the bulk of the work performed in the first BIE cycle. These page objects, when used in tests, comprise the basic building blocks of the use cases.

The use of page objects certainly assists with reaching the research goals of creating a robust automated testing framework that addresses the concerns of modern, Web-based applications, and comprises the elements of the UML diagrams for this project. However, some additional framework elements may be employed that can add not only to the robustness but also the aptness of reflecting the important elements of the SUT.

It was determined, through the reflection and learning phase of the previous BIE iteration, that the concepts of workflows that exist in the UML diagrams can also be modeled as part of the framework. Workflows can exist in the UML diagrams as a collection of activities in activity diagrams and sequence diagrams. For example, in the figure below (Figure 8), the activity diagram for Use Case 4 includes the computational workflow elements of searching for an existing case, and the workflow for entering issuance and payment data for the marriage license. Other examples of workflows that can be found in UML diagrams include the workflow for entering data for a case (as seen in Figure 9) and the affirmation of the marriage license (Figure 10).
**Figure 8:** Activity Diagram for Use Case 4.
Figure 9: Activity Diagram for Use Case 1.
Figure 10: Activity Diagram for Use Case 5.
3.5.2.2 **Building, intervention, and evaluation.**

_UML workflow identification._ The initiation of this BIE cycle included the identification of possible workflows that would not only represent the elements of the UML diagrams in a faithful way, keeping the fidelity of the diagrams as accurate models of the system translated for use as an automated framework, but also ease practitioners' use of the framework and resulting tests. In order to accomplish this task in the BIE cycle, additional evaluation of the UML diagrams was required.

The evaluation of the UML diagrams started with an evaluation of the activity diagrams. UML activity diagrams can be very helpful in representing the business process models used by an organization. It is important to remember that, despite the fact that the organization is using an object-oriented system, these business processes exist independent of the objects used by the SUT, and can cover a wide range of objects used in an object-oriented system. While it is also important to realize that activity diagrams can, at times, reinforce functional decomposition, which may impede testing efforts by losing focus on the overall testing strategy and goals of the testing efforts, with the right strategy, activity diagrams can provide insight as to how to make sense of business requirements outside the context of an information system or application, which can be beneficial for ensuring fitness of use in relation to the defined purpose of the system. For example, activity diagrams are capable of relating parallel activities and complex decision process that may need to also be modeled in the automated testing framework.

For our research, the three use case activity diagrams were studied in order to identify areas where workflows were possibly indicated. These areas tended to focus on the actors’ activities, and to not include areas where decision processes were indicated on behalf of the system. For example, the activity diagram for Use Case 1, Create Case, includes areas where the user selects the option to create a new case, entering new case information and then has the
system verify all data and search for potential duplicates, as well as an activity where the user enters marriage license information for the case into the system, then validates the data entered.

Use Cases 4 and 5 both include the activity where the user searches for an existing case in the system, that is also a separate use case (not included in this research), that involve no decision points. Use Case 4 also includes license approval activities, and the issuance of a print product, while Use Case 5 includes the activity where the user enters and validates marriage data.

Sequence diagrams, as a type of interaction diagram, are also helpful, since they describe the interaction between objects. Depicting the flow of control over time, for a single use case, sequence diagrams show the messages that are sent between objects in the business workflow processes and how those objects interact with each other.

The sequence diagrams for the use cases included in this research include some very important information that can be used to further identify and define what can and cannot be included in possible workflows as part of a test automation framework. The sequence diagram for Use Case 1, Create Case, shows areas where the primary actor interacts with the system, sending messages to different elements of the system. These messages can be grouped together, especially when they are sent to the same object, for example, where the primary actor creates a new case by sending the Create Case message entering the required identification information, and then selecting the option to create a new case. Similarly, the sequence diagram indicates a single message for entering license data, sent to the roll-up boundary object of marriage license pages, that may be grouped together into a single workflow.

Sequence diagrams for use cases 4 and 5 show the user sending messages to system objects for affirming the marriage license, affirming the marriage certificate, issuing and printing
the marriage license, and entering marriage license information. These messages are also candidates for being modeled as workflows in the framework, as they can provide consistent methods of handling the data for the test.

As this testing, and ensemble artifact, as a test automation framework, are to be used for black-box, user interface testing, it is important to understand the internal messages that are being sent inside the system, but not essential for the framework to model. It is, however, important for the framework to be flexible enough to be able to handle all types of return messages the system may send to the primary actor, and care must be taken in the framework to account for the various messages that should be accounted for by the ensemble artifact.

From a business process perspective, the use of workflows can highlight several important points. By introducing workflows into an automated testing framework, these workflows can emphasize many of the core business processes used by the application. This helps the framework and underlying code become better-aligned to the processes used by the end-users of the system, and makes the code used to test the system match closer to how a user would expect the system to operate. Limitations in the workflows can be discovered sooner, and measurements on workflow effectiveness can be made. The organization using the test framework benefits by ensuring that testing performed on the system meets expectations, and more effective decisions about the project should be possible.

Assembly of workflows in framework. The use of workflows in a test automation framework also bring programmatic benefits as well. Workflows contribute to the ideas of the "has-a" relationships among system elements, uniting the objects in a workflow's method into a single framework object. This works toward achieving a primary goal of object-oriented programming, namely code reuse and polymorphic behavior, that also keeps the flexibility of
individual objects. In order to accomplish this goal, each workflow is created as a class of workflow, including, in this BIE cycle:

- LoginWorkflow
- MarriageWorkflow
- OrderProcessingWorkflow
- RegistrationWorkflow

By defining the methods for each class compositionally, rather than by use of inheritance, the desired behavior of the object is achieved, along with code reuse, while also allowing the individual page objects that constitute a workflow to be used independently of a workflow, when required by a test. For example, being able to create a LoginWorkflow, that encapsulates all of the page objects required in order to get a user logged in to the system, allows that workflow class to include methods appropriate for an initial login to the system, subsequent logins to the system (that may require different page objects compared to an initial login), and also allow tests to be performed outside of an established business process, such as testing for incorrect logins and similar ideas outside the scope of this research.

Therefore, inside of each login, methods were created based on the workflows discovered during a review of the UML diagrams during this BIE cycle. The LoginWorkflow included a single method that allows the user to login initially into the system, The MarriageWorkflow includes several methods, one for starting a new marriage case, one for searching for an existing marriage case, one for entering license data, and one for entering marriage data. The OrderProcessingWorkflow includes a method for processing an order and issuing a marriage certificate, and the RegistrationWorkflow includes methods for affirming the license as well as approving the marriage information. Combined, these workflow methods encompass almost the
entirety of our tests, combining page objects from different parts of the system into coherent units that make sense.

In order for these workflow objects to work as self-contained units, they were built using compositional methods, creating new instances of the page objects it needed for each workflow, then using the particular method called to manipulate the objects required to perform the workflow. In the previous BIE cycle, the test itself created new page objects as part of its initial setup, which each test then manipulated directly. Included in each method are also the necessary assertions, again, previously part of the test itself, that checks to ensure the appropriate page loads correctly. In this way, these workflow objects act as hybrids between framework page objects and tests. Traditional test strategy includes the idea that all testing should be performed by the test itself, and that no assertions or verifications should be made by any elements in the framework. This convention was put into place in order to separate concerns between the tests and the framework, in order to add flexibility to how a page object can function, with an understanding that a page object may not always return the same result for each test where it is implemented. While this implementation of workflows as part of a test automation framework violates this principle, it does not diminish the use of the singular page objects when the need arises.

*Implementation of workflows in the tests.* In order to implement these workflows in the tests, the following steps were taken:

1. For each page object that is found in a workflow, but is not found individually in a test, that page object instantiation was removed from the test setup method.

2. For each workflow used in the test, a new instance of that workflow was added to the setup method for the tests.
3. Inside each test, the elements of each workflow, that were moved to an individual workflow method, were removed.

4. Inside each test, the workflow method that uses the required page objects was added, to be called when needed, that also pass the required data elements for each page object inside that method as arguments to that called method.

Through this process of removing the elements contained in the workflows form the tests and replacing them with workflow objects, our testing gains several advantages. One of the more obvious advantages is the fact that the tests become much easier to read. By removing multiple calls to multiple page objects, and replacing them with a single call to a single workflow, each test becomes more manageable, readable, and easier for a non-programmer to review and understand. Additionally, our tests are able to reflect our documentation better. There becomes more connectivity between the tests and the use cases, activity diagrams, and activity diagrams.

*Intervention of artifact in the organization.* Similar to the first BIE cycle, the framework was used to create tests for the three stated use cases, and these tests were run, once a day, for five days, taking note of how well the test implements the framework in accordance with the stated goals of this research project. Again, the tests were run using the Chrome browser, with the same test data for each test, which was the same test data used in the initial BIE cycle. Each test was run had its time recorded in order to compare average run times with other BIE cycles as well as the time taken to manually execute each test case. Data-driven methodologies were again used, where the data for each test was included in a spreadsheet that was read from by each test, and the data for each element in the test was passed as an argument to the framework method, which then passed the data on to the page object that required that particular piece of data.
Evaluation of artifact. The total lines of source code for the entire project, after this BIE cycle, is 867. 794 of those lines were from the framework, and 73 of those were used in the tests. Compared to the initial BIE cycle, the entire project grew by only 12 lines of code. The amount of code used by the tests decreased by 58.8%, down from an initial count of 177. Additionally, the Visual Studio Code Metrics places the overall maintainability index for the entire project at 70, which is the same as the previous maintainability index. Perhaps one of the more interesting numbers is that the maintainability index of the tests increased, from 55 in BIE cycle one, to 63 in BIE cycle two, indicating that the tests have increased in their ability be maintained due to the incorporation of workflows into the framework.

Times for each test run, as well as the average times, are indicated in Table 2. The results indicate that the introduction of workflows into the framework, and subsequent use of workflows for testing, compared to using page objects, as was the case in the first BIE cycle, does not dramatically increase the total time any test takes to complete.

<table>
<thead>
<tr>
<th>Test Date</th>
<th>Use Case 1</th>
<th>Use Case 4</th>
<th>Use Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIE 1 Average</td>
<td>1:19.7</td>
<td>1:22.5</td>
<td>0:39.2</td>
</tr>
<tr>
<td>10/18/2016</td>
<td>1:28.8</td>
<td>1:22.5</td>
<td>0:39.2</td>
</tr>
<tr>
<td>10/19/2016</td>
<td>1:28.8</td>
<td>1:15.5</td>
<td>0:37.3</td>
</tr>
<tr>
<td>10/20/2016</td>
<td>1:29.1</td>
<td>1:15.5</td>
<td>0:38.7</td>
</tr>
<tr>
<td>10/21/2016</td>
<td>1:23.8</td>
<td>1:21.9</td>
<td>0:36.7</td>
</tr>
<tr>
<td>10/24/2016</td>
<td>1:37.2</td>
<td>1:23.6</td>
<td>0:38.0</td>
</tr>
<tr>
<td><strong>BIE 2 Average</strong></td>
<td><strong>1:27.6</strong></td>
<td><strong>1:18.0</strong></td>
<td><strong>0:37.0</strong></td>
</tr>
<tr>
<td><strong>Manual Testing</strong></td>
<td>06:11</td>
<td>01:14</td>
<td>02:17</td>
</tr>
</tbody>
</table>

*Table 2:* Time, per test, per day tested, average time taken per test, and the time spent manually performing each test, for BIE cycle 2.
The evaluation of this iteration also included qualitative data solicited from the organization in the form of open-ended questions asked by the researcher to another quality assurance engineer that works on the same project. The feedback included positive reviews of the workflow concepts, naming conventions, and inheritance traceability. The organizational review also indicated that whereas, the naming conventions do assist with readability and understanding, some additional comments for classes and methods would assist in the understanding of these classes and methods.

3.5.2.3 Reflection and reengineering.

Reflection of design. In this BIE cycle, the focus was placed on reengineering the framework to take advantage of the concept of workflows. The hope was that by incorporating workflows into the framework of the automated test solution, that these workflows would help achieve our goals of test automation by helping to answer our two primary research questions: one regarding the use of the UML testing profile for creating a test automation framework that successfully tests modern, Web-based applications, and one regarding how this framework can be implemented for use in-practice.

It was determined that this focus would be helpful to pursue by reviewing the initial UML design documents, including use case descriptions, activity diagrams, and sequence diagrams. The previous BIE cycle focused on an initial creation of a framework, including a primary focus on an accurate portrayal of the elements of the application, as well as the primary ways that a user would interact with these pages. These application elements were developed as page objects, which include methods for interacting with each object, as well as definitions for each element to be interacted with on the page.
This latest cycle built upon the initial BIE iteration by extending the use of these page objects by implementing them inside of workflows. The UML activity diagrams, for example, demonstrate activities the user, as a primary actor, would take part in when interacting with the system. Additionally, sequence diagrams specify the types of messages sent between the actors of a system and the system itself. These types of specifications can help a team building an automated test solution by showing how the system is used, across a range of objects, and what kinds of inputs to and outputs from the system should be expected.

Considerable programming and redesign were required on the researcher’s part in order to develop the new framework elements. While the page objects were already in existence from the previous BIE cycle, how they were implemented and from where they were called changed in this second BIE cycle. Previously, the test themselves instantiated and called each page object, passing parameters directly. The use of workflows changes the test structure, so that tests instantiate the necessary workflows, call them when needed, and pass parameters to each workflow. The workflows, in turn, create new page objects, and pass along the required parameters to each page method called by that particular workflow.

Adherence to principles. Principle 1: Practice-Inspired Research. By being able to model the workflows from the UML diagrams in the automated testing framework, the technical domain was able to benefit the organizational domain's work by helping to meet its end-goals of effectively modeling the workflows used by end-users in the application. The knowledge gained through this particular iteration can be applied to other test automation projects that need to ensure that their testing efforts match the way end-users actually use the system, as modeled by UML diagrams.
Principle 2: Theory-Ingrained Artifact. This BIE iteration demonstrates that a test automation framework can not only be used for technical tasks, such as ensuring that the application is tested correctly, but can also inform and demonstrate advanced relational concepts that exist in the SUT. Insights into how the elements of the application fit together can be gained by examination of the tests and surrounding structure, so that the automated testing framework reflects, more accurately, how the SUT operates, which can lead to more accurate testing models that represent how end-users utilize the application.

Principle 3: Reciprocal Shaping. The use of this artifact in the organization demonstrates that the extension of automated testing frameworks to include workflows can result in useful design practices that, when successfully implemented, can lead to a synthesis of information between the design of the SUT, the automated testing framework, and the eventual application to be tested. Through continued use and improvement of these design patterns, the SUT and the tests themselves can begin to develop a shared character, where the end-result is the ability of the tests and test framework to match and reflect similar experiences, so that changes that occur in the SUT are discovered by their contradictions found in the tests and related testing frameworks.

Principle 4: Mutually Influential Roles. As a researcher, the primary investigator on this research project was able to discover the workflow patterns that existed in the UML diagrams. Through a thorough investigation of these documents, with an understanding of the state of the automated testing framework at the end of the first BIE cycle, the researcher was able to discern that the activity and sequence diagrams contained additional information regarding these workflows, and how these workflows are reflective of business processes. As a practitioner, the primary investigator was able to implement these workflows in the framework code, and code
the tests in a way that took advantage of these framework changes, in order to determine the impact of these changes to the testing of the system.

Principle 5: Authentic and Concurrent Evaluation. The evaluation of the changes made to the ensemble artifact occurred as the elements were being built by the researcher. As a workflow was added to the framework, that workflow was also added to the appropriate tests, and the elements that workflow replaced were removed from the tests. The first few workflow elements to be added to the framework took some reengineering of some of the basic elements of the framework and test code. This workflow implementation does change some of the basic "rules" of automated testing frameworks, such as the idea that no testing should occur in the framework, but it was also determined that the best way to implement these frameworks, so that they can have maximum impact on, and provide maximum value to, the end-goal research questions, was to include assertions to ensure the test was on the right page for the data to be entered.

Principle 6: Guided Emergence. The concurrent evaluation process proved especially useful during this iteration, as it was determined, during the reengineering of the artifact to include workflow objects, that simple composition of objects into workflows was not enough to ensure proper testing of the SUT. The initial design choice was to simply create composite objects from the page objects created during the first BIE cycle. However, it was soon made clear that the design choice of workflows had to comprise multiple types of objects, including assertions to ensure that the test would be on the right page before implementing the page object's method of interacting with the page. So, while the initial design of workflows came from the use of UML diagrams, as theory-infused artifacts, the emergence of these workflows as hybrid page objects, that are composed of both page objects and elements of tests, led to design changes that improved the overall use of the framework in the tests. The change in initial design
requirements came from the need to ensure that the test would operate on the right page of the application, before entering the data required for the page. This is also correspondent to the UTP design elements of the use of Verdicts in tests, where a test will typically either pass or fail.

The Verdict is used to ensure correctness of the SUT, so it should focus on only identifying elements of the SUT for the determination of a final verdict. In this instance, for example, the previous BIE cycle included formative evaluation elements where the test would fail due to elements of the framework being inconsistent, or elements where the SUT would be in temporary states where it could not receive data from the test, resulting in failing tests, but not due to an incorrect SUT. Therefore, for each test, assertions of page availability and element accessibility were included. These assertions had to also be included in the workflows as well, even though, strictly speaking, assertions should only be found in the testing parts of an automated testing solution, and not in page objects of an automated testing solution. However, the choice was made to include these assertions in the workflows to ensure that each workflow would be on the right page, and that the right elements were accessible, before continuing with the workflow element, as a way to ensure SUT correctness.

Principle 7: Generalized Outcomes. This iteration focused the use of workflows. These workflow elements were incorporated in order to help focus on the research questions, where the UML Testing Profile is being researched in order to create an automated testing framework that models the SUT closely, allowing for resilient tests to be made that accurately portray the state of the application. The solution, for this iteration, was the incorporation of workflows, made via compositional methods, that allow for multiple page objects to be used inside of a single workflow, and that also incorporate assertions that ensures the workflow operates on the right page to be tested, and that the elements for that page area available for use. The design principles
to be derived from this iteration include the use of these workflows, as compositional objects, that can also ensure the test reaches the correct verdict for each test element. These design principles can help ensure that a test automation framework more closely resembles the application it tests, and is also robust enough to return the correct results when used for testing.

An additional positive effect of this solution is that the tests used are much more concise, clear, and easy-to-read, which incorporates design elements of ensuring that the more complicated elements of the solution are hidden in the framework, as the framework is more of the domain of the developer, reserving the tests as the domain of the testers that may not need to know the intricacies of the framework. This solution also promotes code reuse, by making a single workflow available for many tests, easing the amount of maintenance required to keep a test active and accurate, and adding clarity to the tests that use those workflows.

3.5.3 Iteration III: Crafting the artifact

3.5.3.1 Problem formulation. This focus of this iteration was to use the ensemble artifact in-practice, by implementing tests based on the use cases. In this way, the framework created could be thoroughly tested, by extending the testing performed, and, in this manner, test the framework for completeness and reliability beyond the basic use cases covered thus far. These tests could be thought of as extensions of the tests performed up to this point.

It was decided that this was a necessary iteration for this research, so that the built and re-engineered framework could be analyzed for extended use and suitability. The testing performed using the re-engineered workflows indicated that the framework was, indeed, suitable for covering the basic use cases included using the given specifications. However, it was also noted that these use cases did not paint a complete picture of the SUT, in that there were additional states of the marriage case object not covered by these basic use cases. The basic elements of the
object's behavior are tested in the existing use cases, but object state is another important factor to consider when examining test cases.

State and behavior are important elements in object-oriented programming, and any testing performed on a system should include tests of state and behavior of the main objects in the system. For our SUT, the primary object that needs testing is the marriage case. The marriage case in the application has several states in which it can be, depending upon the data saved for the object, so tests that manipulate the states of the marriage case object can be very effective at determining proper system operation.

**3.5.3.2 Building, intervention, evaluation.** In order to generate these new tests, in a meaningful and thorough way, the complete behavioral specification for the marriage case object created, retrieved, and updated by the existing tests needed to be known. UML state machines are a typical way to specify this behavior. Testing the relationships between data and objects that make up an object's state can be accomplished by studying state machines, determining the values that describe the object at the various points in the life of the object, from its initial state to its final state, as well as the events that take place in order to change the states of the object.

It is important to understand what kinds of information are in a state machine and what kinds of data can be extracted for use. State machines may not be useful for all objects in an application but may only be used to describe the more complex objects in order to help simplify their design and ensure proper values, events, and transitions from state-to-state are embodied by the class in the application. As a diagram, state machines identify the different states an object can take, and what events cause them to change from state to state. In this way, the dynamic aspects of a single class can be modeled, and the evolution of an object can be tracked over time.
The state machine for a marriage case is shown in Figure 11. This state machine shows that there are multiple states a marriage case can be in at once, that are displayed as the statuses of the case, and shown in swim lanes for the state machine. The statuses are shown to transition from state to state based upon the events, or actions, that change the values of the different statuses, so that the transitions in state, from initial to final state.

*Building the tests.* In order to translate these states and transitions into tests, each of the statuses to be tested were given their own use case, where the status tested was transitioned through its various states to ensure that the guard conditions were met for each transition. For each status type, the initial state, and all transitions to other states, were tested independently of each other. This testing primarily offered the researcher the chance to ensure that the framework elements would work with additional testing that would occur that uses the framework. By using a UML state machine as a guide, it was believed the formulation of these tests would help put the framework through a more thorough set of tests that would help find weak areas in the framework and determine how well the framework represented the SUT.

As the state machine diagram displays the different states an object can take, as well as information regarding the precise sequence of events that lead to transitions, meaningful sequences can be derived from studying these diagrams. It was hoped that this focus on state coverage and transition coverage would ensure that the critical elements of the framework would be tested and that faults would be found in order to help identify weak points in the framework.
Figure 11: State machine for the marriage class.
Intervention in the organization. The tests were implemented by the organization for use, along with the original three use case tests, in order to obtain a comprehensive evaluation of the artifact, in-practice, in the actual use setting for the framework. These tests were combined in order to form a comprehensive evaluation of the framework, by being able to use these framework elements more continuously and more thoroughly.

Artifact evaluation. The evaluation of the artifact in this, more broad context, highlighted a few weaknesses in the framework. For instance, there were a few methods by which the individual page objects interacted with the elements on the page that needed to be reexamined in order to provide more stability and repeatability in the framework. One of these methods was the method by which data was entered in a predictive text field and then selected from the choices given, in order to enter data into additional fields that rely on that predictive field. The issue at hand was that, whereas, the predictive text was being selected, the SUT was unable to enter the required data in the fields fast enough for the test, and the test was moving on to the next element or page of the application, not letting those field be completely filled in by the application.

Additionally, several other framework elements had to be constructed in order to support this greater variance in testing. Included in this was a way to select the marriage case from a list of search results. The original framework only included a way for a test to select the first marriage case from a list, but the greater number of tests also indicated that greater flexibility was required in order to select a case based on the name of the first party to the marriage.

Other changes to the framework were changes to how the workflows were designed. It was found, by additional testing, that, in order to use the workflows for other tests outside the use cases coded in the first and second BIE cycle, some of the workflows had to be edited slightly by removing their assertions for checking the page to which the test returns after printing the
These kinds of changes were anticipated, as the framework matured and the tests that utilize the framework were expanded in scope. Comments were added to the framework and test elements as well, as was a suggestion indicated in the evaluation of the previous BIE cycle.

The code metrics for this iteration were interesting. For example, the average time taken to complete the use cases dropped by 33%, 9.1%, and 17.3% (for each use case, respectively) compared to the previous iteration. Additionally, the calculated maintainability index of the overall project remains unchanged at 70. The maintainability index of the test project did decrease five points to 58, but the framework gained a point in its maintainability index rating to 71. The total source lines of code (SLOC for the total project increased to 986, with 153 of them being in the tests, and the other 811 of them existing in the framework.

<table>
<thead>
<tr>
<th>Test Date</th>
<th>Use Case 1</th>
<th>Use Case 4</th>
<th>Use Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIE 1 Average</td>
<td>1:19.7</td>
<td>1:22.5</td>
<td>0:39.2</td>
</tr>
<tr>
<td>BIE 2 Average Time</td>
<td>1:27.6</td>
<td>1:18.0</td>
<td>0:37.0</td>
</tr>
<tr>
<td>11/07/2016</td>
<td>0:49.8</td>
<td>1:10.5</td>
<td>0:39.0</td>
</tr>
<tr>
<td>11/08/2016</td>
<td>0:52.5</td>
<td>1:07.8</td>
<td>0:28.6</td>
</tr>
<tr>
<td>11/09/2016</td>
<td>0:53.5</td>
<td>1:09.0</td>
<td>0:28.4</td>
</tr>
<tr>
<td>11/10/2016</td>
<td>1:21.9</td>
<td>1:17.2</td>
<td>0:29.5</td>
</tr>
<tr>
<td>Test Date</td>
<td>Use Case 1</td>
<td>Use Case 4</td>
<td>Use Case 5</td>
</tr>
<tr>
<td>11/11/2016</td>
<td>0:55.9</td>
<td>1:10.0</td>
<td>0:27.5</td>
</tr>
<tr>
<td>BIE 3 Average Time</td>
<td>0:58.7</td>
<td>1:10.9</td>
<td>0:30.6</td>
</tr>
<tr>
<td>Manual Testing</td>
<td>06:11</td>
<td>01:14</td>
<td>02:17</td>
</tr>
</tbody>
</table>

*Table 3:* Time, per test, per day tested, average time taken per test, and the time spent manually performing each test, for BIE cycle 3.
3.5.3.3 Reflection and learning.

Reflection on how the initial design and redesign impacted testing. Many automated testing frameworks are built with a pre-knowledge of the as-is system, so that the end result is a framework built around a testing team's conception, and sometimes misconception, of how an application works, with all of the compromises that are made, assumptions that are inherited, and adjusted expectations that come with the actual building and use of a Web-based application.

This framework was built, primarily, based upon UML diagrams. These UML diagrams model the intended workflows, expected behaviors, and purpose-built relationships that should exist in the system. They also present to the testing team the business processes that should be present in the final application under test. Whereas, this automated testing framework was developed for use on an existing application module, using practitioner knowledge of some of the elements present in the module, the framework was also built for a module on which no previous automated testing had been performed. In order to more fully test these framework construction concepts, it would be helpful to construct a framework the same way for additional application modules, and, perhaps most helpful, for modules not yet developed.

Adherence to principles. Principle 1: Practice-inspired research. Building upon the information gained in the previous iteration, this iteration was an attempt to understand how stable, reliable, and reflective the framework was. This iteration was driven by the need for thoroughly testing the framework in-practice, using test cases that would test actual, important elements in the application. By focusing on the various states and transitions present in the application, this iteration was able to identify important areas in the framework that had not been totally tested in the previous iteration. For example, some elements of the workflows were edited to be made more flexible, based upon the testing performed in this iteration. Whereas this change resulted in more verbose tests, compared to the previous iteration, the workflows, on the whole,
became more easily edited and usable in regards to the other tests built and tested in this iteration.

Principle 2: Theory-ingrained artifact. The literature review describes successful frameworks that are both flexible and able to successfully represent the SUT, so building a framework based off of UML diagrams makes sense. As this was the third iteration, much of the artifact has been shaped by the previous cycles of intervention, evaluation, and reshaping. Many of the design elements identified in this iteration are based off the organization's use of the framework and the tests, but the new tests themselves are still founded on the basis of the UML state machine. The state machine for the marriage case class guided the design of the tests, in order to fully exercise all of the transitions identified in the diagram, testing all possible states for the class.

Principle 3: Reciprocal shaping. Reciprocal shaping played a large role in this iteration, understanding that the IT artifact and the organizational context both influenced each other during the cycle. During this iteration, the recursive cycles of building, intervention, and evaluation became finely grained to a point where it was possible to identify more granular tests that could be performed, and, as such, more fine-tuned testing of the artifact could also be performed. In this way, the framework mutated into its current form, where it is more capable of flexibility while still adhering to good design principles. The tests, being informed by the theory-laden state machines, were constructed and executed by using the theory- and practice-formed framework.

Principle 4: Mutually-influential roles. The research that went into the study of the state machine for the marriage case was pivotal in being able to identify areas of testing that had not been covered yet. Using the researcher's knowledge of state machines was helpful in determining
which elements of the SUT required more attention and testing, and which did not. At the same time, the practical knowledge, of how to implement these tests, using the framework elements, in a way that covers the application in an adequate way, also provided a lot of usable information for this iteration. In this way, both theoretical knowledge and practical understanding helped shape this iteration.

Principle 5: Authentic and concurrent evaluation. As the tests, derived from the state machine diagram, were being coded, the framework was also being modified, to ensure that the tests would run effectively. In this way, the framework was being evaluated as it was being used for developing the tests. The continuous evaluation that the ADR process provides made it possible to develop this ensemble artifact quickly, with some aid from the pre-knowledge of the other application modules. Also, after the tests were created, on-going updates to the framework were made, in order to extend and test the utility of the framework during this BIE cycle. The use of the state machine allowed the researcher to identify tests that assisted in identifying the value of extending the framework. These tests help show the worth of this framework by the use of the framework in such a flexible way.

This iteration also tested the framework in an authentic way, relying on multiple tests over the course of multiple days, that tested multiple elements of the module. This authenticity also assisted in the redesign of some of the framework elements, by testing the application using multiple execution paths while still being able to rely on the workflows developed during the previous iteration. Due to the flexibility required to test these execution paths, some elements of the framework designed in the first two iterations required re-engineering and refinement.

Principle 6: Guided emergence. By expanding the testing in this iteration, the state machine diagram for the primary object in the module provided a good basis for the redesign of
the framework itself. The emergent elements that came out of this iteration included the framework edits, that, ultimately, made the framework both stronger and more flexible. The use of design patterns strengthened the framework through refactoring and redesign.

The design of the state machine, and its use for test generation, lead to additional test cases that naturally exercised the framework in ways that it had not been in previous iterations. These design changes also reflect the original guiding principles for the design of the framework, namely code reuse and a reduction in the maintenance of framework elements. The results of this intervention include more tests and a more flexible framework, where the framework models the SUT accurately and in ways that become more self-documenting and readable, which helps to reach many of the end-goals for this project.

Principle 7: Generalized outcomes. This iteration realized the value of extended testing with, and of, the framework. This iteration also assisted in the formulation of a test automation framework using more of a bottom-up approach, looking at the tests first, and then building a framework to meet those needs, whereas the previous iterations focused more on a top-down approach, where the focus was on the framework elements.

The problem instance faced in this iteration revolved around the implementation of the automated testing framework in a larger context. The previous two BIE cycles focused more on the creation and redesign of the framework, while this iteration took a look at the use and progression of the framework design by utilizing the framework for additional system testing. This iteration was crucial for showing the natural learning curve that occurs with framework design and implementation, where elements of the artifact are maybe not redesigned from basic building blocks, but rather the elements are already sufficiently partitioned, so that these smaller elements may be edited.
This iteration also displays the inherent growth problems that arise during the continued and expanded use of a solution. In order for positive growth to occur, some of the very basic elements of the artifact needed inspection and reorganization, as noted above.

The solution drawn from the iteration can be generalized in several ways. First, many of the elements of this solution were edited using a bottom-up evaluation of the artifact, not a top-down evaluation like in previous iterations. This allowed the researcher to inspect the minute, underlying details of the artifact, inspecting them from a different point of view, seeing connections that had not been previously seen and identifying problems that had not been previously identified.

This iteration also allowed for a restatement of the problem. Some of the issues discovered in this iteration were only discovered by a restatement of the original topic. This is an important point, because this restatement iteration included more design research and thought, not simply adding code to the artifact. In order to be successful with this iteration, the organizational goals had to be restated and reviewed, and the artifact had to be examined for how it met those stated goals.

Experimentation was also part of this iteration in a way it had not been before, that resulted in positive outcomes. The ability to take the time and resources necessary to genuinely experiment, as part of a controlled process, enabled many of the final outcomes of the project. The interactions between the data, the tests, the framework, the toolkit, and the browser are complex, and not all results of the different ways in which the artifact could be developed are apparent. Thusly, experimentation is almost required in order to determine the best route to take in order to achieve optimal results. Throughout this iteration, experimentation played an important role in how the details of the artifact work.
4. Formalization of Learning and Generalization of Outcome

Abstraction of Learning Into Concepts for a Class of Problem

In the initial phases of this ADR process, the class of problems all related to the best way to build an initial automated testing framework that represents the system being tested in a reliable and accurate way. In order to investigate these problems, goals were set for each iteration, and design principles were identified, that research suggested would result in an ensemble artifact that would be a faithful representation of the SUT by the automation framework, using UML and the UML Testing Profile as the guide.

Through the literature review, as well as the iterations, several things became apparent. One of the primary concepts to be garnered from this research is to understand a framework's place in an automated testing solution (Figure 12).

![Figure 12: Where the framework sits in relation to other elements of an automated testing solution.](image)

The figure above shows where the solution elements fit in the overall solution, and where those elements meet. For example, starting with the data elements of the solution, the only solution element that interacts with the data is the test element, and the tests retrieve the data via data tables constructed in the initial stages of the test. The tests, in turn, use elements of the framework, initialized by each test. The framework uses the Selenium toolkit via the toolkit's API, and the toolkit's API is able to manipulate the browser via the WebDriver for that particular browser. It is important to note that this kind of architecture only allows communication between
elements that are next to each other, so that, for instance, data is never passed directly to the framework itself, but is always passed via a test. Likewise, the Selenium toolkit will never be manipulated directly by a test, but will always use framework elements, via control over the toolkit API, in order to achieve its goals.

**Share Outcomes and Assessment with Practitioners**

The researcher for this project is also a practitioner in the organization where the artifact was developed, and the researcher-as-practitioner is a primary intended user. The researcher was able to share some of the findings and outcomes of the research with other organizational practitioners during the research process. The outcomes and assessment of the solution have been shared internally with interested parties in the organization, as well as with practitioners outside the organization.

Internally, the outcomes and assessment will be shared with interested parties in several ways. First, the written material, including this project document, as well as summative method instructions, included in a company-wide wiki document, will be used by the organization, moving forward, as a basis for further development. This will ensure that the project will continue even after the end of the academic elements of this project, so that the organization can realize full use from this artifact. It is intended that this artifact be put to use by the organization moving forward, and the organization intends to replace the existing framework with this new, updated, framework. Additionally, the framework will be presented, orally, to others on the project team, including the developers of the application being tested, for their further input and comments.

Externally, the insights and information that have been gathered during this research have been and will be disseminated to several groups. First, an oral presentation was given to a local
user group, where the methods explored for construction of frameworks was explained, along with an exercise in framework construction for the participants, led by the researcher. It is also planned that this research will be the focus of a widely-distributed podcast titled *Test Talks*, where the researcher will be interviewed by the podcast’s host regarding the research performed and the lessons learned. Additionally, this research will be available through the Dakota State University research database, where it will be made available to all students for further inspection and education.
Articulation of Outcomes as Design Principles

**Design Principle 1: Identify a test architecture and a test context.** In order to understand what your overall automated testing solution and framework comprises, making sure to identify the overall test architecture and test context is important. The identification of a test solution and test framework are primary elements of the UTP, and doing so assists with identifying the critical elements that comprise an automated test framework. The test architecture defines all of the concepts required for performing tests, including the test context as well as elements for defining the tests to be included. The test context is the setting in which test cases can be grouped (such as unit, integration, or system-level), as well as test controls. These basic elements make the implementation of the framework easier by setting not only boundaries and expectations for performance.

**Design Principle 2: Use of UML diagrams for object and method identification.** The use of UML diagrams in a systems development project can help teams conceptualize and document the functional requirements, basic processes, workflows, and relationships internal to and external to a new information system. Not only are the relationships between the various objects inside a system described and illustrated by the various UML diagrams, but the relationships between external actors and other systems are documented as well. These visual design specifications can be used for modeling test automation frameworks as well.

There is a vast amount of information that can be derived from these UML diagrams for the design and construction of automated testing framework artifacts as well. These UML diagrams can assist in the realization of these artifacts by informing the practitioner of the various page objects and methods required in order to effectively model the system. By the active use of these UML documents, an automated test framework that is reflective of the SUT
can be developed that accurately portrays the essential processes, objects, and workflows required for extensively testing the system.

**Design Principle 3: Workflow modeling.** The UML documentation generated for the development of the information system to be tested can also assist with the identification and generation of framework elements that reflect the various workflows implemented by actors using the system. The modeling of these workflows by the test automation framework is important for several reasons. One benefit of this workflow modeling is the fact that the resulting tests that implement the framework components can be easier to read by the use of workflows, while still retaining the flexibility required to thoroughly test the system. Secondly, the use of workflows in the framework can also assist the practitioners using the framework with understanding how the system is expected to be used, as these framework workflows reflect the workflows found in the UML documentation. Finally, the workflows built into the framework, when built and implemented correctly, result in more accurate testing, by testing the application as it is built to be used.

5. Project and Methodological Reflection

**Timelines**

This project was initiated in October of 2015, with the preliminary work being done in researching the possible benefits of completing a project like this for the organization, as well as researching the possibility of working with the organization to produce this project. As of January 2016, the organization had approved the project ideas, and the preliminary ideas for this project started to take shape.

Action Design Research is a time-intensive method, with several areas of the research taking up significant amounts of time. As this project was undertaken by a single researcher,
some of the elements of the project, for example, the initial building of the alpha-version of the artifact, took up a larger portion of the total time taken than others. Additionally, the time spent letting the practitioners use the artifacts in order to provide organizational feedback took up a larger-than-expected amount of time, which slowed the process down, primarily due to a change in organizational personnel over the life of the project. This is a natural consequence of working directly with an organization, and, whereas, it did extend the dates by which elements of the project were complete, it also allowed the researcher to gain valuable feedback from the organization that was fresh, direct, and without judgment in relation to previous solutions that may have existed in the organization.

**Scope**

Originally, the scope of the project was to include many more use cases and, consequently, tests, than what was submitted as the research proposal. Through internal and external discussions, it was decided to focus on the three primary elements of the specific application module to be tested, in order to limit the research to the most important elements. This allowed the researcher to focus efforts on exploring new application elements that have not been previously tested, benefiting both the research goals as well as the organization that implemented the ensemble artifact.

With Action Design Research being iterative in nature, a narrow focus helped the researcher explore further the building blocks of the artifact and ultimate solution, instead of losing focus by requiring additional tests that may not have provided additional insight into the original research problem. This allowed for more incremental development of the final solution, similar to iterative prototyping, where heuristic models could be developed and investigated more thoroughly.
The decision to focus on a new application module, rather than an existing module, was also advantageous to the outcome of the research. By focusing on a previously-untested application module, the researcher was able to approach the problem in a novel manner, with a fresh viewpoint on the system under test. This allowed for a more natural progression of ADR stages, with less supposition of perceived, pre-existing solutions, where the intervention and evaluation could be more authentic in nature without assumptions being made.

**Action Design Research**

The ultimate desire for this project was to complete a research project that added valuable knowledge to the existing body of knowledge. As such, the Action Design Research method was chosen as a way to both solve current practitioner problems as well as contribute to the theoretical knowledge base. With the core of the information systems discipline being the development and use of artifacts, as part of organizations, the use of the ADR methodology was a good choice for being able to incorporate both elements into a single research project. The ability to use the ensemble artifact designed in-context, inside of an actual organization, beyond simple design basic design knowledge, made it possible to evaluate the artifact in an authentic way, leading to a greater understanding of the design principles involved in the final solution.

**The UML Testing Profile**

The UML Testing Profile was chosen as a focus for this research primarily due to the lack of existing research regarding the topic. Certainly, there has been previous research regarding the UTP, but not much recent information was found regarding the use of the UTP for testing more modern web-based applications, and almost nothing was found that related directly to the use of UTP for the design and implementation of an automated testing framework.
There are, however, additional elements of the UTP that were intentionally left out of this research. For example, the UTP also describes the use of data in more detail, utilizing concepts like default data values and the use of a data pool. Most of the UTP elements that were not explored in this research were excluded in an attempt to keep the scope of this research narrow and, thereby, more valuable. Certainly, there have been no warrants made regarding the full coverage of the UML Testing Profile by this project, with an understanding that the importance of this research is based on the ability to explore the components required to build the ensemble artifact, intervene in the organization, and evaluate the results. Therefore, some elements of the UTP were naturally left untouched in this research.

6. Discussion

Throughout this project, design principles for automated testing frameworks using UML and the UTP as guiding principles were discovered. This project shows that the identification of a Test Architecture and Test Context are important for identification of the primary elements of the framework. Also, the use of UML diagrams for framework and method identification can help practitioners model the system to be tested, helping to ensure that the framework consists of the right elements for relaying the essential processes, object, and workflows required for testing. The modeling of workflows in the framework is also very helpful, being informed by UML diagrams, and being able to model the basic processes through which objects in the system change state.

Previous Work

Previous writing on the use of the UTP has generally revolved around several ideas. The UTP has been shown to be effective for unit testing and test case generation for example. Additionally, UML and the UTP have been shown to be helpful in applying model-based testing,
that utilizes additional components not included in this research. Several authors have also delved into the use of UML for the purposes of automated test generation. However, not much focus has been previously placed on the use of UML and the UTP for the generation of test automation frameworks themselves.

**Implications**

The evidence gleaned from this project includes the ensemble artifact, as built, used, and evaluated over the span of three iterations, as used and evaluated in an organizational context, in order to achieve authentic and concurrent evaluation. Currently, this evidence suggests that the UML Testing Profile is capable of being used to generate reliable automated testing frameworks that can be used for testing, inside of a data-driven context, in order to test highly advanced web-based applications. This project suggests that the UML diagrams included as part of the initial design of an application can also be useful in the generation of the framework on which testing of the system is delivered, by revealing not only the various objects that a user would interact with in a system, but also the methods with which external users would interact with these objects.

Furthermore, it has been demonstrated by this project that the workflows and business processes important to the organization can be modeled using UML diagrams, and that as these key operations are ascertained, they may also be modeled in the framework, helping to ensure that the framework, as the basis for the automated testing solution, represents the system accurately and efficiently.

The alternatives to using the UTP for framework construction may not be as favorable, as they include shifting the testing and test solution generation process later in the development cycle, as the objects, relationships, methods, and messages displayed in the UML diagrams...
become actual application components, services, and classes. The intervention techniques explored and discussed in this research can help a quality-assurance testing team, as part of a larger team of developers, business analysts, project managers, and others, ensure that the testing performed is performed sooner in the development process, as well as more accurately, with tools that is more reflective of the system being tested.

The measurements for success for implementing an intervention such as this include the ability to test the application sooner in the development process, instead of having to wait until the system to be tested is a fully-realized and working system. Additional measurements that can be taken to measure progress include the discovery of defects sooner in the testing process, the discovery of defects along primary execution paths in the system, and, ultimately, high levels of end-user satisfaction with the product. These criteria can all point to the positive effects of using the interventions and design principles as described above.

Further Studies

Further investigation in this area is certainly warranted. With the role of automation in systems development projects increasing, the need for robust frameworks that support testing won't decrease anytime soon. While the research undertaken here outlays certain design principles used to achieve measures of success that can be applied to a broader class of problems, this is still one ensemble artifact built in a single specific context. Further research should be applied in additional scenarios, using the design principles detailed here, in order to ascertain their application across more scenarios, where further design principles may be distilled and refined.

More testing of the UTP is also warranted in order to extend the investigation and use of additional UML diagrams for framework creation, as this research is far from exhaustive in this
area. Similarly, additional UTP concepts can be added to research efforts in order to ascertain their value and place in the creation of modern, data-driven testing frameworks.

Additionally, as of the time of this research, the Object Management Group, as the governing body of the UML specifications, is working on an updated version of the UTP. This update will surely bring with it changes to the UTP specifications that will need to be researched and tested through further research.

7. Conclusion

Through an initial review of literature, the following two research questions were posed:

Q1: How can the UML Testing Profile be implemented in order to create a robust automated test framework that addresses the concerns of testing modern, web-based applications?

Q2: How can this framework be implemented in-practice?

Investigating this first question led to the inspection and examination of UML diagrams. The research concluded that test automation frameworks need to approach modern, web-based applications in ways that address the specific concerns found in Web 2.0 technologies in ways that promote code reuse and ease of maintenance.

The design principles detailed above represent a good start for this framework construction. The use of these principles has been shown, through this research, to generate a test automation framework that meets these goals while also being language-agnostic, making the approach adaptable in relation to the specific needs of the project where it is applied.

It is important for any practitioner applying UML Testing Profile concepts to understand that the context, including the system to be tested as well as the end-goals for the testing to be performed, will influence the final design and implementation of any test automation solution,
and the design principles outlined in this research can assist in achieving the goals of a framework.

This research adds to the base of knowledge on the implementation of the UML Testing Profile in several ways:

- The same UML diagrams that are used to specify requirements for the creation of an information system can also be used to create the framework used to test the application.
- UML diagrams contain valuable information for the creation of page objects as well as methods for interacting with the pages of the Web application being tested.
- Those same UML diagrams can also help practitioners identify the workflows that can be modeled by the framework, representing the critical business processes and manipulation of object states, thereby representing the application being tested more accurately, improving testing efforts.

The second question involves the use of the ensemble artifact that was developed via the ADR iterations by an organization, in-practice. During the ADR process, the artifact was tested and evaluated by the creation of tests that implemented the framework artifact. Practically speaking, the artifact has been shown to be effective in supporting the goals of test automation. The framework provides methods for exhaustive testing to be performed, providing correct and complete methods for interacting with the system being tested, while also ensuring that the tests using the framework are separate from the toolkit and browser used, while also being malleable and reusable.

For practitioners, one of the more valuable parts of this implementation is the fact that the framework is created as an accurate model of the SUT. This feature of the methodology allows testing teams to exercise the SUT in a way that is consistent with how end-users are most likely
to use the system, and how the system was developed for use. The design principles outlined in this research may help practitioners find defects faster, compared to other methods of framework construction, by ensuring that the application is being covered thoroughly, in accordance with the initial system design documents. It is also possible to uncover defects in a more effective manner, as the tests, using the framework, have access to the workflows and business processes built into the framework. Finally, testing with a framework built using these design principles, helps ensure that the defects discovered are defects that would ultimately impact the end-users.
Appendix A

UML Use Case Diagram, Activity Diagrams, Communication Diagrams, and Use Case Descriptions for Selected Use Cases

Use Case Diagram for the Electronic Marriage Registration System
Use Case 1 Activity Diagram
Use Case 4 Activity Diagram
Use Case 5 Activity Diagram
Use Case 1 Sequence Diagram
Use Case 4 Sequence Diagram
Use Case 5 Sequence Diagram
USE CASE 01 Create a Record

CHARACTERISTIC INFORMATION

Brief Description
This use case describes the function of entering and editing data for the purpose of creating a Marriage Record in the EMRS.

In the most common scenario, the couple to be married apply for a marriage license at a county clerk office, where an internal clerk would enter data from a paper application into the system by using this use case.

This use case is used to capture information needed for the creation of a marriage license and subsequent certificate on behalf of county clerk offices.

Primary Actor
Internal: Marriage Registrar

Stakeholders
Internal: Marriage Registrar
County Clerk
Party to Marriage A
Party to Marriage B

Trigger
The actor receives marriage data to initiate a Marriage Record

Preconditions
An application for a marriage license is made to a county clerk office

Guarantees

Success End Condition
A new marriage record is created

Failed End Condition
A new marriage record is not created

MAIN SUCCESS SCENARIO
1. The User selects option to create a marriage case
2. The system displays option to start a marriage case
3. The User enters case identification data and selects the option to search
4a. The system validates that all required information has been entered and affirms that no duplicate cases exist
5. The User selects the option to start a new marriage case
6. The system displays the marriage registration pages, assigns a unique identifying case number, and creates a log of the event
7. The User enters marriage license information and selects to validate all data entered
8a. The system validates all data, assigns the appropriate license and marriage statuses, and updates the event log

EXTENSIONS
*a. The User decides to quit:
   *a1. The system discards any unsaved data and exits [fail]

4b. The User does not enter all required case identification data and select the option to search:
   4b1. The system notifies User that case information is still required [repeat at Step 3]

4c. The User enters invalid case identification data and selects the option to search:
   4c1. The system notifies the user that data is invalid [repeat at Step 3]

4d. The User enters data for a previously-started record:
   4d1. The system displays a list of all possible duplicate entries
       4d1a. The User selects the option to Update a Record

8b. Invalid information:
   8b1. The system displays a validation error for each element of invalid data [repeat at 7]
       8b1a. The User selects to override validation errors [repeat at 8a]
### MERS Rules

<table>
<thead>
<tr>
<th>001</th>
<th>The system tracks statuses for the record in the system based on business rules below:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For License Information</td>
</tr>
<tr>
<td></td>
<td>New Event - A new marriage case has been started but has not been validated</td>
</tr>
<tr>
<td></td>
<td>License Invalid - Information for one or more required license items is not valid</td>
</tr>
<tr>
<td></td>
<td>License Invalid with Exceptions - Information for one or more required license items</td>
</tr>
<tr>
<td></td>
<td>is not valid but has been deemed acceptable as-is</td>
</tr>
<tr>
<td></td>
<td>License Valid - Information for all required license items is valid</td>
</tr>
<tr>
<td></td>
<td>For Marriage Information</td>
</tr>
<tr>
<td></td>
<td>New Event - A new marriage case has been started but has not been validated</td>
</tr>
<tr>
<td></td>
<td>Marriage Invalid - Information for one or more required marriage items is not valid</td>
</tr>
<tr>
<td></td>
<td>Marriage Valid With Exceptions - Information for one or more required marriage items</td>
</tr>
<tr>
<td></td>
<td>is not valid but has been deemed acceptable as-is</td>
</tr>
<tr>
<td></td>
<td>Marriage Valid - Information for all required marriage items is valid</td>
</tr>
</tbody>
</table>
USE CASE 04 Issue License

CHARACTERISTIC INFORMATION

Brief Description
This use case describes the function of affirming and issuing a marriage license for the purpose of creating a Marriage Record in the EMRS.

In the most common scenario, the issuance of a marriage license will occur after a marriage case has been created and payment has been submitted, where an internal clerk would affirm the record and then approve and issue, the license.

This use case is used to affirm valid intentions of marriage and issue the marriage license on behalf of county clerk offices.

Primary Actor
Internal: Marriage Registrar

Stakeholders
Internal: Marriage Registrar
County Clerk
Party to Marriage A
Party to Marriage B

Trigger
The actor is required to issue a marriage license

Preconditions
A valid application for marriage has been entered in the system

Guarantees

Success End Condition
A valid marriage license has been issued

Failed End Condition
A valid marriage license has not been issued

MAIN SUCCESS SCENARIO

1a. The User Searches for an Existing Case
2. The User selects the option to approve the marriage license
3. The system displays the option to approve the marriage license
4. The User selects to affirm the marriage license
5a. The system verifies the user is authorized to affirm the license, saves the affirmation to the case and assigns the appropriate issuance statuses
6. The User selects the option to issue the license
7. The system displays the marriage license issuance page
8. The User enters information, including payment data, and selects the option to verify the data entered
9a. The system verifies all the data entered and updates the event log
10. The User selects the option to print the marriage license
11. The system prints the marriage license and assigns the appropriate issuance status

EXTENSIONS

*a. The User decides to quit:
   *a1. The system discards any unsaved data and exits [fail]

1b. The User Creates a New Case

5b. The User affirming the case is not authorized to affirm the license:
   5b1. The system displays a message informing the User they are an invalid user [fail]

9b. Invalid information:
   9b1. The system displays a validation error for each element of invalid data [repeat at 8]
       9b1a. The User selects to override validation errors [repeat at 9a]

MERS Rules

<table>
<thead>
<tr>
<th>002</th>
<th>The system tracks statuses for the record in the system based on business rules below:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For License Issuance</td>
</tr>
<tr>
<td></td>
<td>NA - A new marriage case has been started but has not been validated</td>
</tr>
<tr>
<td></td>
<td>Not Issued - The Marriage License data has not been affirmed</td>
</tr>
<tr>
<td></td>
<td>Ready to Issue - The Marriage License data is affirmed but the license has not been issued</td>
</tr>
<tr>
<td></td>
<td>Issued - The Marriage License data is affirmed and the license has been issued</td>
</tr>
</tbody>
</table>
USE CASE 05 Approve Certificate

CHARACTERISTIC INFORMATION

Brief Description
This use case describes the function of approving a marriage certificate filing for the purpose of making the case available to the State Registrar Office for registration and certificate issuance.

In the most common scenario, the approval of a marriage license filing will occur after a marriage license has been issued, the ceremony has been performed, where an internal clerk would enter data for the marriage and approve the marriage certificate.

This use case is used to approve a valid marriage on behalf of county clerk offices.

Primary Actor
Internal: Marriage Registrar

Stakeholders
Internal: Marriage Registrar
County Clerk
Party to Marriage A
Party to Marriage B
Officiant

Trigger
The actor is required to approve a marriage certificate

Preconditions
A valid marriage has been performed for a marriage where the license has been issued

Guarantees

Success End Condition
The marriage case is made ready for the State Registrar’s office to approve, register, and issue

Failed End Condition
The marriage case is not made ready for the State Registrar’s office to approve, register, and issue

MAIN SUCCESS SCENARIO

1. The User Searches for an Existing Case where a valid marriage license has been issued
2. The User enters marriage certificate information and selects to validate all data entered
3a. The system validates all data, assigns the appropriate marriage status, and updates the event log
4. The User selects the option to approve the marriage certificate filing
5. The system displays the option to approve the marriage certificate filing
6. The User selects to affirm the marriage certificate filing
7a. The system saves the affirmation to the case and assigns the appropriate marriage certificate status

EXTENSIONS

*a. The User decides to quit:
   *a1. The system discards any unsaved data and exits [fail]

3b. Invalid Information:
   3b1. The system displays a validation error for each element of invalid data [repeat at 2]
      3b1a. The User selects to override validation errors [repeat at 3a]

7b. The User affirming the case is not authorized to affirm the license:
   7b1. The system displays a message informing the User they are an invalid user [fail]

MERS Rules

<table>
<thead>
<tr>
<th>003</th>
<th>The system tracks statuses for the record in the system based on business rules below:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For Marriage Filing Approval</td>
</tr>
<tr>
<td></td>
<td>NA - A new marriage case has been started but has not been validated</td>
</tr>
<tr>
<td></td>
<td>Not Approved - The Marriage filing data has not been affirmed</td>
</tr>
<tr>
<td></td>
<td>Approved - The marriage filing data has been approved</td>
</tr>
</tbody>
</table>
Appendix B

Page Object Model Classes for BIE 1

BasePage.cs

using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;

namespace Framework
{
    class BasePage
    {
        protected readonly IWebDriver Driver;
        protected readonly By WaitAMomentMessage = By.Id(".blockUI.blockMsg.blockPage");
        protected readonly By CaseIdText = By.Id("Case Id");
        protected readonly By CaseStatusText = By.Id("Body_ctlRegistrationHeader_lblStatus");
        protected readonly By SearchButton = By.Id("Body_MPContent_MC1_btnSearch");
        protected readonly By StartNewCaseButton = By.Id("Body_MPContent_MC1_btnNewCase");
        protected readonly By ValidatePageButton = By.XPath("//input[@value='Validate Page']");
        protected readonly By NextButton = By.XPath("//input[@value='Next']");
        protected readonly By ClearButton = By.XPath("//input[@value='Clear']");
        protected readonly By SaveButton = By.XPath("//input[@value='Save']");
        protected readonly By ReturnButton = By.XPath("//input[@value='Return']");
        protected readonly By ComboBoxSelection = By.CssSelector(".RadComboBoxDropDown_Dave");

        protected BasePage(IWebDriver driver)
        {
            Driver = driver;
        }

        protected void Visit(string url)
        {
            if (url.Contains("http"))
            {
                Driver.Navigate().GoToUrl(url);
            }
            else
            {   

        }
protected IWebElement Find(By locator) => Driver.FindElement(locator);

protected void Click(By locator)
{
    Find(locator).Click();
}

protected void Type(By locator, string inputText)
{
    Find(locator).SendKeys(inputText);
}

protected void EnabledType(By locator, string inputText)
{
    if (IsEnabled(locator))
        Find(locator).SendKeys(inputText);
}

protected void DisplayedType(By locator, string inputText)
{
    if (IsDisplayed(locator))
        Find(locator).SendKeys(inputText);
}

protected void Replace(By locator, string inputText)
{
    Find(locator).Clear();
    Find(locator).SendKeys(inputText);
}

protected void Select(By locator, string inputText)
{
    new SelectElement(Find(locator)).SelectByText(inputText);
}

protected void SelectComboBox(By locator, string inputText)
{  
    Find(locator).SendKeys(inputText);  
    if (IsDisplayed(ComboBoxSelection, 10))  
        Click(ComboBoxSelection);  
}

protected void Check(By locator, string check)
{
    if (!string.IsNullOrEmpty(check))  
        Find(locator).Click();
}

public void SelectAllCheckBoxes()
{
    System.Collections.Generic.IList<IWebElement> ckOverrides =  
        Driver.FindElements(By.XPath("//input[@type='checkbox']"));
    
    foreach (IWebElement ck in ckOverrides)
    {
        if (ck.Displayed && !ck.Selected)  
            ck.Click();
    }
}

protected bool IsDisplayed(By locator)
{
    try {
        return Find(locator).Displayed;
    } catch (NoSuchElementException) {
        return false;
    }
}

protected bool IsDisplayed(By locator, int maxWaitTime)
{
    try {
        WebDriverWait wait = new WebDriverWait(Driver,  
            System.TimeSpan.FromSeconds(maxWaitTime));  
        wait.Until(ExpectedConditions.ElementIsVisible(locator));  
        return true;
    } catch (WebDriverException) {
        return false;
    }  
}
} catch(WebDriverTimeoutException) {
    return false;
}

protected bool IsEnabled(By locator)
{
    try {
        return Find(locator).Enabled;
    }
    catch (InvalidElementStateException) {
        return false;
    }
}

protected void SwitchToFrame(string locator) => Driver.SwitchTo().Frame(locator);

protected void SwitchToDefaultFrame() => Driver.SwitchTo().DefaultContent();

protected string GetTitle => Driver.Title;

protected string GetText(By locator)
{
    var text = Find(locator);
    return text.Text;
}

public string CaseId() => GetText(CaseIdText);

public string CaseStatus() => GetText(CaseStatusText);

protected void Search()
{
    Click(SearchButton);
}

protected void StartNewCase()
{
    Click(StartNewCaseButton);
}
public void ValidatePage()
{
    Click(ValidatePageButton);
}

public void Next()
{
    Click(NextButton);
}

public void Save()
{
    Click(SaveButton);
}

using OpenQA.Selenium;
using NUnit.Framework;
namespace Framework
{
class LoginPage : BasePage
{
    By LoginForm = By.Id("Body_MPContent_image");
    By UsernameText = By.Id("Body_MPContent_txtUserName");
    By PasswordText = By.Id("Body_MPContent_txtPassword");
    By SubmitButton = By.Id("Body_MPContent_btnLogon");
    By FailureMessage = By.Id("Body_MPContent_divError");
    By SubOfficeLink = By.Id("Body_MPContent_lstDomains_linkOffice_0");
    By LoginPanel = By.Id("Body_MPContent_UpdatePanel1");

    public LoginPage(IWebDriver driver) : base(driver) {}

    public bool FailureMessagePresent() => IsDisplayed(FailureMessage);
    public void GoTo(string target)
    {
        Visit(target);
    }
}
using OpenQA.Selenium;

namespace Framework
{
    class HomePage : BasePage
    {
        protected readonly By InformationWindow = By.Id("RadInformationWindow");
        protected readonly By OKButton = By.Id("Body_MPContent_MC1_btnClose");
        protected readonly By FastLinksPanel = By.CssSelector(".fast_links");

        public HomePage(IWebDriver driver) : base(driver) { }

        public bool PageDisplays() => IsDisplayed(FastLinksPanel, 10);

        public void CloseInformationWindow(string frame)
        {
            SwitchToFrame(frame);
            Click(OKButton);
            SwitchToDefaultFrame();
        }
    }
}
namespace Framework
{

class ValidationSummaryPage : BasePage
{
    protected readonly By ValidationSummaryPanel = By.Id("Body_ctlValidationSummary_dgValidationSummary");
    public ValidationSummaryPage(IWebDriver driver) : base(driver) { }

    public bool PageDisplays() => IsDisplayed(ValidationSummaryPanel, 10);
}
}

namespace Framework
{

class AffirmationsPage : BasePage
{
    protected readonly By AffirmationsPanel = By.Id("Body_MPContent_MC1_trAffirm");
    protected readonly By Affirmation1CheckBox = By.Id("Body_MPContent_MC1_cblAffirmations_0");
    protected readonly By AffirmButton = By.Id("Body_MPContent_MC1_btnAffirm");

    public AffirmationsPage(IWebDriver driver) : base(driver) { }

    public bool PageDisplays() => IsDisplayed(AffirmationsPanel, 10);

    public void Affirm()
    {
        SelectAllCheckboxes();

        Click(AffirmButton);
    }
}
using OpenQA.Selenium;

namespace Framework
{
    class MarriageRegistrationMenu : BasePage
    {
        protected readonly By OfficiantLink = By.LinkText("Officiant");
        protected readonly By ApproveLicenseLink = By.LinkText("Approve License");
        protected readonly By ValidateRegistrationLink = By.LinkText("Validate Registration");
        protected readonly By IssueThisRecordLink = By.LinkText("Issue This Record");
        protected readonly By ApproveFilingLink = By.LinkText("Approve Filing");

        public MarriageRegistrationMenu(IWebDriver driver) : base(driver) { }

        public void Officiant()
        {
            Click(OfficiantLink);
        }

        public void ApproveLicense()
        {
            Click(ApproveLicenseLink);
        }

        public void ApproveFiling()
        {
            Click(ApproveFilingLink);
        }

        public void ValidateRegistration()
        {
            Click(ValidateRegistrationLink);
        }

        public void IssueThisRecord()
        {
        }
    }
}
Click(IssueThisRecordLink);

TopBar.cs

using OpenQA.Selenium;

namespace Framework
{

class TopBar : BasePage
{

    protected readonly By MainLink = By.LinkText("Main");
    protected readonly By MessagesLink = By.LinkText("Messages");
    protected readonly By LifeEventsLink = By.LinkText("Life Events");
    protected readonly By BirthLink = By.LinkText("Birth");
    protected readonly By MarriageLink = By.LinkText("Marriage");
    protected readonly By SearchLink = By.LinkText("Search");
    protected readonly By StartNewCaseLink = By.LinkText("Start/Edit New Case");

    public TopBar(IWebDriver driver) : base(driver) { }

    private void Select(By topMenuLink, By subMenuLink)
    {
        Click(topMenuLink);
        Click(subMenuLink);
    }

    private void Select(By topMenuLink, By midMenuLink, By subMenuLink)
    {
        Click(topMenuLink);
        Click(midMenuLink);
        Click(subMenuLink);
    }

    public void SelectMessages(By MainLink, By MessagesLink)
    {
        Select(MainLink, MessagesLink);
    }
}
public void SelectBirthSearch()
{
    Select(LifeEventsLink, BirthLink, SearchLink);
}

public void SelectMarriageSearch()
{
    Select(LifeEventsLink, MarriageLink, SearchLink);
}
public void SelectMarriageStartNewCase()
{
    Select(LifeEventsLink, MarriageLink, StartNewCaseLink);
}

FilingOfficialPage.cs

using OpenQA.Selenium;

namespace Framework
{
    class FilingOfficialPage : BasePage
    {
        protected readonly By FilingOfficialPanel = By.CssSelector("div[code='MARRIAGE_FILING_OFFICIAL']");
        protected readonly By FilingOfficialComboBox = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOffical_plFilingRegistrar_radComboBox_input");
        protected readonly By FilingOfficialFirstNameText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOffical_plFilingRegistrar_ctlName_mdFirstName_mdFirstName");
        protected readonly By FilingOfficialMiddleNameText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOffical_plFilingRegistrar_ctlName_mdMiddleName_mdMiddleName");
        protected readonly By FilingOfficialLastNameText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOffical_plFilingRegistrar_ctlName_mdLastName_mdLastName");
    }
}
protected readonly By FilingOfficialSuffixText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plFilingRegistrar_ctlName_mdNameSuffix_mdNameSuffix");
protected readonly By FilingOfficialTitleDropdown = By.Id("Body_MPContent_MC1_dgLocalOfficial_plFilingRegistrar_ctlName_mdTitle_mdTitle") ;
protected readonly By FilingOfficialTitleOtherText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plFilingRegistrar_ctlName_mdTitleOtherSpecify") ;
protected readonly By LocalFileDateText = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdLocalFileDate_mdLocalFileDate_txtDate");

public FilingOfficialPage(IWebDriver driver) : base(driver) { }

public bool PageDisplays() => IsDisplayed(FilingOfficialPanel, 10);

public void Populate(string filingOfficialFirstName, string filingOfficialMiddleName, string filingOfficialLastName,
string filingOfficialSuffix, string filingOfficialTitle, string filingOfficialTitleOther, string localFileDate)
{
    SelectComboBox(FilingOfficialComboBox, filingOfficialLastName);
    Type(LocalFileDateText, localFileDate);
}
}

LocalOfficialPage.cs

using OpenQA.Selenium;

namespace Framework
{
    class LocalOfficialPage : BasePage
    {
        protected readonly By LocalOfficialPanel = By.CssSelector("div[code='MARRIAGE_LOCAL_OFFICIAL']");
        protected readonly By DateLicenseIssuedText = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdIssueDate_mdIssueDate_txtDate");
    }
}
protected readonly By WaiverSubmittedCheckbox = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdWaiverSubmitted_mdWaiverSubmitted");
protected readonly By DateLicenseEffectiveText = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdLicenseEffectiveDate_mdLicenseEffectiveDate_txtDate");
protected readonly By DateLicenseExpiresText = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdLicenseExpirationDate_mdLicenseExpirationDate_txtDate");
protected readonly By DateIntentionsFiledText = By.Id("Body_MPContent_MC1_dgLocalOfficial_mdIntentionsFileDate_mdIntentionsFileDate_txtDate");
protected readonly By CountyOfIssuanceText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrarOffice_radComboBox_Input");
protected readonly By IssuingOfficialComboBox = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_radComboBox_Input");
protected readonly By IssuingOfficialFirstNameText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdFirstName_mdFirstName");
protected readonly By IssuingOfficialMiddleNameText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdMiddleName_mdMiddleName");
protected readonly By IssuingOfficialLastNameText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdLastName_mdLastName");
protected readonly By IssuingOfficialSuffixText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdNameSuffix_mdNameSuffix");
protected readonly By IssuingOfficialTitleDropdown = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdTitle_mdTitle");
protected readonly By IssuingOfficialTitleOtherText = By.Id("Body_MPContent_MC1_dgLocalOfficial_plIssuingRegistrar_ctlName_mdTitleOtherSpecify_mdTitleOtherSpecify");

public LocalOfficialPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(LocalOfficialPanel, 10);
public void Populate(string licenseIssued, string waiverSubmitted, string licenseEffective, string licenseExpires,
string intentionsFiled, string countyOfIssuance, string issuingOfficial, string issuingOfficialFirstName, string issuingOfficialMiddleName, string issuingOfficialLastName, string issuingOfficialSuffix, string issuingOfficialTitle, string issuingOfficialTitleOther)

    {
        SelectComboBox(IssuingOfficialComboBox, issuingOfficialLastName);
        Type(DateLicenseIssuedText, licenseIssued);
        Check(WaiverSubmittedCheckbox, waiverSubmitted);
        Type(DateIntentionsFiledText, intentionsFiled);
    }
}
}

MarriageSearchPage.cs

using OpenQA.Selenium;

namespace Framework
{
    class MarriageSearchPage : BasePage
    {
        protected readonly By SearchCriteriaWindow = By.Id("Body_MPContent_MC1_searchCriteria");
        protected readonly By DateOfMarriageStart = By.Id("Body_MPContent_MC1_dtBegEventDate_txtDate");
        protected readonly By DateOfMarriageEnd = By.Id("Body_MPContent_MC1_dtEndEventDate_txtDate");

        public MarriageSearchPage(IWebDriver driver) : base(driver) { }
        public bool PageDisplays() => IsDisplayed(SearchCriteriaWindow, 10);
        public void ByDateOfMarriage(string domStart)
        {
            Type(DateOfMarriageStart, domStart);
            Search();
        }
        public void ByDateOfMarriage(string domStart, string domEnd)
        {
            Type(DateOfMarriageStart, domStart);
            Type(DateOfMarriageEnd, domEnd);
        }
MarriageSearchResultsPage.cs

using OpenQA.Selenium;

namespace Framework
{
    class MarriageSearchResultsPage : BasePage
    {
        protected readonly By SearchResultsPanel = By.Id("Body_MPContent_MC1_panelMarriageListInnerDiv");
        protected readonly By PartyANameLink = By.Id("Body_MPContent_MC1_dgMarriageList_PartyAFullName_0");

        public MarriageSearchResultsPage(IWebDriver driver) : base(driver) { }

        public bool PageDisplays() => IsDisplayed(SearchResultsPanel, 10);
        public void SelectFirstRecord()
        {
            Click(PartyANameLink);
        }
        public void StartNewCase()
        {
            Click(StartNewCaseButton);
        }
    }
}

MarriageStartNewCasePage.cs

using System;
using System.Collections.Generic;
using System.Text;
using OpenQA.Selenium;

namespace Framework
class MarriageStartNewCasePage : BasePage
{
    protected readonly By SearchCriteriaWindow = By.Id("Body_MPContent_MC1_searchCriteria");
    protected readonly By PartyAFirstNameText = By.Id("Body_MPContent_MC1_txtPartyAFirstName");
    protected readonly By PartyALastNameText = By.Id("Body_MPContent_MC1_txtPartyALastName");
    protected readonly By PartyBFirstNameText = By.Id("Body_MPContent_MC1_txtPartyBFirstName");
    protected readonly By PartyBLASText = By.Id("Body_MPContent_MC1_txtPartyBLastName");
    protected readonly By DateOfMarriageText = By.Id("Body_MPContent_MC1_dtEventDate_txtDate");
    protected readonly By CityofMarriageText = By.Id("ctl00_ctl00_Body_MPContent_MC1_ctlAddress_mdAddressCity_mdAddressCity_Input ");
    protected readonly By CountyOfMarriageText = By.Id("Body_MPContent_MC1_txtPartyAFirstName");

    public MarriageStartNewCasePage(IWebDriver driver) : base(driver) { }
    public bool PageDisplays() => IsDisplayed(SearchCriteriaWindow, 10);
    public void Populate(string partyAFirst, string partyALast, string partyBFirst, string partyBLast, string dateOfMarriage)
    {
        Type(PartyAFirstNameText, partyAFirst);
        Type(PartyALastNameText, partyALast);
        Type(PartyBFirstNameText, partyBFirst);
        Type(PartyBLASText, partyBLast);
        Type(DateOfMarriageText, dateOfMarriage);

        Search();
    }
}

OfficiantPage.cs

using OpenQA.Selenium;
namespace Framework
{
    class OfficiantPage : BasePage
    {
        protected readonly By OfficiantPanel = By.CssSelector("div[code='MARRIAGE_OFFICIANT']");
        protected readonly By DateOfMarriageText = By.Id("Body_MPContent_MC1_dgOfficiant_mdDateOfMarriage_mdDateOfMarriage_txtDate");
        protected readonly By CeremonyTypeDropdown = By.Id("Body_MPContent_MC1_dgOfficiant_mdCeremonyType_mdCeremonyType");
        protected readonly By PlaceOfMarriageCityText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlPlaceOfMarriageAddress_mdAddressCity_mdAddressCi");
        protected readonly By PlaceOfMarriageCountyText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlPlaceOfMarriageAddress_mdAddressCounty_mdAddressCounty_Input");
        protected readonly By PlaceOfMarriageStateText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlPlaceOfMarriageAddress_mdAddressState_mdAddressState_Input");
        protected readonly By OfficiantFirstNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdFirstName_mdFirstName");
        protected readonly By OfficiantMiddleNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdMiddleName_mdMiddleNa");
        protected readonly By OfficiantLastNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdLastName_mdLastName");
        protected readonly By OfficiantSuffixText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdNameSuffix_mdNameSuffix ");
        protected readonly By OfficiantTitleDropdown = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdTitle_mdTitle");
        protected readonly By OfficiantTitleOtherText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantName_mdTitleOtherSpecify_mdTitleO");
        protected readonly By OfficiantStreetNumberText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressStreetNumber_md AddressStreetNumber");
    }
}
protected readonly By OfficiantPreDirectionalDropdown = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressPreDirectional_mdAddressPreDirectional");

protected readonly By OfficiantStreetNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressStreetName_mdAddressStreetName");

protected readonly By OfficiantStreetDesignatorText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressStreetDesignator_mdAddressStreetDesignator");

protected readonly By OfficiantPostDirectionalText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressPostDirectional_mdAddressPostDirectional");

protected readonly By OfficiantApartmentText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressApartmentNumber_mdAddressApartmentNumber");

protected readonly By OfficiantCityText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressCity_mdAddressCity");

protected readonly By OfficiantCountyText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressCounty_mdAddressCounty");

protected readonly By OfficiantStateText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressState_mdAddressState");

protected readonly By OfficiantCountryText = By.Id("ctl00_ctl00_Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressCountry_mdAddressCountry");

protected readonly By OfficiantZipCodeText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlOfficiantAddress_mdAddressZipCode_mdAddressZipCode");

protected readonly By OfficiantPhoneNumberText = By.Id("Body_MPContent_MC1_dgOfficiant_mdPhoneNumber_mdPhoneNumber");

protected readonly By WitnessAFirstNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness1Name_mdFirstName_mdFirstName");

protected readonly By WitnessAMiddleNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness1Name_mdMiddleName_mdMiddleName");

protected readonly By WitnessALastNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness1Name_mdLastName_mdLastName");
protected readonly By WitnessASuffixText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness1Name_mdNameSuffix_mdNameSuffix ");
protected readonly By WitnessBFirstNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness2Name_mdFirstName_mdFirstName");
protected readonly By WitnessBMiddleNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness2Name_mdMiddleName_mdMiddleName ");
protected readonly By WitnessBLastNameText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness2Name_mdLastName_mdLastName");
protected readonly By WitnessBSuffixText = By.Id("Body_MPContent_MC1_dgOfficiant_ctlWitness2Name_mdNameSuffix_mdNameSuffix ");
protected readonly By DateOfCommissionText = By.Id("Body_MPContent_MC1_dgOfficiant ctlRegistrationCustom_repControls_mdDate1_0_mdDate1_0_txtDate_0");

public OfficiantPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(OfficiantPanel, 10);
public void Populate(string ceremonyType, string marriageCity, string marriageCounty, string marriageState, string officiantFirstName, string officiantMiddleName, string officiantLastName, string officiantSuffix, string officiantTitle, string officiantTitleOther, string streetNumber, string preDirectional, string streetName, string streetDesignator, string postDirectional, string apartment, string city, string county, string state, string country, string zip, string phoneNumber, string witnessAFirstName, string witnessAMiddleName, string witnessALastName, string witnessASuffix, string witnessBFirstName, string witnessBMiddleName, string witnessBLastName, string witnessBSuffix, string dateOfCommission)
{
    Select(CeremonyTypeDropdown, ceremonyType);
    Type(PlaceOfMarriageCityText, marriageCity);
    Type(PlaceOfMarriageCountyText, marriageCounty);
    Type(OfficiantFirstNameText, officiantFirstName);
    Type(OfficiantMiddleNameText, officiantMiddleName);
    Type(OfficiantLastNameText, officiantLastName);
    Type(OfficiantSuffixText, officiantSuffix);
Type(OfficiantTitleDropdown, officiantTitle);
EnabledType(OfficiantTitleOtherText, officiantTitleOther);
Type(OfficiantStreetNumberText, streetNumber);
Type(OfficiantPreDirectionalDropdown, preDirectional);
Type(OfficiantStreetNameText, streetName);
Type(OfficiantStreetDesignatorText, streetDesignator);
Type(OfficiantPostDirectionalText, postDirectional);
Type(OfficiantApartmentText, apartment);
Type(OfficiantCityText, city);
Type(OfficiantCountyText, country);
Type(OfficiantStateText, state);
Replace(OfficiantCountryText, country);
Type(OfficiantZipCodeText, zip);
Replace(OfficiantPhoneText, phoneNumber);
Type(WitnessAFirstNameText, witnessAFirstName);
Type(WitnessAMiddleNameText, witnessAMiddleName);
Type(WitnessALastNameText, witnessALastName);
Type(WitnessASuffixText, witnessASuffix);
Type(WitnessBFirstNameText, witnessBFirstName);
Type(WitnessBMiddleNameText, witnessBMiddleName);
Type(WitnessBLastNameText, witnessBLastName);
Type(WitnessBSuffixText, witnessBSuffix);
Type(DateOfCommissionText, dateOfCommission);
}
}
}

PartyAAttributesPage.cs

using System;
using OpenQA.Selenium;

namespace Framework
{
    class PartyAAttributesPage : BasePage
    {
        protected readonly By PartyAAttributesPanel =
            By.CssSelector("div[code='MARRIAGE_PARTY_A_ATTRIBUTES']");
protected readonly By PartyASSNText = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdSsn_mdSsn");
protected readonly By PartyASSNNoneRadio = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_radSsnNone");
protected readonly By PartyASSNUnknownRadio = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_radSsnUnknown");
protected readonly By PartyAMarriageNumberDropdown = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdNumThisMarriage");
protected readonly By PartyAMarriageEndedDropdown = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdPrevMarriage");
protected readonly By PartyAMarriageEndedText = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdDateLastMarriage_txtDate");
protected readonly By PartyAHispanicDropdown = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdHispanicOrigin");
protected readonly By PartyAHispanicOtherText = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdHispanicOther");
protected readonly By PartyARaceWhiteCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkBox_7794");
protected readonly By PartyARaceBlackCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkBox_7795");
protected readonly By PartyARaceAmIndian1Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox_7796_113");
protected readonly By PartyARaceAmIndian2Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox_7796_114");
protected readonly By PartyARaceAsIndianCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7797");
protected readonly By PartyARaceChineseCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7798");
protected readonly By PartyARaceFilipinoCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7799");
protected readonly By PartyARaceJapaneseCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7800");
protected readonly By PartyARaceKoreanCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7801");
protected readonly By PartyARaceVietnameseCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7802");
protected readonly By PartyARaceOtherAsianCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7803");
protected readonly By PartyARaceOtherAsian1Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7803_117");
protected readonly By PartyARaceOtherAsian2Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7803_121");
protected readonly By PartyARaceHawaiianCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7804");
protected readonly By PartyARaceGuamianCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7805");
protected readonly By PartyARaceSamoanCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7806");
protected readonly By PartyARaceOtherPacIslandCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkbox_7807");
protected readonly By PartyARaceOtherPacIsland1Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7807_118");
protected readonly By PartyARaceOtherPacIsland2Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7807_122");
protected readonly By PartyARaceOtherCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkBox_7794");
protected readonly By PartyARaceOther1Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7808_116");
protected readonly By PartyARaceOther2Text = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_textBox7808_120");
protected readonly By PartyARaceUnknownCheckbox = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_cblRace_cblRace_checkBox_7809");
protected readonly By PartyAEducationDropdown = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdEducation_mdEducation");
protected readonly By PartyAOccupationText = By.Id("Body_MPContent_ctlPartyAAttributes_ctlPartyAAttributes_dgPartyAttributes_mdOccupation_mdOccupation");

public PartyAAttributesPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyAAttributesPanel, 10);

public void Populate(string marriageNumber, string marriageEnded, string marriageEndedDate, string hispanicOrigin, string hispanicOther, string raceWhite, string raceBlack, string raceAmIndian, string raceAmIndianText1, string raceAmIndianText2, string raceAsIndian, string raceChinese, string raceFilipino, string raceJapanese, string raceKorean, string raceVietnamese, string raceOtherAsian, string raceOtherAsianText1, string raceOtherAsianText2, string raceHawaiian, string raceGuamanian, string raceSamoaan, string raceOtherPacIsland, string raceOtherPacIslandText1, string raceOtherPacIslandText2, string raceOther, string raceOtherText1,
string raceOtherText1, string raceOtherText2, string raceUnknown, string education, string occupation)
{
    Select(PartyAMarriageNumberDropdown, marriageNumber);
    Select(PartyAMarriageEndedDropdown, marriageEnded);
    Type(PartyAMarriageEndedText, marriageEndedDate);
    Select(PartyAHispanicDropdown, hispanicOrigin);
    EnabledType(PartyAHispanicOtherText, hispanicOther);
    Check(PartyARaceWhiteCheckbox, raceWhite);
    Check(PartyARaceBlackCheckbox, raceBlack);
    Check(PartyARaceAmIndianCheckbox, raceAmIndian);
    Check(PartyARaceAsIndianCheckbox, raceAsIndian);
    Check(PartyARaceChineseCheckbox, raceChinese);
    Check(PartyARaceFilipinoCheckbox, raceFilipino);
    Check(PartyARaceJapaneseCheckbox, raceJapanese);
    Check(PartyARaceKoreanCheckbox, raceKorean);
    Check(PartyARaceVietnameseCheckbox, raceVietnamese);
    Check(PartyARaceOtherAsianCheckbox, raceOtherAsian);
    Check(PartyARaceHawaiianCheckbox, raceHawaiian);
    Check(PartyARaceGuamianCheckbox, raceGuamian);
    Check(PartyARaceSamoanCheckbox, raceSamoan);
    Check(PartyARaceOtherPacIslandCheckbox, raceOtherPacIsland);
    Select(PartyAOccupationDropdown, education);
    Type(PartyAOccupationText, occupation);
    DisplayedType(PartyARaceAmIndian1Text, raceAmIndianText1);
    DisplayedType(PartyARaceAmIndian2Text, raceAmIndianText2);
    DisplayedType(PartyARaceOtherAsian1Text, raceOtherAsianText1);
    DisplayedType(PartyARaceOtherAsian2Text, raceOtherAsianText2);
    DisplayedType(PartyARaceOtherPacIsland1Text, raceOtherPacIslandText1);
    DisplayedType(PartyARaceOtherPacIsland2Text, raceOtherPacIslandText2);
    DisplayedType(PartyARaceOther1Text, raceOtherText1);
    DisplayedType(PartyARaceOther2Text, raceOtherText2);
}
}
}

PartyAOtherInformationPage.cs

using OpenQA.Selenium;
namespace Framework
{
    class PartyAOtherInformationPage : BasePage
    {
        protected readonly By PartyAOtherInformationPanel = By.CssSelector("div[code='MARRIAGE_PARTY_A_OTHER_INFORMATION']");
        protected readonly By PartyAStreetNumberText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressStreetNumber_mdAddressStreetNumber");
        protected readonly By PartyAPreDirectionalDropdown = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressPreDirectional_mdAddressPreDirectional");
        protected readonly By PartyAStreetNameText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressStreetName_mdAddressStreetName");
        protected readonly By PartyAStreetDesignatorText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressStreetDesignator_mdAddressStreetDesignator_Input");
        protected readonly By PartyAPostDirectionalText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressPostDirectional_mdAddressPostDirectional");
        protected readonly By PartyAApartmentText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressApartmentNumber_mdAddressApartmentNumber");
        protected readonly By PartyACityText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressCity_mdAddressCity_Input");
        protected readonly By PartyACountyText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressCounty_mdAddressCounty_Input");
        protected readonly By PartyAStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressState_mdAddressState_Input");
        protected readonly By PartyACountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressCountry_mdAddressCountry_Input");
        protected readonly By PartyAZipCodeText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlResidentAddress_mdAddressZipCode_mdAddressZipCode");
protected readonly By PartyAFirstNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlPartyLeg alNameAfterMarriage_mdFirstName_mdFirstName");
protected readonly By PartyAMiddleNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlPartyLeg alNameAfterMarriage_mdMiddleName_mdMiddleName");
protected readonly By PartyAOtherMiddleNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlPartyLeg alNameAfterMarriage_mdOtherMiddleName_mdOtherMiddleName");
protected readonly By PartyALastNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlPartyLeg alNameAfterMarriage_mdLastName_mdLastName");
protected readonly By PartyASuffixAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_ctlPartyLeg alNameAfterMarriage_mdNameSuffix_mdNameSuffix");
protected readonly By PartyAAffidavitOfAgeCheckbox = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_mdAffidavi tOfAge_mdAffidavitOfAge");
protected readonly By PartyADateSignedText = By.Id("Body_MPContent_PartyACtl_ctlPartyAOtherInformation_dgPartyOtherInfo_mdPartyDa teSigned_mdPartyDateSigned_txtDate");

public PartyAOtherInformationPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyAOtherInformationPanel, 10);

public void Populate(string streetNumber, string preDirectional, string streetName, string streetDesignator, string postDirectional, string apartment, string city, string county, string state, string country, string zip, string firstName, string middleName, string otherMiddleName, string lastName, string suffix, string affidavit, string dateSigned)
{
    Type(PartyAStreetNumberText, streetNumber);
    Type(PartyAPreDirectionalDropdown, preDirectional);
    Type(PartyAStreetNameText, streetName);
    Type(PartyAStreetDesignatorText, streetDesignator);
    Type(PartyAPostDirectionalText, postDirectional);
    Type(PartyAApartmentText, apartment);
    Type(PartyACityText, city);
    Type(PartyACountyText, county);
    Type(PartyAStateText, state);
    Replace(PartyACountryText, country);
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK

```csharp
using OpenQA.Selenium;

namespace Framework
{
    class PartyAPage : BasePage
    {
        protected readonly By PartyAPanel = By.CssSelector("div[code='MARRIAGE_PARTY_A']");
        protected readonly By PreferredTitleDropdown = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_mdPreferredTitle_mdPreferredTitle");
        protected readonly By FirstNameCurrentText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalName_mdFirstName_mdFirstName");
        protected readonly By MiddleNameCurrentText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalName_mdMiddleName_mdMiddleName");
        protected readonly By OtherMiddleNameCurrentText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalName_mdOtherMiddleName_mdOtherMiddleName");
        protected readonly By LastNameCurrentText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalName_mdLastName_mdLastName");
        protected readonly By SuffixCurrentText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalName_mdNameSuffix_mdNameSuffix");
    }
}
```

PartyAPage.cs
protected readonly By CopyCurrentNameButton = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_btnCopyLegalName");
protected readonly By FirstNamePriorText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalNameAtBirth_mdFirstName");
protected readonly By MiddleNamePriorText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalNameAtBirth_mdMiddleName");
protected readonly By OtherMiddleNamePriorText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalNameAtBirth_mdOtherMiddleName");
protected readonly By LastNamePriorText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalNameAtBirth_mdLastName");
protected readonly By SuffixPriorText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyLegalNameAtBirth_mdNameSuffix");
protected readonly By BirthplaceStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyBirthPlace_mdAddressState_mdAddressState/Input");
protected readonly By BirthplaceCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyA_dgParty_ctlPartyBirthPlace_mdAddressCountry_mdAddressCountry/Input");
protected readonly By DateOfBirthText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_mdDateOfBirth_mdDateOfBirth/txtdate");
protected readonly By AgeText = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_mdAge_mdAge");
protected readonly By GenderDropdown = By.Id("Body_MPContent_PartyACtl_ctlPartyA_dgParty_mdGender_mdGender");

public PartyAPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyAPanel, 10);
public void Populate(string preferredTitle, string firstNameCurrent, string middleNameCurrent, string otherMiddleNameCurrent, string lastNameCurrent, string suffixCurrent, string firstNamePrior,
string middleNamePrior,
string otherMiddleNamePrior,
string lastNamePrior,
string suffixPrior,
string birthPlaceState,
string birthPlaceCountry,
string dateOfBirth,
string age,
string sex)
{
    Select(PreferredTitleDropdown, preferredTitle);
    Type(MiddleNameCurrentText, middleNameCurrent);
    Type(OtherMiddleNameCurrentText, otherMiddleNameCurrent);
    Type(SuffixCurrentText, suffixCurrent);
    Type(FirstNamePriorText, firstNamePrior);
    Type(MiddleNamePriorText, middleNamePrior);
    Type(OtherMiddleNamePriorText, otherMiddleNamePrior);
    Type(FirstNamePriorText, lastNamePrior);
    Type(SuffixPriorText, suffixPrior);
    Type(BirthplaceStateText, birthPlaceState);
    Replace(BirthplaceCountryText, birthPlaceCountry);
    Type(DateOfBirthText, dateOfBirth);
    Type(AgeText, age);
    Select(GenderDropdown, sex);
}
}

PartyAParentsPage.cs

using OpenQA.Selenium;

namespace Framework
{
    class PartyAParentsPage : BasePage
    {
        protected readonly By PartyAParentsPanel =
            By.CssSelector("div[code='MARRIAGE_PARTY_A_PARENTS']");
protected readonly By PartyAFatherFirstNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherName_mdFirstName_mdFirstName");
protected readonly By PartyAFatherMiddleNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherName_mdMiddleName_mdMiddleName");
protected readonly By PartyAFatherLastNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherName_mdLastName_mdLastName");
protected readonly By PartyAFatherSuffixText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherName_mdNameSuffix_mdNameSuffix");
protected readonly By PartyAFatherBirthStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherBirthPlace_mdAddressState_mdAddressState_Input");
protected readonly By PartyAFatherBirthCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyFatherBirthPlace_mdAddressCountry_mdAddressCountry_Input");
protected readonly By PartyAMotherFirstNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherNameAtBirth_mdFirstName_mdFirstName");
protected readonly By PartyAMotherMiddleNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherNameAtBirth_mdMiddleName_mdMiddleName");
protected readonly By PartyAMotherLastNameText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherNameAtBirth_mdLastName_mdLastName");
protected readonly By PartyAMotherSuffixText = By.Id("Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherNameAtBirth_mdNameSuffix_mdNameSuffix");
protected readonly By PartyAMotherBirthStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherBirthPlace_mdAddressState_mdAddressState_Input");
protected readonly By PartyAMotherBithCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyAParentsCtl_ctlPartyAParents_dgPartyParent_ctlPartyMotherBirthPlace_mdAddressCountry_mdAddressCountry_Input");

public PartyAParentsPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyAParentsPanel, 10);
public void Populate(string fatherFirstName, string fatherMiddleName, string fatherLastName,
                      string fatherSuffix, string fatherBirthState, string fatherBirthCountry,
                      string motherFirstName, string motherMiddleName, string motherLastName,
                      string motherSuffix, string motherBirthState, string motherBirthCountry)
{
    Type(PartyAFatherFirstNameText, fatherFirstName);
    Type(PartyAFatherMiddleNameText, fatherMiddleName);
    Type(PartyAFatherLastNameText, fatherLastName);
    Type(PartyAFatherSuffixText, fatherSuffix);
    Type(PartyAFatherBirthStateText, fatherBirthState);
    Replace(PartyAFatherBirthCountryText, fatherBirthCountry);
    Type(PartyAMotherFirstNameText, motherFirstName);
    Type(PartyAMotherMiddleNameText, motherMiddleName);
    Type(PartyAMotherLastNameText, motherLastName);
    Type(PartyAMotherSuffixText, motherSuffix);
    Type(PartyAMotherBirthStateText, motherBirthState);
    Replace(PartyAMotherBirthCountryText, motherBirthCountry);
}

PartyBAttributesPage.cs

using System;
using OpenQA.Selenium;

namespace Framework
{
    class PartyBAttributesPage : BasePage
    {
        protected readonly By PartyBAttributesPanel =
By.CssSelector("div[code='MARRIAGE_PARTY_B_ATTRIBUTES']");
        protected readonly By PartyBSSNText =
By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdSsn_mdSsn");
        protected readonly By PartyBSSNNoneRadio =
By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_radSsnNone");
protected readonly By PartyBSSNUnknownRadio = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_radSsnUnknown");

protected readonly By PartyBMARRiAgeNumberDropdown = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdNumThisMarriage_mdNumThisMarriage");

protected readonly By PartyBMARRiAgeEndedDropdown = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdPrevMarriage_mdPrevMarriage");

protected readonly By PartyBMARRiAgeEndedText = By.Id("Body_MPContent_ctlPartyBAttributesCtlPartyBAttributes_dgPartyAttributesB_mdDateLastMarriage_mdDateLastMarriage_txtDate");

protected readonly By PartyBHispanicDropdown = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdHispanicOrigin_mdHispanicOrigin");

protected readonly By PartyBHispanicOtherText = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdHispanicOther_mdHispanicOther");

protected readonly By PartyBRaceWhiteCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7794");

protected readonly By PartyBRaceBlackCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7794_113");

protected readonly By PartyBRaceAmIndian1Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7796");

protected readonly By PartyBRaceAmIndian2Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7796_114");

protected readonly By PartyBRaceAsIndianCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7797");

protected readonly By PartyBRaceChineseCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7798");
protected readonly By PartyBRaceFilipinoCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7799");
protected readonly By PartyBRaceJapaneseCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7800");
protected readonly By PartyBRaceKoreanCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7801");
protected readonly By PartyBRaceVietnameseCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7802");
protected readonly By PartyBRaceOtherAsianCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7803");
protected readonly By PartyBRaceOtherAsian1Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7803_117");
protected readonly By PartyBRaceOtherAsian2Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7803_121");
protected readonly By PartyBRaceHawaiianCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7804");
protected readonly By PartyBRaceGuamianCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7805");
protected readonly By PartyBRaceSamoaCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7806");
protected readonly By PartyBRaceOtherPacIslandCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkBox_7807");
protected readonly By PartyBRaceOtherPacIsland1Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7807_118");
protected readonly By PartyBRaceOtherPacIsland2Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7807_122");
protected readonly By PartyBRaceOtherCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkboxBox_7794");
protected readonly By PartyBRaceOther1Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7808_116");
protected readonly By PartyBRaceOther2Text = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_textBox7808_120");
protected readonly By PartyBRaceUnknownCheckbox = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_cblRace_cblRace_checkboxBox_7809");
protected readonly By PartyBEducationDropdown = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdEducation_mdEducation");
protected readonly By PartyBOccupationText = By.Id("Body_MPContent_ctlPartyBAttributes_ctlPartyBAttributes_dgPartyAttributesB_mdOccupation_mdOccupation");

public PartyBAttributesPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyBAttributesPanel, 10);

public void Populate(string marriageNumber, string marriageEnded, string marriageEndedDate, string hispanicOrigin, string hispanicOther, string raceWhite, string raceBlack, string raceAmIndian, string raceAmIndianText1, string raceAmIndianText2, string raceAsIndian, string raceChinese, string raceFilipino, string raceJapanese, string raceKorean, string raceVietnamese, string raceOtherAsian, string raceOtherAsianText1, string raceOtherAsianText2, string raceHawaiian, string raceGuamian, string raceSamoan, string raceOtherPacIsland, string raceOtherPacIslandText1, string raceOtherPacIslandText2, string raceOther, string raceOtherText1, string raceOtherText2, string raceUnknown, string education, string occupation) {
    Select(PartyBMarriageNumberDropdown, marriageNumber);
    Select(PartyBMarriageEndedDropdown, marriageEnded);
    Type(PartyBMarriageEndedText, marriageEndedDate);
    Select(PartyBHispanicDropdown, hispanicOrigin);
using OpenQA.Selenium;
	namespace Framework
{

class PartyBOtherInformationPage : BasePage
{
    // Code for PartyBOtherInformationPage.cs
}
protected readonly By PartyBOtherInformationPanel = By.CssSelector("div[code='MARRIAGE_PARTY_B_OTHER_INFORMATION']");

protected readonly By PartyBStreetNumberText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressStreetNumber_mdAddressStreetNumber");

protected readonly By PartyBPreDirectionalDropdown = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressPreDirectional_mdAddressPreDirectional");

protected readonly By PartyBStreetNameText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressStreetName_mdAddressStreetName");

protected readonly By PartyBStreetDesignatorText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressStreetDesignator_mdAddressStreetDesignator_Input");

protected readonly By PartyBPostDirectionalText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressPostDirectional_mdAddressPostDirectional");

protected readonly By PartyBApartmentText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressApartmentNumber_mdAddressApartmentNumber");

protected readonly By PartyBCityText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressCity_mdAddressCity_Input");

protected readonly By PartyBCountyText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressCounty_mdAddressCounty_Input");

protected readonly By PartyBStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressState_mdAddressState_Input");

protected readonly By PartyBCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressCountry_mdAddressCountry_Input");

protected readonly By PartyBZipCodeText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlResidentAddress_mdAddressZipCode_mdAddressZipCode");

protected readonly By PartyBFirstNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation_dgPartyBOtherInfo_ctlPartyLegalNameAfterMarriage_mdFirstName_mdFirstName");
protected readonly By PartyBMiddleNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_ctlPartyLegalNameAfterMarriage_mdMiddleName_mdMiddleName");
protected readonly By PartyBOtherMiddleNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_ctlPartyLegalNameAfterMarriage_mdOtherMiddleName_mdOtherMiddleName");
protected readonly By PartyBLastNameAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_ctlPartyLegalNameAfterMarriage_mdLastName_mdLastName");
protected readonly By PartyBSuffixAfterMarriageText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_ctlPartyLegalNameAfterMarriage_mdNameSuffix_mdNameSuffix");
protected readonly By PartyBAffidavitOfAgeCheckbox = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_mdAffidavitOfAge_mdAffidavitOfAge");
protected readonly By PartyBDateSignedText = By.Id("Body_MPContent_PartyACtl_ctlPartyBOtherInformation dgPartyBOtherInfo_mdPartyDateSigned_txtDate");

public PartyBOtherInformationPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyBOtherInformationPanel, 10);

public void Populate(string streetNumber, string preDirectional, string streetName, string streetDesignator, string postDirectional, string apartment, string city, string county, string state, string country, string zip, string firstName, string middleName, string otherMiddleName, string lastName, string suffix, string affidavit, string dateSigned)
{
Type(PartyBStreetNumberText, streetNumber);
Type(PartyBPreDirectionalDropdown, preDirectional);
Type(PartyBStreetNameText, streetName);
Type(PartyBStreetDesignatorText, streetDesignator);
Type(PartyBPostDirectionalText, postDirectional);
Type(PartyBApartmentText, apartment);
Type(PartyBCityText, city);
Type(PartyBCountyText, county);
Type(PartyBStateText, state);
Replace(PartyBCountryText, country);
Type(PartyBZipCodeText, zip);
Type(PartyBFirstNameAfterMarriageText, firstName);
using OpenQA.Selenium;

namespace Framework
{
    class PartyBPage : BasePage
    {
        protected readonly By PartyBPanel = By.CssSelector("div[code='MARRIAGE_PARTY_B']");
        protected readonly By PreferredTitleDropdown = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_mdPreferredTitle_mdPreferredTitle");
        protected readonly By FirstNameCurrentText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalName_mdFirstName_mdFirstName");
        protected readonly By MiddleNameCurrentText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalName_mdMiddleName_mdMiddleName");
        protected readonly By OtherMiddleNameCurrentText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalName_mdOtherMiddleName_mdOtherMiddleName");
        protected readonly By LastNameCurrentText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalName_mdLastName_mdLastName");
        protected readonly By SuffixCurrentText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalName_mdNameSuffix_mdNameSuffix");
        protected readonly By CopyCurrentNameButton = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_btnCopyLegalName");
    }
}
protected readonly By FirstNamePriorText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalNameAtBirth_mdFirstName_mdFirstName"]);
protected readonly By MiddleNamePriorText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalNameAtBirth_mdMiddleName_mdMiddleName"]);
protected readonly By OtherMiddleNamePriorText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalNameAtBirth_mdOtherMiddleName_mdOtherMiddleName"]);
protected readonly By LastNamePriorText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalNameAtBirth_mdLastName_mdLastName"]);
protected readonly By SuffixPriorText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyLegalNameAtBirth_mdNameSuffix_mdNameSuffix"]);
protected readonly By BirthplaceStateText = By.Id("ctl00_ctl00_Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyBirthPlace_mdAddressState_mdAddressState_Input"]);
protected readonly By BirthplaceCountryText = By.Id("ctl00_ctl00_Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_ctlPartyBirthPlace_mdAddressCountry_mdAddressCountry_Input"]);
protected readonly By DateOfBirthText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_mdDateOfBirth_mdDateOfBirth_txtDate"]);
protected readonly By AgeText = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_mdAge_mdAge"]);
protected readonly By GenderDropdown = By.Id("Body_MPContent_partnerBCtl_ctlPartyB_dgPartyB_mdGender_mdGender"]);

public PartyBPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyBPanel, 10);
public void Populate(string preferredTitle, string firstNameCurrent, string middleNameCurrent, string otherMiddleNameCurrent, string lastNameCurrent, string suffixCurrent, string firstNamePrior, string middleNamePrior, string otherMiddleNamePrior, string lastNamePrior, string suffixPrior, string birthPlaceState, string birthPlaceCountry, string dateOfBirth, string age, string sex)
{
    Select(PreferredTitleDropdown, preferredTitle);
Type(MiddleNameCurrentText, middleNameCurrent);
Type(OtherMiddleNameCurrentText, otherMiddleNameCurrent);
Type(SuffixCurrentText, suffixCurrent);
Type(FirstNamePriorText, firstNamePrior);
Type(MiddleNamePriorText, middleNamePrior);
Type(OtherMiddleNamePriorText, otherMiddleNamePrior);
Type(LastNamePriorText, lastNamePrior);
Type(SuffixPriorText, suffixPrior);
Type(BirthplaceStateText, birthPlaceState);
Replace(BirthplaceCountryText, birthPlaceCountry);
Type(DateOfBirthText, dateOfBirth);
Type(AgeText, age);
Select(GenderDropdown, sex);
}
}
}

PartyBParentsPage.cs

using OpenQA.Selenium;

namespace Framework
{

class PartyBParentsPage : BasePage
{
    protected readonly By PartyBParentsPanel =
By.CssSelector("div[code='MARRIAGE_PARTY_B_PARENTS']");
    protected readonly By PartyBFatherFirstNameText =
By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents_dgPartyParentB_ctlPartyFatherName_mdFirstName_mdFirstName");
    protected readonly By PartyBFatherMiddleNameText =
By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents_dgPartyParentB_ctlPartyFatherName_mdMiddleName_mdMiddleName");
    protected readonly By PartyBFatherLastNameText =
By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents_dgPartyParentB_ctlPartyFatherName_mdLastName_mdLastName");
    protected readonly By PartyBFatherSuffixText =
By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents_dgPartyParentB_ctlPartyFatherName_mdNameSuffix_mdNameSuffix");
protected readonly By PartyBFatherBirthStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyFatherBirthPlace_mdAddressState_mdAddressState_Input");
protected readonly By PartyBFatherBirthCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyFatherBirthPlace_mdAddressCountry_mdAddressCountry_Input");
protected readonly By PartyBMotherFirstNameText = By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherNameAtBirth_mdFirstName_mdFirstName");
protected readonly By PartyBMotherMiddleNameText = By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherNameAtBirth_mdMiddleName_mdMiddleName");
protected readonly By PartyBMotherLastNameText = By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherNameAtBirth_mdLastName_mdLastName");
protected readonly By PartyBMotherSuffixText = By.Id("Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherNameAtBirth_mdNameSuffix_mdNameSuffix");
protected readonly By PartyBMotherBirthStateText = By.Id("ctl00_ctl00_Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherBirthPlace_mdAddressState_mdAddressState_Input");
protected readonly By PartyBMotherBirthCountryText = By.Id("ctl00_ctl00_Body_MPContent_PartyBParentsCtl_ctlPartyBParents dgPartyParentB_ctlPartyMotherBirthPlace_mdAddressCountry_mdAddressCountry_Input");

public PartyBParentsPage(IWebDriver driver) : base(driver) { }
public bool PageDisplays() => IsDisplayed(PartyBParentsPanel, 10);

public void Populate(string fatherFirstName, string fatherMiddleName, string fatherLastName,
string fatherSuffix, string fatherBirthState, string fatherBirthCountry,
string motherFirstName, string motherMiddleName, string motherLastName,
string motherSuffix, string motherBirthState, string motherBirthCountry)
{
    Type(PartyBFatherFirstNameText, fatherFirstName);
    Type(PartyBFatherMiddleNameText, fatherMiddleName);
    Type(PartyBFatherLastNameText, fatherLastName);
    Type(PartyBFatherSuffixText, fatherSuffix);
    Type(PartyBFatherBirthStateText, fatherBirthState);
    Replace(PartyBFatherBirthCountryText, fatherBirthCountry);
##############
MarriageFastOrderPage.cs

using Atlassian.Jira;
using OpenQA.Selenium;

namespace Framework
{
    class MarriageFastOrderPage : BasePage
    {
        protected readonly By PartyAPanel = By.Id("Body_MPContent_MC1_OrderHeader1_contents_div");
        protected readonly By SourceDropdown = By.Id("Body_MPContent_MC1_selSource");
        protected readonly By PriorityDropdown = By.Id("Body_MPContent_MC1_selPriority");
        protected readonly By DeliveryDropdown = By.Id("Body_MPContent_MC1_selDelivery");
        protected readonly By MarriageLicenseCheckbox = By.Id("Body_MPContent_MC1_rpServices_chkBox_2");
        protected readonly By QuantityText = By.Id("Body_MPContent_MC1_rpServices_txtQuantity_2");
        protected readonly By RequestReasonDropdown = By.Id("Body_MPContent_MC1_rpServices_selRequestReason_2");
        protected readonly By OtherReasonText = By.Id("Body_MPContent_MC1_rpServices_txtReasonOther_2");
        protected readonly By CalculateFeesButton = By.Id("Body_MPContent_MC1_btnCalcFees");
        protected readonly By CashCheckbox = By.Id("Body_MPContent_MC1_chkCash");
    }
}
protected readonly By CashText = By.Id("Body_MPContent_MC1_txtCashAmount");
protected readonly By SaveandValidateButton = By.Id("Body_MPContent_MC1_btnSave");
protected readonly By IssueLink = By.Id("Body_MPContent_MC1_dgService_lnkIssue_0");
protected readonly By IssuanceFrame = By.Id("RadPopupWindow");
protected readonly By PrintIssuanceLink = By.Id("Body_MPContent_ctlIssuance_ctlIssuance_btnPrint");
protected readonly By SecurityPaperNumberText = By.Id("Body_MPContent_ctlIssuance_ctlIssuance_dgIssuanceList_txtSecurityPaperNumber_0");
protected readonly By SaveIssuanceButton = By.Id("Body_MPContent_ctlIssuance_ctlIssuance_btnSave");
protected readonly By CompleteLink = By.Id("Body_MPContent_ctlIssuance_ctlIssuance_btnComplete");

public MarriageFastOrderPage(IWebDriver driver) : base(driver) { }

public bool PageDisplays() => IsDisplayed(PartyAPanel, 10);

public void Populate(string source,
    string priority,
    string delivery,
    string marriageLicense,
    string quantity,
    string requestReason,
    string requestReasonOther,
    string paymentType,
    string paymentAmount)
{
    Select(SourceDropdown, source);
    Select(PriorityDropdown, priority);
    Select(DeliveryDropdown, delivery);
    Check(MarriageLicenseCheckbox, marriageLicense);
    Type(QuantityText, quantity);
    Select(RequestReasonDropdown, requestReason);
    Click(CalculateFeesButton);
    if (!IsDisplayed(WaitAMomentMessage, 10))
        Click(CashCheckbox);
    Click(SaveandValidateButton);
if (!IsDisplayed(WaitAMomentMessage, 10))
    IssueNonCertified();
if (!IsDisplayed(WaitAMomentMessage, 10))
    Click(ReturnButton);
}

private void IssueNonCertified()
{
    Click(IssueLink);
    SwitchToFrame("RadPopupWindow");
    SelectAllCheckboxes();
    Click(PrintIssuanceLink);
    if (!IsDisplayed(WaitAMomentMessage, 10))
        SelectAllCheckboxes();
    Click(CompleteLink);
    SwitchToDefaultFrame();
}
Appendix C
Workflow Classes for BIE 2

LoginWorkflow.cs

using NUnit.Framework;
using OpenQA.Selenium;

namespace Framework
{
    [TestFixture]
    class LoginWorkflow : BasePage
    {
        private LoginPage Login;
        private HomePage Home;

        public LoginWorkflow(IWebDriver driver) : base(driver) { }

        public void InitialLogin(string target, string userName, string password)
        {
            Login = new LoginPage(Driver);
            Home = new HomePage(Driver);

            Login.GoTo(target);
            Assert.That(Driver.Title, Is.EqualTo("Logon"));
            Login.With(userName, password);
            Assert.That(Home.PageDisplays);
            Assert.That(Driver.Title, Is.EqualTo("Home"));
        }
    }
}

MarriageWorkflow.cs

using NUnit.Framework;
using OpenQA.Selenium;

namespace Framework
{
    [TestFixture]
}
class MarriageWorkflow : BasePage {
    private TopBar TopBar;
    private MarriageStartNewCasePage StartNewCase;
    private MarriageSearchPage MarriageSearch;
    private MarriageSearchResultsPage SearchResults;
    private PartyAPage PartyA;
    private PartyAParentsPage PartyAParents;
    private PartyAOtherInformationPage PartyAOtherInformation;
    private PartyAAttributesPage PartyAAttributes;
    private PartyBPage PartyB;
    private PartyBParentsPage PartyBParents;
    private PartyBOtherInformationPage PartyBOtherInformation;
    private PartyBAttributesPage PartyBAttributes;
    private LocalOfficialPage LocalOfficial;
    private MarriageRegistrationMenu RegistrationMenu;
    private OfficiantPage Officiant;
    private FilingOfficialPage FilingOfficial;

    public MarriageWorkflow(IWebDriver driver) : base(driver) { }

    public void StartNew(string partyAFirstNameCurrent, string partyALastNameCurrent, string partyBFirstNameCurrent, string partyBLastNameCurrent, string dateOfMarriage) {
        TopBar = new TopBar(Driver);
        StartNewCase = new MarriageStartNewCasePage(Driver);
        MarriageSearch = new MarriageSearchPage(Driver);
        SearchResults = new MarriageSearchResultsPage(Driver);
        TopBar.SelectMarriageStartNewCase();
        Assert.That(StartNewCase.PageDisplays);
        Assert.That(Driver.Title, Is.EqualTo("Case Search"));
        StartNewCase.Populate(partyAFirstNameCurrent, partyALastNameCurrent, partyBFirstNameCurrent, partyBLastNameCurrent, dateOfMarriage);
        Assert.That(SearchResults.PageDisplays);
        Assert.That(Driver.Title, Is.EqualTo("Case Search"));
        SearchResults.StartNewCase();
    }
public void Search(string dateOfMarriage)
{
    TopBar = new TopBar(Driver);
    MarriageSearch = new MarriageSearchPage(Driver);
    SearchResults = new MarriageSearchResultsPage(Driver);
    PartyA = new PartyAPage(Driver);
    TopBar.SelectMarriageSearch();

    Assert.That(MarriageSearch.PageDisplays());
    Assert.That(Driver.Title, Is.EqualTo("Search Case"));

    MarriageSearch.ByDateOfMarriage(dateOfMarriage);
    SearchResults.SelectFirstRecord();
    Assert.That(PartyA.PageDisplays);
    Assert.That(Driver.Title, Is.EqualTo("Party A"));
}

public void EnterLicenseData(string partyATitle, string partyAFirstNameCurrent, string partyAMiddleNameCurrent, string partyAOtherMiddleNameCurrent, string partyALastNameCurrent, string partyASuffixCurrent, string partyAFirstNamePrior, string partyAMiddleNamePrior, string partyAOtherMiddleNamePrior, string partyALastNamePrior, string partyASuffixPrior, string partyABirthplaceState, string partyABirthplaceCountry, string partyADateOfBirth, string partyAAge, string partyAGender, string partyAFatherFirstName, string partyAFatherMiddleName, string partyAFatherLastName, string partyAFatherSuffix, string partyAFatherBirthState, string partyAFatherBirthCountry, string partyAMotherFirstName, string partyAMotherMiddleName, string partyAMotherLastName, string partyAMotherSuffix, string partyAMotherBirthState, string partyAMotherBirthCountry, string partyAStreetNumber, string partyAPreDirectional, string partyAStreetName, string partyAStreetDesignator, string partyAPostDirectional, string partyAApartment, string partyACity, string partyACounty, string partyAState, string partyACountry, string partyAZip, string partyAFirstNameAfterMarriage, string partyAMiddleNameAfterMarriage, string partyAOtherMiddleNameAfterMarriage, string partyALastNameAfterMarriage, string partyASuffixAfterMarriage,
string partyAAffidavit, string partyADateSigned,
string partyAMarriageNumber, string partyAMarriageEnded,
string partyAMarriageEndedDate, string partyAHispanicOrigin,
string partyAHispanicOther, string partyARaceWhite,
string partyARaceBlack, string partyARaceAmIndian,
string partyARaceAmIndianText1, string partyARaceAmIndianText2,
string partyARaceAsIndian, string partyARaceChinese,
string partyARaceFilipino, string partyARaceJapanese,
string partyARaceKorean, string partyARaceVietnamese,
string partyARaceOtherAsian, string partyARaceOtherAsianText1,
string partyARaceOtherAsianText2, string partyARaceHawaiian,
string partyARaceGuamian, string partyARaceSamoan,
string partyARaceOtherPacIsland, string partyARaceOtherPacIslandText1,
string partyARaceOtherPacIslandText2, string partyARaceOther,
string partyARaceOtherText1, string partyARaceOtherText2,
string partyARaceUnknown, string partyAEducation, string
partyAOccupation,
string partyBTitle, string partyBFirstNameCurrent,
string partyBMiddleNameCurrent, string partyBOtherMiddleNameCurrent,
string partyBLastNameCurrent, string partyBSuffixCurrent,
string partyBFirstNamePrior, string partyBMiddleNamePrior,
string partyBOtherMiddleNamePrior, string partyBLastNamePrior,
string partyBSuffixPrior, string partyBBirthplaceState,
string partyBBirthplaceCountry, string partyBDateOfBirth,
string partyBAge, string partyBGender,
string partyBFatherFirstName, string partyBFatherMiddleName,
string partyBFatherLastName, string partyBFatherSuffix,
string partyBFatherBirthState, string partyBFatherBirthCountry,
string partyBMotherFirstName, string partyBMotherMiddleName,
string partyBMotherLastName, string partyBMotherSuffix,
string partyBMotherBirthState, string partyBMotherBirthCountry,
string partyBStreetNumber, string partyBPreDirectional,
string partyBStreetName, string partyBStreetDesignator,
string partyBPostDirectional, string partyBApartment,
string partyBCity, string partyBCounty,
string partyBState, string partyBCountry,
string partyBZip, string partyBFirstNameAfterMarriage,
string partyBMiddleNameAfterMarriage, string
partyBOtherMiddleNameAfterMarriage,
string partyBLastNameAfterMarriage, string partyBSuffixAfterMarriage,
string partyBAffidavit, string partyBDateSigned,
string partyBMarriageNumber, string partyBMarriageEnded,
string partyBMarriageEndedDate, string partyBHispanicOrigin,
string partyBHispanicOther, string partyBRaceWhite,
string partyBRaceBlack, string partyBRaceAmIndian,
string partyBRaceAmIndianText1, string partyBRaceAmIndianText2,
string partyBRaceAsIndian, string partyBRaceChinese,
string partyBRaceFilipino, string partyBRaceJapanese,
string partyBRaceKorean, string partyBRaceVietnamese,
string partyBRaceOtherAsian, string partyBRaceOtherAsianText1,
string partyBRaceOtherAsianText2, string partyBRaceHawaiian,
string partyBRaceGuamanian, string partyBRaceSamoan,
string partyBRaceOtherPacIsland, string partyBRaceOtherPacIslandText1,
string partyBRaceOtherPacIslandText2, string partyBRaceOther,
string partyBRaceOtherText1, string partyBRaceOtherText2,
string partyBRaceUnknown, string partyBEducation, string
partyBOccupation,
string dateLicensedIssued, string waiverSubmitted,
string dateLicenseEffective, string dateLicenseExpiration,
string dateIntentionsFiled, string countyOfIssuance,
string issuingOfficial, string issuingOfficialFirstName,
string issuingOfficialMiddleName, string issuingOfficialLastName,
string issuingOfficialSuffix, string issuingOfficialTitle,
string issuingOfficialTitleOther)
{
    PartyA = new PartyAPage(Driver);
    PartyAParents = new PartyAParentsPage(Driver);
    PartyAOtherInformation = new PartyAOtherInformationPage(Driver);
    PartyAAttributes = new PartyAAttributesPage(Driver);
    PartyB = new PartyBPage(Driver);
    PartyBParents = new PartyBParentsPage(Driver);
    PartyBOtherInformation = new PartyBOtherInformationPage(Driver);
    PartyBAttributes = new PartyBAttributesPage(Driver);
    LocalOfficial = new LocalOfficialPage(Driver);

    Assert.That(PartyA.PageDisplays);
    Assert.That(Driver.Title, Is.EqualTo("Party A"));

    PartyA.Populate(partyATitle, partyAFirstNameCurrent,
    partyAMiddleNameCurrent, partyAOtherMiddleNameCurrent,
partyALastNameCurrent, partyASuffixCurrent, partyAFirstNamePrior,
partyAMiddleNamePrior,
partyAOtherMiddleNamePrior, partyALastNamePrior, partyASuffixPrior,
partyABirthplaceState,
partyABirthplaceCountry, partyADateOfBirth, partyAAge, partyAGender);

PartyA.Next();

Assert.That(PartyAParents.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party A Parents"));

PartyAParents.Populate(partyAFatherFirstName, partyAFatherMiddleName,
partyAFatherLastName,
partyAFatherSuffix, partyAFatherBirthState, partyAFatherBirthCountry,
partyAMotherFirstName, 
partyAMotherMiddleName, partyAMotherLastName, partyAMotherSuffix, 
partyAMotherBirthState, partyAMotherBirthCountry);

PartyAParents.Next();

Assert.That(PartyAOtherInformation.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party A Other Information"));

PartyAOtherInformation.Populate(partyAStreetNumber, partyAPreDirectional,
partyAStreetName,
partyAStreetDesignator, partyAPostDirectional, partyAApartment,
partyACity, partyACounty, 
partyAState, partyACountry, partyAZip, partyAFirstNameAfterMarriage, 
partyAMiddleNameAfterMarriage, 
partyAOtherMiddleNameAfterMarriage, partyALastNameAfterMarriage, 
partyASuffixAfterMarriage, partyAAffidavit, partyADateSigned);
PartyAOtherInformation.Next();

Assert.That(PartyAAttributes.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party A Attributes"));

PartyAAttributes.Populate(partyAMarriageNumber, partyAMarriageEnded, 
partyAMarriageEndedDate, partyAHispanicOrigin,
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK

partyAHispanicOther, partyARaceWhite,
partyARaceBlack, partyARaceAmIndian,
partyARaceAmIndianText1, partyARaceAmIndianText2,
partyARaceAsIndian, partyARaceChinese,
partyARaceFilipino, partyARaceJapanese,
partyARaceKorean, partyARaceVietnamese,
partyARaceOtherAsian, partyARaceOtherAsianText1,
partyARaceOtherAsianText2, partyARaceHawaiian,
partyARaceGuamian, partyARaceSamoan,
partyARaceOtherPacIsland, partyARaceOtherPacIslandText1,
partyARaceOtherPacIslandText2, partyARaceOther,
partyARaceOtherText1, partyARaceOtherText2,
partyARaceUnknown, partyAEducation, partyAOccupation);

PartyAAttributes.Next();

Assert.That(PartyB.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party B"));

PartyB.Populate(partyBTitle, partyBFirstNameCurrent,
partyBMiddleNameCurrent, partyBOtherMiddleNameCurrent,
partyBLastNameCurrent, partyBSuffixCurrent,
partyBFirstNamePrior, partyBMiddleNamePrior,
partyBOtherMiddleNamePrior, partyBLastNamePrior,
partyBSuffixPrior, partyBBirthplaceState,
partyBBirthplaceCountry, partyBDateOfBirth,
partyBAge, partyBGender);

PartyB.Next();

Assert.That(PartyBParents.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party B Parents"));

PartyBParents.Populate(partyBFatherFirstName, partyBFatherMiddleName,
partyBFatherLastName, partyBFatherSuffix,
partyBFatherBirthState, partyBFatherBirthCountry,
partyBMotherFirstName, partyBMotherMiddleName,
partyBMotherLastName, partyBMotherSuffix,
partyBMotherBirthState, partyBMotherBirthCountry);
PartyBParents.Next();

Assert.That(PartyBOtherInformation.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party B Other Information"));

PartyBOtherInformation.Populate(partyBStreetNumber, partyBPreDirectional, 
partyBStreetName, partyBStreetDesignator, 
partyBPostDirectional, partyBApartment, 
partyBCity, partyBCounty, 
partyBState, partyBCountry, 
partyBZip, partyBFirstNameAfterMarriage, 
partyBMiddleNameAfterMarriage, partyBOtherMiddleNameAfterMarriage, 
partyBLastNameAfterMarriage, partyBSuffixAfterMarriage, 
partyBAffidavit, partyBDateSigned);

PartyBOtherInformation.Next();

Assert.That(PartyBAttributes.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Party B Attributes"));

PartyBAttributes.Populate(partyBMarriageNumber, partyBMarriageEnded, 
partyBMarriageEndedDate, partyBHispanicOrigin, 
partyBHispanicOther, partyBRaceWhite, 
partyBRaceBlack, partyBRaceAmIndian, 
partyBRaceAmIndianText1, partyBRaceAmIndianText2, 
partyBRaceAsIndian, partyBRaceChinese, 
partyBRaceFilipino, partyBRaceJapanese, 
partyBRaceKorean, partyBRaceVietnamese, 
partyBRaceOtherAsian, partyBRaceOtherAsianText1, 
partyBRaceOtherAsianText2, partyBRaceHawaiian, 
partyBRaceGuamian, partyBRaceSamoa, 
partyBRaceOtherPacIsland, partyBRaceOtherPacIslandText1, 
partyBRaceOtherPacIslandText2, partyBRaceOther, 
partyBRaceOtherText1, partyBRaceOtherText2, 
partyBRaceUnknown, partyBEducation, partyBOccupation);

PartyBAttributes.Next();

Assert.That(LocalOfficial.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("LocalOfficial"));
LocalOfficial.Populate(dateLicensedIssued, waiverSubmitted, 
dateLicenseEffective, dateLicenseExpiration, 
dateIntentionsFiled, countyOfIssuance, 
issuingOfficial, issuingOfficialFirstName, 
issuingOfficialMiddleName, issuingOfficialLastName, 
issuingOfficialSuffix, issuingOfficialTitle, 
issuingOfficialTitleOther);
LocalOfficial.Save();
}

public void EnterMarriageData(string ceremonyType, string placeOfMarriageCity, 
string placeOfMarriageCounty, string placeOfMarriageState, 
string officiantFirstName, string officiantMiddleName, 
string officiantLastName, string officiantSuffix, 
string officiantTitle, string officiantTitleOther, 
string officiantStreetNumber, string officiantPreDirectional, 
string officiantStreetName, string officiantStreetDesignator, 
string officiantPostDirectional, string officiantApartment, 
string officiantCity, string officiantCounty, 
string officiantState, string officiantCountry, 
string officiantZipCode, string officiantPhone, 
string witnessAFirstName, string witnessAMiddleName, 
string witnessALastName, string witnessASuffix, 
string witnessBFirstName, string witnessBMiddleName, 
string witnessBLastName, string witnessBSuffix, string 
dateOfCommission, 
string filingOfficialFirstName, string filingOfficialMiddleName, 
string filingOfficialLastName, string filingOfficialSuffix, 
string filingOfficialTitle, string filingOfficialTitleOther, string 
localFileDate)
{
    RegistrationMenu = new MarriageRegistrationMenu(Driver); 
    Officiant = new OfficiantPage(Driver); 
    FilingOfficial = new FilingOfficialPage(Driver);

    RegistrationMenu.Officiant();

    Assert.That(Officiant.PageDisplays); 
    Assert.That(Driver.Title, Is.EqualTo("Officiant"));
Officiant.Populate(ceremonyType, placeOfMarriageCity, placeOfMarriageCounty, placeOfMarriageState, officiantFirstName, officiantMiddleName, officiantLastName, officiantSuffix, officiantTitle, officiantTitleOther, officiantStreetNumber, officiantPreDirectional, officiantStreetName, officiantStreetDesignator, officiantPostDirectional, officiantApartment, officiantCity, officiantCounty, officiantState, officiantCountry, officiantZipCode, officiantPhone, witnessAFirstName, witnessAMiddleName, witnessALastName, witnessASuffix, witnessBFirstName, witnessBMiddleName, witnessBLastName, witnessBSuffix, dateOfCommission);

Officiant.Next();

Assert.That(FilingOfficial.PageDisplays);
Assert.That(Driver.Title, Is.EqualTo("Filing Official"));

FilingOfficial.Populate(filingOfficialFirstName, filingOfficialMiddleName, filingOfficialLastName, filingOfficialSuffix, filingOfficialTitle, filingOfficialTitleOther, localFileDate);
FilingOfficial.Save();

RegistrationWorkflow.cs

using NUnit.Framework;
using OpenQA.Selenium;

namespace Framework
{
    [TestFixture]
    class RegistrationWorkflow : BasePage
    {
}
private MarriageRegistrationMenu RegistrationMenu;
private ValidationSummaryPage ValidationSummary;
private AffirmationsPage Affirmations;

public RegistrationWorkflow(IWebDriver driver) : base(driver) { }

public void Validate()
{
    RegistrationMenu = new MarriageRegistrationMenu(Driver);
    ValidationSummary = new ValidationSummaryPage(Driver);
    RegistrationMenu.ValidateRegistration();

    Assert.That(ValidationSummary.PageDisplays);
    Assert.That(Driver.Title, Is.EqualTo("Validation Summary"));
}

public void AffirmLicense()
{
    RegistrationMenu = new MarriageRegistrationMenu(Driver);
    Affirmations = new AffirmationsPage(Driver);

    RegistrationMenu.ApproveLicense();

    Assert.That(Affirmations.PageDisplays());
    Assert.That(Driver.Title, Is.EqualTo("Affirmations"));

    Affirmations.Affirm();
}

public void ApproveFiling()
{
    RegistrationMenu = new MarriageRegistrationMenu(Driver);
    Affirmations = new AffirmationsPage(Driver);

    RegistrationMenu.ApproveFiling();

    Assert.That(Affirmations.PageDisplays());
    Assert.That(Driver.Title, Is.EqualTo("Affirmations"));

    Affirmations.Affirm();
}
OrderProcessingWorkflow.cs
using NUnit.Framework;
using OpenQA.Selenium;
namespace Framework
{
    [TestFixture]
    class OrderProcessingWorkflow : BasePage
    {
        private MarriageRegistrationMenu RegistrationMenu;
        private MarriageFastOrderPage OrderPage;
        private PartyAPage PartyA;

        public OrderProcessingWorkflow(IWebDriver driver) : base(driver) { }

        public void IssueNonCertified(string source, string priority, string delivery, string marriageLicense, string quantity, string requestReason, string requestReasonOther, string paymentType, string paymentAmount)
        {
            RegistrationMenu = new MarriageRegistrationMenu(Driver);
            OrderPage = new MarriageFastOrderPage(Driver);
            PartyA = new PartyAPage(Driver);

            RegistrationMenu.IssueThisRecord();

            Assert.That(OrderPage.PageDisplays);
            Assert.That(Driver.Title, Is.EqualTo("Fast Order"));

            OrderPage.Populate(source, priority, delivery, marriageLicense, quantity, requestReason, requestReasonOther, paymentType, paymentAmount);

            Assert.That(PartyA.PageDisplays);
            Assert.That(Driver.Title, Is.EqualTo("Party A"));
        }
    }
}
References


THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK


doi:http://dx.doi.org/10.1016/j.jss.2013.07.028


doi:10.1109/RCIS.2011.6006841


THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK


Wehrmeister, M. A., Ceron, L. M., & Silva, J. L. d. (2012). Early Verification of Embedded
THE DEVELOPMENT OF AN AUTOMATED TESTING FRAMEWORK


