National Oracle Eminence whitepaper
Agile method for composite applications
## Contents

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Approach for developing composite applications</td>
</tr>
<tr>
<td>8</td>
<td>Case Study</td>
</tr>
<tr>
<td>11</td>
<td>Leading practices</td>
</tr>
<tr>
<td>10</td>
<td>Conclusion</td>
</tr>
<tr>
<td>11</td>
<td>Contacts</td>
</tr>
</tbody>
</table>

As organizations continue to look for ways to reduce information technology (IT) costs, increase profit margin, and drive market share, they are often choosing IT projects that aim to leverage existing applications in their portfolio to deliver on these goals. As a result, more organizations are embarking on composite, mobile, and web application development projects that aim to increase the return on investment (ROI) on current IT investments by building new capabilities and increased usability from existing applications. The resulting solution often provides improvements to current business processes as well as to the end user experience.

The development of composite, mobile, and web applications can vary from simple to complex, but the real challenge lies with the approach/methodology adopted to deliver it. There has been a long-standing debate on what methodology is best for delivering projects; some view a waterfall approach as being best suited, while others opt for an agile approach. Both have their merits and shortcomings, which are tied to project constraints such as stakeholders and other internal and external factors that influence the project.

Based on our experience, the preferred approach for delivering these types of applications provides for a combination of components of both methodologies. The waterfall approach provides a framework that can assist in keeping such projects on track and in budget, but an agile approach is preferred when working through elements of the user interface and usability design. By allowing an approach where the requirements and design build gradually as development progresses, IT can roll out these applications with the speed, flexibility, and adaptability composite, mobile, and web applications promote.

This document discusses a software development life cycle for these types of implementations that is rooted in both waterfall and agile principles and which leverages the Oracle SOA suite and WebCenter suite in the rapid development of composite, mobile, and web applications. The approach discussed assumes that an assessment to support the deployment of such applications has been conducted and completed.
Approach for developing composite applications

Waterfall and agile projects take fundamentally different approaches in completing requirements, design, development, and testing activities. It is important to understand the underlying concept behind these methods and why they fall short when they are applied to the implementation and delivery of composite, mobile, and web applications. The following sections provide a high-level summary of waterfall and agile methods and also discusses why applying each method alone for the implementation of composite, mobile, and web applications falls short.

**Waterfall development approach**

The waterfall development approach follows a sequential process in which the tasks/activities performed flow from one phase of the software development life cycle (SDLC) to the next in succession like a waterfall. The series of phases, which begin with requirements gathering and end with product release, have defined start and end times, as well as exit criteria that must be met before the next phase in the life cycle can begin. Figure 1 below depicts a simple application development project over an 18-week period with milestones at the end of each phase.

Waterfall model is typically recommended for implementations where the customers know what they want up front. This approach is also suitable for typical ERP implementations where the requirements do not tend to change over the course of the implementation. Therefore, the requirements are frozen and key design decisions are taken right at the beginning of the implementation. Since documentation plays a key role in ERP implementations, each phase is accurately documented which will serve as reference for subsequent phases and also enable efficient knowledge transition.

Though the waterfall methodology is an effective model, it does not provide the flexibility and agility required to deliver composite, mobile, and web applications commissioned to address changing business process needs fueled by rapidly changing market demand. Furthermore, the development of these applications can be quite complex, so they require a methodology in which requirements are not completely frozen early and that allows end users to be part of the design/development life cycle.

![Figure 1: Waterfall Traditional Waterfall Delivery](image-url)
### Agile Development Approach

Agile development focuses on flexibility and speed by breaking development into smaller interactive blocks. Each sprint or iteration is time boxed and a set of tasks are chosen to be completed in the sprint. The diagram below shows three sprints over a period of 12 weeks to successfully complete a small project using agile methodology.

Due to the agile approaches’ iterative nature, the end product of the project tends to be of high quality due to the heavy stakeholder involvement in requirements gathering, design, development, and testing phases. This is one of the key strengths of agile development approach and why it is favored over the waterfall approach in some circles. Agile also provides opportunities to improve the final product through multiple iterations and frequent releases.

### Challenges in using a pure agile approach for composite application development

#### Managing Cost
It is difficult to develop realistic estimates of the level of work needed to complete the application; this is primarily because at the beginning of the project the full application scope and requirements are not fully known. For many application development projects, which tend to run on a fixed budget, executing the project on pure agile principles might not be the preferred approach. Though embracing change is one of the key agile principles, change comes with a cost. The total cost of the development effort will be the number of iterations multiplied by the cost per sprint. If there are too many changes involved that drive additional iterations, it is possible to run out of budget before delivering an application that fulfills all or most of the requirements envisioned. This is driven by there being no comprehensive requirements gathering phase before the project commences. We will discuss this further in the next section.

---

**Figure 2: Agile Development Sprints**

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Sprint 1</td>
<td>Development Sprint 2</td>
<td>Development Sprint 3</td>
<td>Development Sprint 4</td>
<td>Development Sprint 5</td>
<td>Development Sprint 6</td>
<td>Development Sprint 7</td>
<td>Development Sprint 8</td>
<td>Development Sprint 9</td>
<td>Development Sprint 10</td>
<td>Development Sprint 11</td>
<td>Development Sprint 12</td>
</tr>
</tbody>
</table>

- **D**: Development
- **R**: Requirements Gathering
- **T**: Testing
- **Deploy Product**: Release
**Requirements phase**

Agile projects start with minimal requirements and additional requirements are gathered and integrated during each iteration. For a composite application implementation, it is important to have a solid foundational set of requirements with defined upper and lower limits that provide the framework for the application functionality; this also helps to limit additional requirements to only areas that will provide the largest user benefit.

**Documentation**

Agile development does not focus heavily on documentation; a complex integration of heterogeneous applications used to form a composite application needs to have detailed documentation. Lack of sufficient documentation such as this would pose a great difficulty in understanding how to leverage the disparate applications as well as knowing what the “rails” are to remain between once a sprint begins. Without this information, there would be an increase in the risk of scope creep due to the lack of detailed requirements documentation. Additionally, a lack of sufficient documentation will likely lead developers to speculate on the requirements discussed and proceed with the development based on incorrect information, which may cause unnecessary confusion and rework. In addition, the maintenance team may have nothing to refer to after the project rollout, other than test cases and code bundles.

In order to prevent these issues, proper documentation needs to be produced for both the underlying applications used in the solution as well as the specifics of the new composite, web, or mobile package. This documentation should cover not only the detailed requirements, but also the key design and architectural decisions, functional designs, technical designs, and test results. This information can be useful not only during the project to help speed development, but can also be invaluable after product launch to assist with production support or future enhancements.

**What is the way out?**

The reality is that undertaking a composite, mobile, and web application project requires adopting effective traits from both waterfall and agile approaches. For our discussion, we will call this hybrid approach (waterfall agile model) the “WAgile” Approach.

The WAgile approach will use the effective traits of both worlds to define a new adapted methodology to address the key challenges described above. In this method, a solid requirement-gathering phase is employed, during which detailed requirements are gathered and brainstorming around the current process is performed to enable full process understanding as well as to recommend better processes if necessary. This is one of the key stages from the waterfall model adopted below.

---

**Figure 3**

<table>
<thead>
<tr>
<th>Requirements gathering</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Gather detailed business requirements</td>
</tr>
<tr>
<td>- Estimate and prioritize features</td>
</tr>
<tr>
<td>- Produce initial release plan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sprint planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Update product backlog</td>
</tr>
<tr>
<td>- Identify and estimate iteration tasks</td>
</tr>
<tr>
<td>- Committo iteration Baddog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sprint Phase 2-4 weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define and design</td>
</tr>
<tr>
<td>Build/test</td>
</tr>
<tr>
<td>Validate/document</td>
</tr>
<tr>
<td>Deploy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sprint review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Showcase of “done” features</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sprint retrospective</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Project wrap up</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Project review</td>
</tr>
<tr>
<td>- Training</td>
</tr>
<tr>
<td>- Support</td>
</tr>
</tbody>
</table>
In a typical waterfall model, the end product is delivered after all the stages have been completed (i.e., the requirements, design, development, and test). However, in the WAgile approach, the software will be delivered in an agile fashion after the requirement gathering stage. Each sprint will have a sprint planning sub-phase, which will identify the backlog; determine the requirements which will be addressed; break them down to manageable tasks; and start the development process in parallel.

In the WAgile approach, creating and maintaining detailed requirements documentation and technical design documentation is important during each sprint. As discussed in the previous section, the lack of proper requirements documentation can lead to a number of issues. Additionally, the developer may not be clear on the requirements discussed at the beginning of the project, which may cause unnecessary confusion around the specific requirements discussed with the end users. This may lead to delivering an application that does not address all user needs, or spending significant time in re-work, which will inflate the project budget. Maintaining this documentation throughout the project will likely help to alleviate these issues.

The technical documentation serves as a guide for developers who will work on subsequent sprints to understand the reasoning of key design decisions. It is also important that any new developers who maintain the application be able to review the document in order to provide the necessary support.

Define: The requirements-gathering phase gathers the overall requirements and would have the upper and the lower limits of each requirement and the overall project scope clearly defined. In each sprint, the first few days will be used for the define phase, during which we will define the set of tasks that we will be targeting in that cycle. Backlog from previous sprints is also considered during the define phase, and it is determined whether or not incomplete components from previous sprints will be incorporated into this phase. Tasks are split and assigned to developers/teams. Reusable and utility services should ideally be built first.

Design: In the first iteration, an application design is performed, which is actually part of the sprint tasks. Designing the flow of the application, identifying the tech stack, and creating a solution blue print are also completed during this initial design phase. These documents are updated in subsequent phases only with additional design considerations.

---

**Figure 4**

WAgile Delivery
Build: This is the key stage where application coding is performed. Critical factors such as the usage of reusable services and coding standards should be observed. The code needs to be refactored so that additional changes in the subsequent cycles can be integrated easily. Caution should be exercised when exposing a service that will be available outside the application for consumption. Once exposed and utilized by other applications, they become much more difficult to modify, so they should be deferred until the time they become necessary.

Validate and document: Unit testing is performed for the code and additional automated test scripts are written for the newly developed code. The already available test scripts will be executed to ensure the newer code has not introduced any defects.

Technical documentation should be updated with key design decisions, the functionality of the artifacts developed, database schema design, and performance benchmarks. This will act as a reference for the subsequent sprints and maintenance phases.

After these phases, the application is deployed and is ready for use. Feedback is gathered from the existing build and incorporated into the next sprint. The figure below provides a summary of the key documentation activities that occur during the phases for a given sprint.

Benefits of the WAgile Approach

User acceptance
One of the key causes of composite application failure is that the application post-development does not operate or function the way the end user envisioned. Though it might achieve the goal of streamlining business processes, the stakeholders may have expected a different application look and feel or a different user experience. On longer projects, which may take over a year to complete, the end user has to wait until near the end of the project to see how the application looks and behaves. If the project stakeholders are not satisfied with the outcome, then potentially the time and money spent on this effort goes to waste.

WAgile development helps to address this pressing issue by embracing incremental development. This means that the solution will evolve over iterations. The first iteration will carry minimal features and will give stakeholders a view of the application from the very early stages. The subsequent iterations will incorporate their feedback, and they will be able to help guide the evolution of the application through the design – development – feedback loop.
Design complexity
Composite application development is usually technically complex as it integrates multiple applications, such as legacy enterprise resource planning (ERP) systems, mainframe applications, point solutions, hosted Software as a Service products, and master data management (MDM) running on various platforms and technologies. The WAgile approach helps to decrease the complexity of application development by breaking the functionality into manageable chunks during each sprint. This allows the developers to focus on portions of desired process and properly design the underlying integration of the composite more effectively than trying to do a complete design up front and then having to deal with incorporating changes for the “gotchas” discovered later in testing.

Responsiveness to change
Change is virtually constant in software development. When building composite applications in the WAgile approach, the impact of these changes often become more manageable than in a waterfall approach. Since the services and components developed are modular in fashion, the reusability and the reversibility are higher. Also, since the design – development – test cycles are iterative, it makes it much easier to incorporate changes identified later in the project.

Mitigate risks
A vital differentiator between agile and traditional methodologies is that agile focuses on a deep collaboration between the stakeholders and the development teams. This helps identify project inhibitors in the early stages and also gives the end users much more of a hand in the application being developed. By leveraging this key aspect of agile development, the WAgile approach helps to mitigate risks by addressing stakeholder ownership, user experience design, functional usability, and user adoption issues.

Higher quality
The composite application is tested several times before the final release. This aims to ensure the quality of the software will be much higher, compared to software developed in a traditional approach. Applications developed using WAgile methodology will likely gain wider acceptance (as the users have been seeing the application for a while) quality is much higher with very few defects in the final stage.

The following section discusses the case study for a high tech manufacturing company looking to increase its current customer base, improve current business process practices and provide single front end to end users by leveraging existing IT investments.
Case study

Company ABC is a high tech manufacturer that conducts business nationwide. To support its businesses nationally, the company uses distributors and channel sales partners to distribute its products. Materials are held at various warehouses which can be a central or satellite warehouse location. Given the rapidly changing technology landscape and increased competition, the CEO wants to expand the current customer base by offering new products and innovative ways to order and track product information. The CEO wants to transform the company into a recognized global industry leader with the ability to provide “just-in-time” information services to support the pursuit of these new customers effectively and efficiently. The CEO wants to leverage current IT investments to achieve her goals and remain competitive in the market.

Below are some of the stated business requirements:

- Identify and create an inventory of application functionality and expose the critical functionality of each application through a single user interface
- Leverage pre-built solutions and services to accelerate the delivery of the solution
- Seamlessly integrate disparate applications and provide a consistent homogenous user experience
- Enable customers to order inventory directly from Company business partner website, and orders are typically LTL (Less than Truck Load).
- Enable customers to connect to the business partner website and track status of orders using information overlaid on top of mapping application

As a first step, the existing business functionalities were identified and built into process flows. These flows were reviewed by both the UI development team and the business integration development team. Some of the key business process flows implemented in the application were:

1. Order creation, updation and cancellation
2. Order search and status
3. Invoice creation
4. Invoice search
5. Customer registration
6. Discounts

As part of the first phase, the requirements gathering team discussed the current business processes followed by the customer and the challenges faced on the existing system. They also virtualized the end state of the product, during which the upper and a lower limit of requirements were set. For example, the customer had multiple third party systems which dealt with HR, Finance and Order Management systems. It was decided that in the current transformation scope, only Order management applications were to be considered.

Specific business processes were identified and defined in scope. As part of this phase, the requirement gathering document was prepared and was presented to the customer. After a detailed conference room pilot session, the requirement documents were signed off by the client. This document was used to finalize the upper and the lower level requirements of this engagement.

In the first sprint, the solution architect on the team reviewed the process flows designed in the requirements gathering phase and decomposed them into services to be orchestrated to support the desired business functionality. The resulting services were developed and registered with Oracle Service Registry. Common utilities like the error logging and reporting framework were designed, developed and documented.

Oracle Webcenter was chosen for its ability to present user friendly frontends quickly and dynamically. One of the key benefits of using Oracle Webcenter is its ability to improve productivity by providing rich user experience to its customers to access contextual information.
The UI development team began their design work in parallel by outlining the wireframes for the screen/page flows for the application based on the process flow diagrams. The UI development using Webcenter started once the WSDL contracts for the services developed by the business integration team were shared. The business services layer was built on existing and new web services. As the application functionality was encapsulated in the web services, no business validations were performed in the UI layer.

The subsequent phase began with the define stage where the tasks for the stage were identified. The first component developed was the order search service. Once the service was created using Jdeveloper and tested using webservices testing tools, the WSDL contracts were shared with the UI team. The UI team started their development work on the Order search screen. As per leading practices, the sharing of the WSDL contract was delayed until it was tested completely by the webservices team. This reduced the rework when changes were required during subsequent test cycles. Once the order creation screen was developed as per the wireframe design, appropriate functionality testing was carried out and documented.

The code base at the end of the sprint was source controlled and baselined. The end application was deployed and released for testing for the stake holders. Additional functionality and feedback from the stake holders were incorporated in subsequent sprints and the product was deployed and released for usage at the end of each sprint. Quality and acceptance by users of the product was promoted by the constant involvement of the stakeholders.

At the end of the last sprint, as part of the project wrap up, all the documentation was handed over to the client IT team and a roadmap to implement a support methodology of the software was presented to the client.

During each phase of development, the leading practices specific to WAgile development was followed. The leading practices are described in detail later in this document.

The above reference architecture diagram was leveraged in identifying the different tools and products required to build and deliver the composite application. Figure 7 below is the solution architecture diagram developed for the solution.
Gather requirements: Gathering requirements is one of the most important phases. A composite application developed will likely gain the user acceptance primarily because of two reasons; it solves the current business problem which in many cases is disparate systems integrated into one; and secondly, the ease of use. To this end, the requirement-gathering sessions will act as a forum to understand the current business process, brainstorm and find improvement areas, and discuss challenges faced in day-to-day activities. This phase should involve the end user champions and key business process stakeholders.

Design: The design of the composite application should be loosely coupled, and, at the same time, cohesive. Though the underlying components are modular, the end user should be able to accomplish a business task with ease from the application. The design should be loosely coupled and we should be able to smoothly swap the participating application in the composite application. For example, a tax calculation service should be separate rather than having that inbuilt as part of a payroll software.

The design could be thought of in advance, but must be acted upon only when necessary. We should only design and develop software for solid requirements in hand and not work on speculative requirements.

Extreme caution should be exercised before exposing services externally, as they may prove difficult to reverse. Enough thought process should go in.

Common coding guidelines: Multiple teams will develop their components in parallel and this requires a common coding guideline. This is one of the first activities for a development team; the team should discuss and arrive at a consensus over the standards. It can be minimal, but the standards should be set before the first iteration, and importantly, be adhered to in the subsequent iterations. Some of the key coding standards include the usage of common utility services or methods, naming standards, the tools or integrated development environments to be used, and a common error handling and reporting framework, design, and document templates.

Refactoring: Martin Flower says, “Refactoring is a process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure” As we discussed, WAgie project iterations are naturally short and developers tend to overlook this step. Refactoring brings the following benefits:

- Modular code
- Code easier to understand
- Easier to modify
- Easier to add new features

WAgie is iterative, and features are added in each iteration — these activities are a must for the coding process. It should be part of the developers DNA to refactor the code before it is released for testing.

Code regression testing: As discussed above there are two challenges in testing the application when following a WAgie approach. The testing effort increases and the complete composite application have to be retested during each iteration. The second is due to the unavailability of the target system to test, as that is also being developed.

To tackle the first case, various test automation tools should be evaluated and identified. For example, for testing the SOA composites developed by the Oracle SOA suite, the usage of a composite test framework will be effective to automate a repeatable test of composite applications. The manual work could be reduced.

Usage of emulations enables the developer to simulate the behavior of the binding services outside the components with which your SOA composite application interacts during execution.

Active stakeholder’s participation is a must in this case. Though this is a challenge as it needs continuous involvement and in normal cases, the stakeholders will have their day job to take care of. An effective situation would be to have a dedicated set of individuals from the client’s business team to test and share feedback.

Active stakeholder’s participation: Feedback gathering is an integral part of WAgie development. As discussed in the previous section, stakeholder involvement throughout all the phases is important. After each sprint, the deployed application has to be tested by the stakeholders to ensure the development has produced effective results. Based on the feedback received, the changes should be incorporated in the next sprint.
Conclusion

The demand to deliver a user interface that provides the functionality and features available in existing applications and systems continues to grow within companies. To meet the demand, IT organizations should consider employing a hybrid development approach (W-Agile), which provides the speed and flexibility needed to deliver composite, mobile, and web applications that address the business needs of the company. This approach offers the efficiencies of both waterfall and agile methodology through flexibility in requirements definition controlled by set criteria for adding to, or removing from the requirements based on customer feedback. Furthermore, the iterative nature of this approach enables rapid deployment releases of the application to the customer for review. This enables the customer to be part of the overall solution since they see it evolve and provide feedback on the functionality provided as well as the usability of the solution.
For more information about this paper, please contact:

Yusuf Lawal
Manager
Deloitte Consulting LLP
111 South Wacker Drive
Chicago, IL 60606
+1 312 486 4346
ylawal@deloitte.com

Dwaraknath Krishnamurthi
Manager
Deloitte Consulting LLP
123 Divyasree Technopolis, Yemlur P.O
Bengaluru, India 560037
+91 990 050 2738
dwkrishnamurthi@deloitte.com

Rattan Singh
Senior Manager
Deloitte Consulting LLP
25 Broadway
New York, NY 10004
+1 212 618 4610
rattansingh@deloitte.com

Donald Schmidt Jr.
Director
Deloitte Consulting LLP
111 South Wacker Drive
Chicago, IL 60606
+1 312 486 5290
doschmidt@deloitte.com

Clint Cummings
Director
Deloitte Consulting LLP
191 Peachtree St,
Atlanta, GA
+1 404 631 2795
ccummings@deloitte.com

For more information on the Deloitte National Oracle Eminence initiative, please contact:

Mike Reid
Director
Deloitte Consulting LLP
695 Town Center Drive, Suite 1200
Costa Mesa, CA 92626
+1 714 436 7669
mireid@deloitte.com

Mike Brown
Principal
Deloitte Consulting LLP
299 South Main Street, Suite 1900
Salt Lake City, UT 84111
+1 801 366 6876
mikbrown@deloitte.com