accentureconsulting

Strategy | Consulting | Digital | Technology | Operations

Technical Analysis

High performance. Delivered.

Table of Content

1 Executive Summary

1.1 Scope of Analysis

Accenture has in accordance with the task defined by Skatteministeriet conducted a technical analysis to review the EFI and DMI Applications, their integration with each other and the rest of the SKAT IT estate.

This technical analysis is based on review of selected parts of the EFI and DMI Applications as well as selected processes undertaken during the development of EFI and DMI. The report is made on our experience and assumption that the conclusions in the report are representative also for the part of the EFI and DMI Applications that have not been reviewed.

The report does not give a full picture with regards to the current state of EFI and DMI, the reasons that have led to the current state of the Applications (EFI and DMI), or if the current state of the Applications are consistent with the original contractual requirements as described in the EFI and DMI contracts.

The scope of the analysis included the analysis, design, build and test phases of the EFI and DMI Applications, and the combined EFI+DMI System. Most of the analysis was performed on a sample basis. Our approach has been to select core areas that are essential for the system operation, such as receipt of a claim, salary deduction, payment plan and order of coverage, for CPR customers. These are found in almost every collections system. There is a risk that this could lead to a misleadingly positive view of the System, as this core functionality is the functionality most likely to be used and therefore working.

1.2 Methodology for the Technical Analysis

The methodology for the technical analysis was to compare the approach used to build the System to a normal system building approach. The Accenture Delivery Methods (ADM) was used as the reference for the comparison. ADM provides a comprehensive definition of the required processes, tasks and deliverables that should be completed across the phases of the Software Development Life Cycle (SDLC).

As each phase of the SDLC was assessed, relevant documents were identified, examined and assessed. ADM was used as the reference and framework with a degree of flexibility. The overall EFI Programme and EFI and DMI Applications were not created by Accenture and therefore were not created using the Accenture Delivery Methods. Therefore, a degree of flexibility was required in assessing the existing documents and materials against ADM. Where content equivalent to an ADM document was available in other document(s) these were accepted and assessed.

Common industry practice is to describe software systems by detailed documented requirements that define what the system must do, in order to operate correctly. Requirements are used as the basis of testing (i.e. validation) that the system actually performs these functions.

Early in the analysis, it was identified that the Original Requirements for the Applications and System contained less detail than expected for a system of this scale, in our experience.

The analysis approach was extended to include an exercise to document the Needs (requirements) for the System, and to trace the existence of these needs through the Applications and System, in a sample of core areas for debt collection.

A detailed description of the methodology for the assessment is provided in Section 3.

1.3 Key Findings

What is the Current State of the System?

Documentation & Design

- Because of the lack of detailed documented requirements, it is not possible to accurately describe the state of the System.
  - There is not a set of detailed documented requirements for the EFI or DMI Applications or the combined EFI+DMI System.
  - Normal practice is to describe the completion state of a System in terms of the percentage of detailed documented requirements that it meets, validated by testing.
  - Therefore, it is not possible to accurately describe the current state of the System in a normal way, or to complete the System.
- During the design of EFI and DMI there was no documented end-to-end process description of how the overall solution should work to process claims from start to finish. The high-level designs that did exist (use cases) were separate for EFI and DMI. These were not integrated and they were also not used to validate that the end-to-end solution worked

Structure

- EFI and DMI are separate Applications. They are developed and deployed as separate technical components. Together EFI and DMI are a single System with integration into the SKAT IT estate.
  - Neither EFI nor DMI can perform any useful collections function without the other. For example, screens that display the details of debts or interests require EFI and DMI to interact.
  - It is our opinion they should therefore be considered as a single software System that should have requirements, processes, designs, and tests for the System.

  - When EFI or DMI require replacement, there is either a very large body of work required to re-implement the same interfaces again on the new component, or both EFI and DMI must be replaced together.

- The debt collection System is separated into two tightly integrated Applications. In our experience collections System are usually implemented as a single application. As a result of this separation, performance, reliability, maintenance costs and functionality are negatively impacted.

- EFI and DMI has been designed with the aspiration of providing future flexibility. Some flexibility has been provided, however it is not clear this level of flexibility was required. Most of the flexibility has not in fact been used to date. It is not clear all the functionality controlled by the flexibility works.

- The interfaces between EFI and DMI and the systems they interact with including the ones within the rest of the SKAT were insufficiently designed. This led to a number of problems such as:
  - Extensive, ongoing change being required over a long period to fully evolve the interfaces.
  - EFI and DMI Applications taking more time and effort to finalise.
  - Making the testing of realistic paths through the code difficult due to being unable to create accurate program stubs. Stubs are normally used in large system implementations to test components or applications well before end-to-end (complete system) testing is performed. Realistic stubs, which enable testing of more than the simple path depend on detailed designs

Technical

- EFI does not implement rigorous external validation on input data from external Systems or end users. EFI+DMI is partially or completely unable to process records that are missing key fields, and lacks mechanisms to manage the consequences of this. As a result, the System cannot process some real world data
- Most automation in the System has been disabled since the go-live. The event based processing approach, used for “monitoring” results in excessive numbers of events and changes from a customer perspective. In practice, this means that if the automation were enabled as originally designed and implemented, customers would receive excessive numbers of communications and events from EFI+DMI
- DMI is the financial engine at the core of the System. Unusually for a financial application it does not enforce Referential Integrity. This has allowed DMI to store invalid records. This reduces the data quality in DMI and the overall integrity of financial processing in SKAT Collections.
- DMI represents an unusual approach to an SAP Revenue implementation. There is a low usage of SAP package functionality combined with an unusually high percentage of custom objects. The consequences of this are that the licensed SAP product functionality is relatively lightly used. DMI is in fact, mostly a custom

Application, rather than an SAP application. This will impact maintainability over the lifespan of DMI

- The source code for EFI is slightly better than average quality, the source code to DMI slightly lower than average quality. The quality refers to technical aspects of the code such as complexity, error handling, maintainability, cohesion and commenting. The source code quality is not a primary cause of problems with the System

How did the System get into this state?

- The overall testing approach was compromised both by the lack of detailed documented requirements to test for, and the test approach. Analysis of the testing records confirmed with the EFI Programme test manager shows that 53.8% of the originally planned tests were executed. 52.3% of the original tests (80.5% of non-descoped tests at go-live) were recorded as passing testing. This resulted in a substantially untested System being put into production use
- The scope of the System was not well defined at any stage. The complexity of the Original Needs was not well described in the Original Requirements and use cases, or at any subsequent phase. It is our opinion that the complexity of implementing the variety of debts and legal persons, combined with the extent of automation was never fully understood. This resulted in essential functionality being omitted from the System
- There was a pervasive approach through the analysis, design, build and test of the System to focus on the "simple path", the most common path through the System. Throughout the Original Requirements, use cases, designs, code and tests the System is missing the detail required to process the variety and complexity of situations encountered in production. This also resulted in essential functionality being omitted from the System
- There was no design (solution blueprint) for the overall EFI+DMI System, either at the start or since. Although some materials exist, these fall far short of what would be expected to design a System of this complexity. Based on this lack of documentation it is our opinion that the System level "SDLC" activities were largely not performed. This resulted in problems integrating EFI and DMI, causing delays and defects to the overall EFI Programme
- There was an assumption that the System would receive valid data from the internal and external systems feeding EFI/DMI with claims, and from all users inputting data manually. This assumption is identified because (a) the System does not do normal levels of validation and (b) it was stated in requirements workshops that the EFI Programme had no mandate to control incoming information. This does not appear to be a reasonable assumption, based on our experience of other systems, and the fact that non-valid data has been input to the System. This resulted in the System failing to process correctly when presented with incorrect production data
- Assessments and reports by neutral external experts between 2012-13 prior to the go-live date identified lack of design, testing and migration as risks. These risks

were communicated to senior EFI Programme representatives.

Chronological Context for EFI Programme and Findings

The events analysed in this report are depicted on the timeline below. Events depicted with grey are unconfirmed dates, where the EFI Programme has not confirmed the exact dates. There are a number of key points with regard to the overall timeline.

- EFI and DMI were separate projects, performed by separate vendors, and managed by SKAT.
- The most significant is that the design of EFI and DMI was not performed in a tightly coordinated manner. Given the number and the complexity of the dependencies between EFI and DMI, close coordination would have been required between the Applications for the System to integrate correctly.
- Overall, there was poor stage containment (completing and closing phases of work, before progressing to the next phase) both within the EFI and DMI Applications, and overall for the combined EFI+DMI System.

In our opinion, the original project timeline was not feasible due to the technical dependencies. The iterative nature of resolving the dependencies resulted in additional work and suboptimal design.

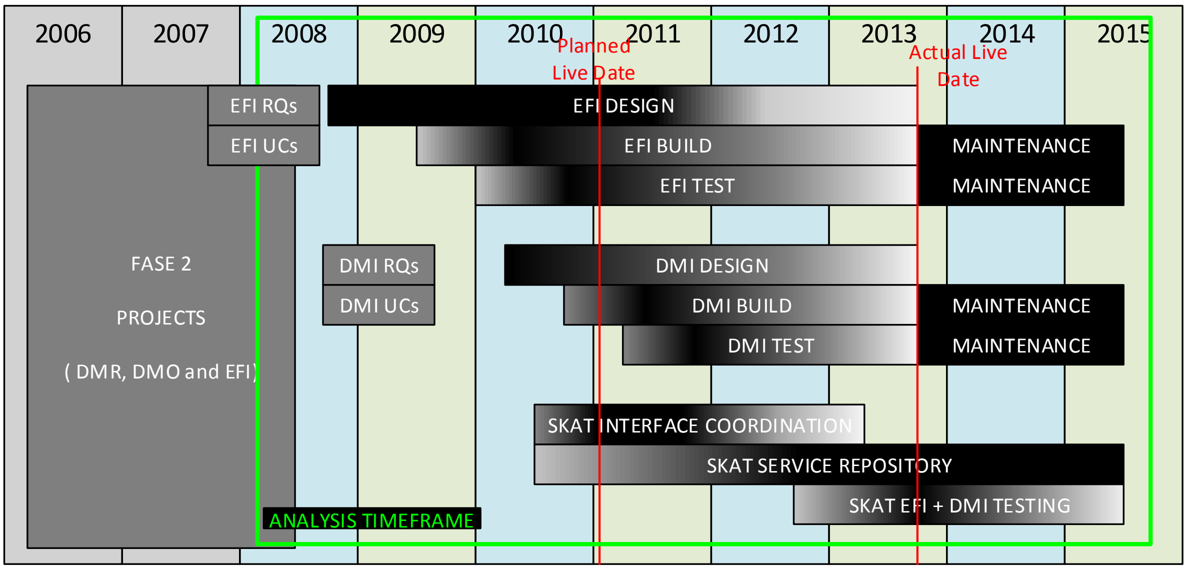


Figure 1 Timeline

2 Scope

2.1 Scope of Analysis

Accenture has in accordance with the task defined by Skatteministeriet conducted a technical analysis to review the EFI and DMI Applications, their integration with each other and the rest of the SKAT IT estate.

This technical analysis is based on review of selected parts of the EFI and DMI Applications as well as selected processes undertaken during the development of EFI and DMI. The report is made on our experience and assumption that the conclusions in the report are representative also for the part of the EFI and DMI Applications that have not been reviewed.

The aspects of the report does not give a full picture with regards to the current state of EFI and DMI, the reasons that have led to the current state of the Applications (EFI and DMI), or if the current state of the Applications are consistent with the original contractual requirements as described in the EFI and DMI contracts.

This analysis included the following:

- System Requirements Review
- Review of the EFI and DMI Applications
- Software Development Life Cycle (SDLC) review
- Automated and manual code review of EFI
- Semi-Automated code review and functional review of DMI
- Thorough analysis of a sample of core areas described below

As part of the analysis, we planned to trace requirements through the phases of development, from analysis through design, build and test. Although there were Original Requirements and use cases, the analysis team assessed that that these did not provide sufficiently detailed documented requirements to provide the necessary input for design, build and test. Therefore, in order to trace requirements through the Software Development Life Cycle we had to run workshops with EFI Programme representatives to gather and document detailed Needs for a sample of core areas. The analysis team selected a number of core functional areas that are central to the collections function. These are found in almost every collections system. The selected sample areas were as follows:

Area	Focus Area
Modtag Fordring (Receive Claim)	Validation: legal rules applicable to claims

Kundesaldi (Client Account Balance)	Order of Coverage: distribution of payments across claim(s), principal, interest, expiration interruption
Betalingsordning (Payment Plans)	For CPR (personal customers), automatic initial creation of a forced payment plan, adding of new claims to existing mandatory payment plan
Lønindeholdelse (Salary Deduction)	Main functionality for starting and stopping deduction, notifications/decisions, and aging of claims during deduction.

Table 1 Sample Areas

2.2 Limitations

This analysis is based on reviews of the documentation stated in the Appendix as well as workshops and interviews. This analysis has only examined the Original Requirements and use cases documents listed in the Appendix. The design, build and elements of the test of EFI and DMI Applications are performed by separate project teams. The scope has not included investigation of internal processes (e.g. unit testing) within these teams.

In many cases, the analysis has been performed on a sample basis (e.g. review of a sample of source code or designs) or a time-boxed basis (e.g. limiting the number of requirements gathering workshops). Where this has been necessary this is noted in the detailed description of scope (Section 2) or in the relevant Assessment Details description.

2.3 Scope of Requirements Gather and Trace

As described in Section 2.1 above, it was identified early on that there was no detailed documented definition of the Needs for EFI or DMI. Detailed, documented requirements are a dependency for design, build and test of a software system. In order to resolve this issue, the analysis team ran requirements workshops to gather and document in detail the Needs for a sample of core areas.

The areas selected are core to collections functionality. Within these core areas, we analysed the most common situations e.g. adult personal (CPR) customers. Less common scenarios e.g. personally owned businesses (PEF) and children were not included in the scope. The rationale for this approach was (a) to examine the System in areas where the majority of claims were processed and where it was expected that the System would be working and (b) it was expected that in these core areas the Needs would be well understood.

The difference between our approach and the “simple path” approach described in Section 5.10 is that our requirements analysis strove to gather and document the full complexity of Needs (all needed paths) within a small, commonly used area of the System, whereas in the simple path approach, the full needed complexity is missed, because some needed paths are not identified. Obviously, there is a risk that this could lead to a misleadingly positive view of the System, as this core functionality is the functionality most likely to be used and therefore working.

Statement on thoroughness of requirements analysis:

- It was necessary to perform the requirements analysis within a limited timeframe. Three (or four) workshops were allocated to perform the requirements gathering, typically with 2-4 days between workshops for requirements owners to perform additional research and analysis
- The approach taken was to analyse the Needs in limited, core areas in as much detail as possible (ideally to full detail)
- In practice, it was necessary sometimes to limit the scope of the area, due to the constrained timeframe lack of EFI Programme representative knowledge of the area or lack of legal stakeholder knowledge of the area. To the extent possible, scope limiting was done through exclusion of additional breadth (e.g. only considering adult CPR customers, and excluding minors, PEF and CVR) while ensuring that the full depth of Needs in the examined areas were identified

Kammeradvokaten’s Statement on Requirements Analysis:

- Kammeradvokaten did not have sufficient time to perform the factual research and thereto related legal research necessary to provide fully validated opinions, and has provided the following guidance about status of the legal requirements provided
- We have assisted the analysis team and SKAT during a series of workshops in ascertaining a non-exhaustive list of legal needs/requirements concerning four different parts of EFI, namely “Modtag fordring”, “Tvungne betalingsordninger”, “Lønindeholdelse”, and “Kundesaldi”. During this process we have assisted in explaining the legal framework (the main rules) of each of the areas, in as far as the facts of the matter have been presented by SKAT etc., and in as far as the legal framework was within the scope of the workshops. Our contributions have thus been our best effort to describe and explain the (main) rules of the legal framework
    - Our reviews have in turn been made on the basis of the facts, which we have been presented by SKAT etc. We positively know that the facts we have been presented have not been exhaustive. In consequence, our presentation of the legal framework, needs and demand is also not exhaustive. There are several exceptions, modifications and situations, which we have not taken into consideration. Many of these due to the simple fact that we have not contemplated or fully clarified their existence/relevance. This means that there are many situations outside the main rules, which we have not discussed during the workshop nor have we reviewed on them afterwards
    - Our contributions and review have in no small extent been based on assumptions on facts, which have not been tested. In no small extent has it been impossible – within the timeframe – to ascertain the actual facts. An example of this would be that it has until now not been possible to achieve clear knowledge on (all) the claim types in EFI despite several requests and meetings with SKAT. This seems to be due to the fact that no detailed overview and knowledge concerning this issue is present within SKAT

- During the process we have also not validated whether EFI's application of actions, which are described in legal conceptual terms, are executed according to these legal terms. An example of this would be that we have not validated whether EFI's conceptual application of the terms pro rata liability or solidary liability are made in (full) accordance with these concepts legal framework. Throughout our review we have assumed that EFI's application of these terms/concepts is in (full) accordance with the legal framework for the terms/concepts. Whether this is actually the case is only possible for us to validate if we receive an exhaustive walkthrough of EFI's handling of the individual concept. We have assumed that such a walkthrough would be outside the scope of this analysis of the chosen parts of EFI. We have also until now been unable to ascertain such knowledge from SKAT. This also seems to be due to the fact that no such clear knowledge is present within SKAT

2.4 Assumptions

The purpose of the report has not been to consider who is responsible for the decisions taken during the project execution. The report does not include a legal review or an assessment of the fulfilment of contractual obligations under the EFI and DMI contracts. The report can thus not be used to conclude whether or to what extend any of the parties involved in, the project execution can be held legally responsible for their involvement in the project.

The report is based on the information and documentation provided to the analysis team by the EFI Programme. The documents examined and meetings are listed in the Appendix. In most cases, a sample based approach was applied. In these cases, the approach was to select core, commonly used areas. The reason for selecting these areas was to take a conservative approach – focusing on areas that would be expected to work. Unfortunately, there is a risk that this shows a more positive view than in reality, as peripheral areas, which in some cases are known to be more problematic, receive less focus.

We have had parts of this analysis reviewed by EFI Programme representatives who have reviewed and confirmed many, but not all, of the facts in this report. Specifically, incomplete or no review feedback has been received on the following:

- The trace of Needs (definitive, documented requirements) through the Original Requirements, use cases, designs, code and tests (Section 5.1). They were issued for review on 30th June 2015, but no feedback at all has been received
- All non-contractual dates (e.g. start and end of interface coordination in Section 4)

Content in sections 5.2, 5.3, 5.4, 5.5, 5.8, 5.9, 5.10, 5.11, 5.12 are based on materials supplied, but the detailed findings have not been reviewed by EFI Programme representatives as of this version. If difference evidence is supplied, or missing documentation identified, the findings may change.

3 Approach & Methodology

3.1 Approach

A systematic and structured approach has been taken to perform this analysis

This has included the following phases:

- **Initial orientation**: During this phase, an initial assessment of the EFI+DMI System was performed. This included meetings with stakeholders and an initial review of documentation (Original Requirements, use cases, designs, tests…)
- **Definition of analysis scope and approach**: The term "Software Development Life Cycle (SDLC)" is used to describe the process of creating a software system. Based on the initial orientation it was determined that a number of analyses were required to examine specific aspects of the SDLCs used to create the current EFI and DMI Applications and the overall EFI+DMI System
- A central part of this analysis was **establishing and tracing definitive requirements (Needs)** through each stage of the SDLC to examine where problems and issues arose
- Other analyses examined other aspects including the **end-to-end** design and testing processes, and **the approaches and principles** used to guide the creation of the System
- **Analysis execution**: The analysis process was performed. This involved workshops with stakeholders, review of documentation, code and other SDLC artefacts
- **Report creation and finalisation**: The report has been drafted, reviewed with relevant experts and finalised

3.2 Methodology: Establishing and Tracing Needs

The illustrative model below shows the major phases in a Software Development Life Cycle (SDLC). Each phase should align, with each phase performed completely. In this analysis, we took a number of sample areas, documented the Needs, and then traced the alignment of these Needs through the following phases and artefacts.

The methodology for the analysis was based on tracing Needs through each phase of the SDLC. The purpose of doing this is to establish whether the Needs were implemented correctly by the System or Applications. For example, in order to implement a Need, it would be expected that

a) The Need is documented in detail. A requirement should be clear, unambiguous and provide sufficient detail for design and test. Frequently requirements are initially created at high level, before being refined to a more detailed level.
b) The Need is included in designs, source code and tests in order to implement the Need, and validate that the Need is in fact implemented.

c) There is documented traceability to identify where and how the Need is implemented. This could be a spreadsheet or specialist requirements tracking software. This traceability is critical, to ensure that when Needs change that the impact of the change to Needs can be identified on designs, code and tests accurately.

The diagram below illustrates this concept. Conceptually, there is an Original Need that is "complete", and all the other phases of work should be equally complete, and align to the Original Need. This must be demonstrated through documentation for the Need in each phase.

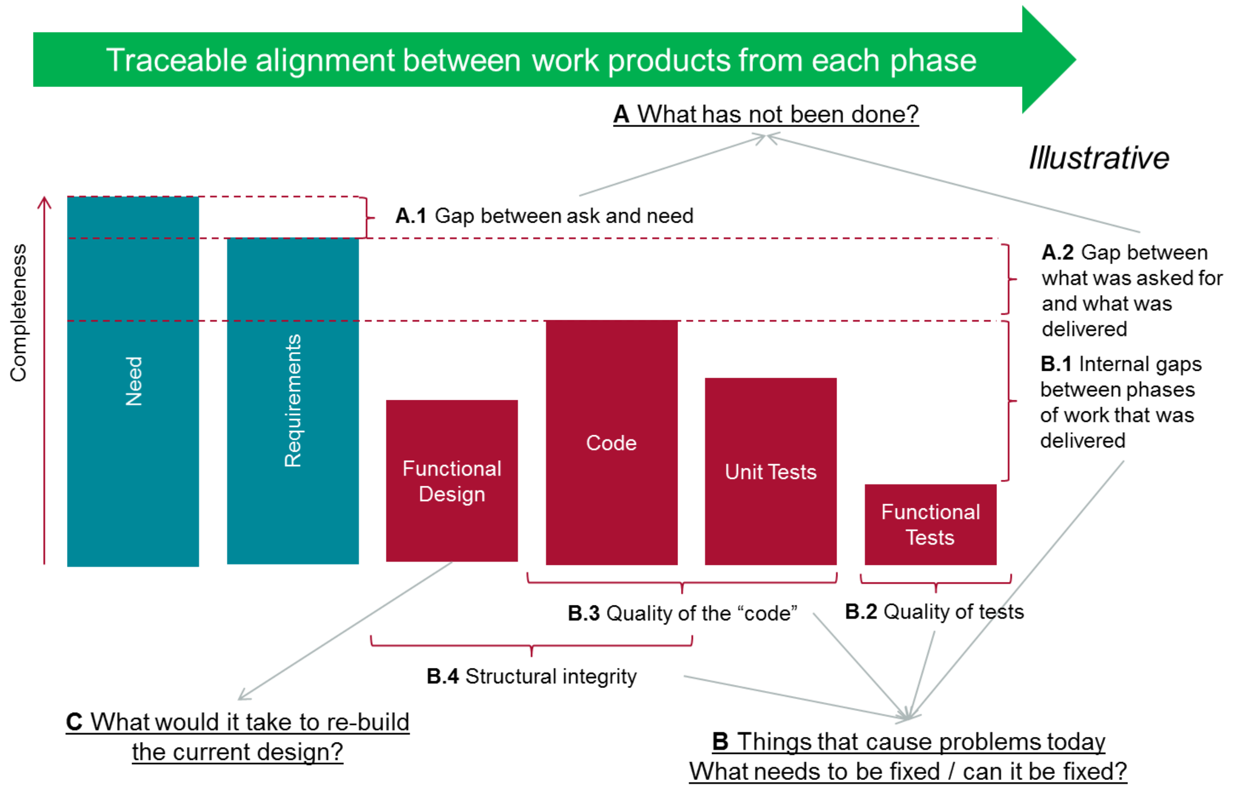

Traceable alignment between work products from each phase
A What has not been done?
Illustrative
Completeness
Need
Requirements
Functional Design
Code
Unit Tests
Functional Tests
A.1 Gap between ask and need
A.2 Gap between what was asked for and what was delivered
B.1 Internal gaps between phases of work that was delivered
B.3 Quality of the "code"
B.2 Quality of tests
B.4 Structural integrity
C What would it take to re-build the current design?
B Things that cause problems today What needs to be fixed / can it be fixed?

Figure 2 Tracing Needs

		Activity	Outcome
A	A.1	Map EFI as-is processes, and create a high level view of current functions and status. See functional report.	Understand what parts of the process • EFI covers and is in use today • EFI covers but is not in use
	A.2	Take sample areas, gather requirements, based on Original Need Section 5.1	Understand what parts of the process • EFI does not cover and was not in the Original Requirements and use cases
B	B.1	Traceability review (done by reviewing documents and code) Section 5.1	Identification of breakpoints in the flow from Original Requirements and use cases, through design to code and test artefacts
	B.1	End to end SDLC review: coordination between EFI and DMI Sections 5.2, 5.3, 5.4,5.5	Assess approach for coordination between EFI and DMI components of collections.
	B.1	Design depth review (done by comparing design and code manually) Section 5.1	Does the design document the functionality of the System as it is implemented today?
	B.2	Review test case quality Section 5.1, 5.10	Do test cases cover the functionality intended?
	B.3	Automated code review Sections 5.5.4, 5.7	Quality of code delivered (syntax, use of comments in code, code complexity etc.)
	B.3	Manual code review Sections 5.5.4, 5.7	Verify structural aspects of code – modularity, use of common services etc.
	B.4	Structural review (collection of observations throughout the other work streams) Section 5.9, 5.11, 5.12	Is the structure as it is implemented today (both functional and technical) such that the System can work?

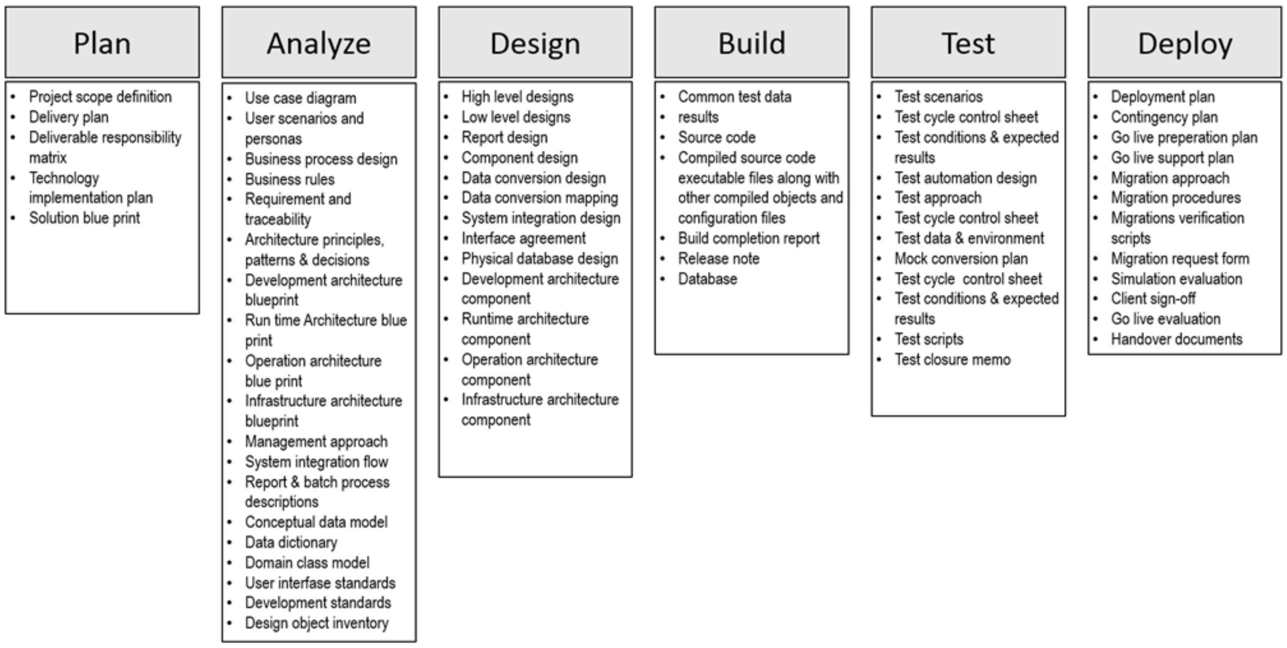
Table 2 Activities and Outcomes

3.3 Methodology: Software Development Life Cycle Assessment

The Accenture Delivery Methods (ADM) was used as the basis for elements of this assessment. There are a number of different ADM packages each tailored to the specific needs of different types of project. In performing this assessment, the ADM for SOA/BPM (Version 1.1) and ADM for Custom Development (V5.5) were primarily used. On overview of ADM is provided in Section 10.

In the context of our analysis, ADM has been used as a reference for processes, tasks and deliverables which should be completed across the phases of the project. ADM checklists, examples and templates have also been used as references for the deliverables that should be in place and for the content of deliverables.

When using the ADM, we did apply a degree of flexibility especially in terms of the references and framework, as we understand that the overall EFI Programme and EFI+DMI Applications were not created by Accenture and therefore were not created using the ADM. Therefore, we ensured that we applied a degree of flexibility when assessing the existing documents and materials against the ADM. Where content equivalent to an ADM document was available in other document(s) these were accepted and assessed against our standards. For example, although ADM prescribes the documents below as required, where documents supplied were substantially able to provide the content of a document below, this content was used.



Plan

- Project scope definition
- Delivery plan
- Deliverable responsibility matrix
- Technology implementation plan
- Solution blue print

Analyze

- Use case diagram
- User scenarios and personas
- Business process design
- Business rules
- Requirement and traceability
- Architecture principles, patterns & decisions
- Development architecture blueprint
- Run time Architecture blue print
- Operation architecture blue print
- Infrastructure architecture blueprint
- Management approach
- System integration flow
- Report & batch process descriptions
- Conceptual data model
- Data dictionary
- Domain class model
- User interfase standards
- Development standards
- Design object inventory

Design

- High level designs
- Low level designs
- Report design
- Component design
- Data conversion design
- Data conversion mapping
- System integration design
- Interface agreement
- Physical database design
- Development architecture component
- Runtime architecture component
- Operation architecture component
- Infrastructure architecture component

Build

- Common test data results
- Source code
- Compiled source code executable files along with other compiled objects and configuration files
- Build completion report
- Release note
- Database

Test

- Test scenarios
- Test cycle control sheet
- Test conditions & expected results
- Test automation design
- Test approach
- Test cycle control sheet
- Test data & environment
- Mock conversion plan
- Test cycle control sheet
- Test conditions & expected results
- Test scripts
- Test closure memo

Deploy

- Deployment plan
- Contingency plan
- Go live preperation plan
- Go live support plan
- Migration approach
- Migration procedures
- Migrations verification scripts
- Migration request form
- Simulation evaluation
- Client sign-off
- Go live evaluation
- Handover documents

Figure 3 Documentation Requirements

4 Chronology of Events

Context

Prior to the implementation of EFI and DMI, between 2005 – 2013 SKAT performed debt collection using two distinct software systems. These systems had a very low level of automation.

Following the centralization of debt collection, SKAT established the EFI Programme to deliver a highly automated, unified debt collection System. The EFI and DMI Applications were the core elements of the overall solution. As described in Section 5.2, this report finds that EFI and DMI together should be considered a single System. This report covers the implementation of the EFI and DMI Applications and the overall System, which this report refers to as EFI+DMI.

There had been an earlier initiative to design DMI prior to 2008. The DMI project is shown as starting design in 2008, when the work started on the current live implementation of DMI.

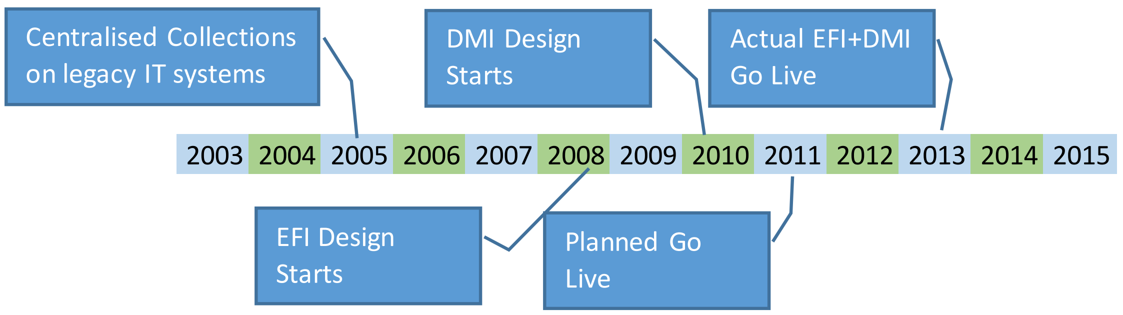


Figure 4 DMI Timeline

The diagram below depicts the timeline for the implementation of EFI and DMI.

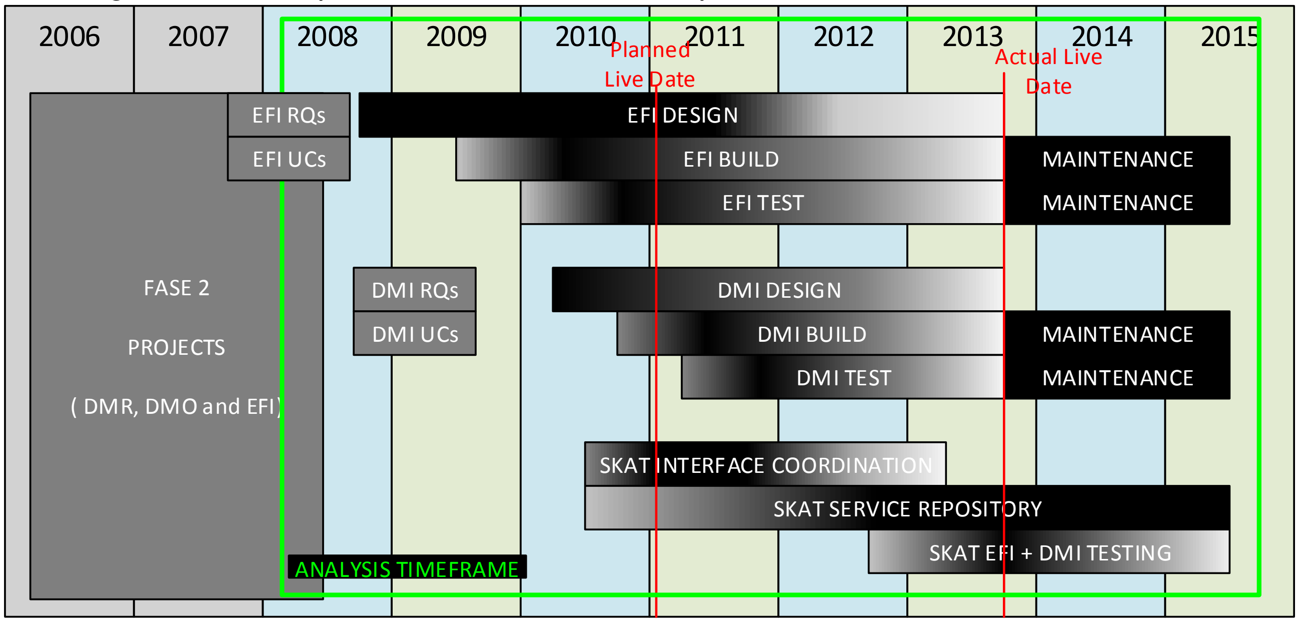


Figure 5 EFI and DMI Timeline

Note: Dates above are definite where there is documentation to support this. Approximate dates are depicted by shading and in the process of review and confirmation with the EFI Programme. There are a number of key points with regard to the overall timeline:

- The most significant is that the design of EFI and DMI was not performed in a tightly coordinated manner. Given the number and complexity of the dependencies between EFI and DMI, it would normally be expected that the design phases would be tightly aligned
- The EFI project started 70 weeks before DMI, even though the EFI project is heavily dependent on DMI
- Integrations between EFI and DMI were not prescribed. Instead, EFI and DMI projects worked to evolve an interface design, leading to extensive iterative changes to interfaces
- It appears that design activities continued throughout the course of project overlapping with the build and test phases. This introduces instability as a constantly changing design requires rework of the build and test work
- Overall, there was poor stage containment (completing and closing phases of work, before progressing to the next phase) both within the EFI and DMI Applications, and overall for the combined EFI+DMI System

In our opinion, the original project timeline was not feasible due to the technical dependencies. The iterative nature of resolving the dependencies resulted in additional work and suboptimal design.

5 Findings and Consequences

5.1 Requirements Trace

5.1.1 Purpose

IT systems are usually defined in terms of specific requirements that the system must provide, in order for the system to function correctly. For example, a requirement could state that every valid claim must have a "due date". Based on an initial review of the EFI Original Requirements and use cases, it appeared that these were at an unusually high level. Additionally, a brief defect root cause analysis illustrated that a relatively high proportion of defects and incidents had a root cause of being insufficiently defined in either the design, use cases or Original Requirements.

The purpose of our analysis therefore, was to gather and document requirements in detail (for selected areas) based on the Original Need and the intention for the EFI design during 2008-13, as expressed by the workshop participants. These Needs (detailed documented requirements) were then traced through the Original Requirements, use cases, functional and technical designs, source code and tests, to establish where the process for developing the Needs into a working System failed. The Needs were also traced to the Original Requirements, however, because the Original Requirements were very high level and primarily non-functional requirements (as distinct from functional requirements), a complete trace was not possible and we have documented this alongside the other traces in the Requirement Traceability Matrices (RTMs).

5.1.2 Key Findings

The following are our key finding from the requirement trace:

- We have not been able to identify detailed documented requirements for the EFI or DMI Applications or the combined EFI+DMI System despite a number of requests. Based on this absence of evidence, we conclude that there is no detailed documented definition of what EFI, DMI or EFI+DMI are required to do
- The Original Requirements and use cases for EFI and DMI did not contain sufficient detail to enable the System to be designed, built or tested, based on the trace of Needs through the identified documents. This is based on the results of the trace of the Needs through the Original Requirements and use cases
- Frequently requirements are initially created at high level, before being refined to a more detailed level, but we have been unable to identify any more detailed requirements than the Original Requirements
- Although additional detail was identified during the design phase and incorporated into the designs, this detail did not ensure that all of the Needs had been satisfied
  - When this additional detail was identified or implemented, the appropriate documentation was not always created and/or updated according using the SDLC approach which is designed to ensure traceability
- There is little documented traceability of Original Requirements or use cases in the designs, code or tests. The traceability that does exist is completely inadequate to verify whether and where requirements and the use cases have been implemented,

because tracing involves searching for names or use case numbers across the entire document library, which is not reliable, rather than a structured requirements traceability matrix

5.1.3 Assessment Details

The Needs (detailed documented requirements) gathering assessed a small proportion of the functionality in EFI+DMI. Although a precise measure is not possible, we estimate that the analysis has covered 5% of the total functionality within the System.

A total of 243 (Updated per 29.06.2015) Needs were documented across the analysis areas, through 3-4 workshops per area. This would suggest a detailed set of Needs for the entire System would comprise approximately 5,000 requirements, which based on our experience is comparable to similar collection systems. However even though the EFI Programme's original aspirations for EFI were for a highly ambitious System which was intended to do more than collections systems in other countries. The Original Requirements for EFI comprise approximately 400 Original Requirements (EFI 02 Leverandørens kravopfyldelse FA v1_00, EFI 02 Leverandørens kravopfyldelse S v1_00), with approximately 700 pages of use cases (EFI 01_02 Forr processer og akt beskrivelser v1_00). In our opinion, this indicates that EFI+DMI was significantly under-specified by these documents. Frequently requirements are initially created at high-level, before being refined to a more detailed level, but we have been unable to identify any more detailed requirements than the Original Requirements. These areas have been assessed through requirements gathering workshop and review sessions, the list of which can be found in Appendix 7.

In terms of our assessment, the scale used to assess trace completeness when tracing the Needs into the use cases, designs, code and test was as follows:

Match (out of 5)	Short Description	Description
5	Complete	Need is fully described
4	Substantially described	Need is well described, but minor details are unclear. An average designer / developer would require only minor clarifications to get this functionality fully implemented and working
3	Partially described	Need is partially described, but essential details are unclear. An average designer / developer would require significant clarifications to get this functionality fully implemented and working, but the unclear parts should fit within existing components and modules
2	Slightly described	There are references to the Need, but these are unclear. No designer / developer could implement this Need without major clarifications and/or structural modifications to the area
1	Absent	Need is not described at all

Table 3 Completeness Scale

The following table summarizes our findings from the requirements trace. The Needs gathered and documented in workshops with EFI Programme representatives were “traced” through the Original Requirements, use cases, designs, code and tests.

The average score core indicates the extent to which the Needs were identified as existing within the relevant artefacts for each phase of the Software Development Life Cycle. The Observations provides a brief interpretation of the score at each phase.
Any system that establishes and maintains requirements traceability, will consistently score “5” on the scale below at each phase.

Appendix 8 shows some examples of missing and incomplete Needs across all areas.

Phase	Modtag Fordring (Receive Claim)	Kundesaldi (Client Account Balance)	Betalings-ordning (Payment Plans)	Lønindeholdelse (Salary Deduction)	Avg. Score	Observations
Needs (Newly gathered detailed requirements)	86	85	22	50	243 (total)	There were a total of 243 Needs captured as part of the requirements workshops
Average match V Original Requirements	No match with Original Requirements. Original Requirements were primarily “non-functional” and of ~400 requirements ~50 were functional. No match was found.					No Needs were found in the Original Requirements.
Average match V original Use Cases	2.1	2.2	3.0	2.8	2.5	The Needs were slightly – partially described in original use cases.
Change Requests affecting the tracing	3	1	0	0	4 (total)	The 4 change requests described a total of 5 of the Needs to some extent.
Average match V Designs	3.2	4.5	4.1	2.9	3.7	The design specifications substantially describes

						the Needs. However, across all areas, Needs have been missed.
Average match V Code	3.1	N/A	4.8	4.5	4.1	The code substantially covers the Needs. However, across all areas, Needs have been missed. This score indicates that there is some evidence the function has been coded, not that it is working correctly.
Average match V Tests	2.2	3.4	3.3	2.9	3.0	The score of 3 indicates that on average the System has been partially tested successfully. To score “5” on testing, there must both be a test case, and the test case must be recorded as passed.

Table 4 Scoring

5.1.4 Accenture Assessment

During the EFI and DMI projects, no detailed set of testable Original Needs was created. Without this definition, it is our opinion that the subsequent phases of activity including design, build and test, which depend on these detailed documented requirements was seriously compromised.

As seen in the areas assessed, sufficient clarity was reached in some core areas through the design process to align needs and designs (see score of 4.5 for Client Account Balances designs), but in most areas this did not happen. The substantial lack of clarity about the Original Needs in the Original Requirements, use cases and specifications is the primary root cause as to why EFI+DMI does not fully meet SKAT's business needs.

As of July 2015, there is no detailed, documented and complete definition of what EFI, DMI or EFI+DMI is required to do. It is therefore our opinion that any report of the System being a certain percentage complete is unreliable, as there is no definition of what complete is.

5.2 End to end Software Development Life Cycle Review

5.2.1 Purpose

EFI and DMI are software applications. The process used to create these types of applications is termed a Software Development Life Cycle (SDLC). There are many possible variations to a SDLC.

However, the purpose of this section is to assess key aspects of the SDLC used to create the overall EFI + DMI System. Given the scale and complexity of what was required (as previously outlined), both in the functionality within each Application and the overall functionality of the combined System, certain disciplines are normally required for a successful outcome.

From a technical perspective:

- EFI and DMI are separate Applications. They are developed and deployed as completely separate technical components. They do not share source code, runtime platform or database.
- Together EFI and DMI are a single System. Neither EFI nor DMI can perform any useful Collections function without the other.

Creating a large system, composed of separate integrated applications is a viable approach to system development. However, successful delivery of such a system requires work to design, build and test the overall system, over and above the work required to create each application.

The following sections provide our findings and assessments in relation to important elements of the System SDLC, specifically:

- 5.3 Solution Blueprint Review
- 5.4 Interface Specifications
- 5.5 End to End Testing Approach
- 5.8 Complexity of Design
- 5.9 Solution Architecture Observations

5.2.2 Key Findings

Our key findings in relation to our review of the EFI + DMI SDLC are that

- EFI and DMI are separate applications, which are closely coupled and integrated, with extensive dependencies on each other. For example, screens that display the details of debts or interests require EFI and DMI to interact. It is our opinion they should therefore be considered as a single software System that should have requirements, processes, designs, and tests for the System.
- Although some System level design materials have been provided (see section 5.4 below) these fall far short of what would be expected to design a System of this complexity. Based on this lack of documentation it is our opinion that the System level "SDLC" activities were largely not performed.

5.2.3 Assessment Details

Following is the detailed assessment related to our findings, specifically our evidence view on the functionality and architecture:

Evidence of an EFI + DMI Software Development Life Cycle

The assessment team requested from the EFI Programme management, both verbally and in writing documents and materials related to the overall SDLC or the technical coordination of EFI and DMI. Although some System level design materials have been provided (see section 5.4 below) it is our experience these fall far short of what would be expected to successfully design a system of any complexity.

Assessment that EFI and DMI Applications Together Comprise a Single System

A Single Functional System

In a collections system, the core collections functions are tightly linked together including Claim Management, Auto Compliance Actions, Treatments, Payment Plans, Correspondence, Reconciliation, Contact Management and Debtor Management.

The EFI Programme's design for implementing these functions in most cases involves part of the function being performed in EFI and part in DMI. At each point where there is a transition from EFI to DMI and vice versa, interface(s) are required to enable the Applications to integrate and to provide the overall "end-to-end" function. Terms such as "cohesive functionality" and "tight coupling" are used to describe the fact that these functions are closely related. This can be seen in the following version of the functional map, which highlights that most functionality is partly in EFI and partly in DMI or that it does not exist in either Application. Where functionality is spread across EFI and DMI, the functional area is shaded in both EFI and DMI colours to denote this. All other systems are grouped under other. See Functional Report for additional information.

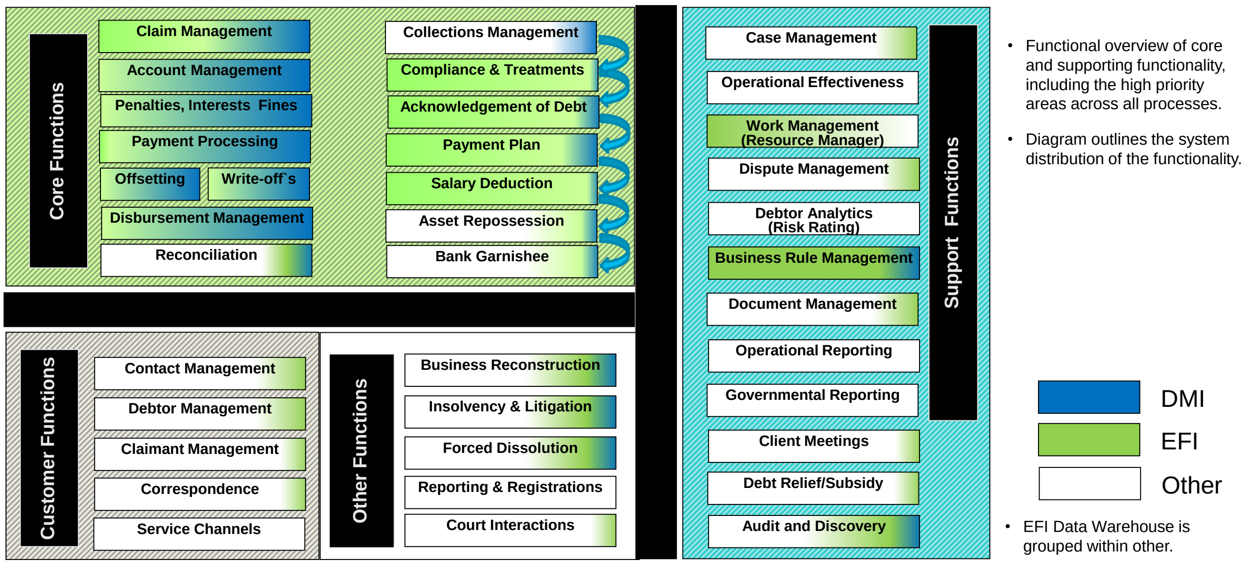


Figure 6 EFI+DMI Processes

<u>A Single Technical System Composed of 2 Separate Applications</u>
Our assessment has shown that technically EFI and DMI are completely separate Applications, built with different technologies (Java and SAP ABAP respectively). EFI and DMI have 297 web service interfaces in total; with 49 of these between EFI and DMI, (many of the other interfaces are internal to sub-applications within EFI). With this level of integration between the EFI and DMI Applications, in our opinion the overall EFI+DMI must be considered to be a single system, which requires technical coordination and end-to-end testing as a system. EFI screens and processes depend on, and cannot operate without, DMI data or logic, accessed via interfaces.

During our investigation, we found a diagram that was created by the EFI Programme test team that provides the only complete view of EFI+DMI interaction with other systems. Even though EFI+DMI are viewed as a system, it can be seen from this diagram that they have multiple links between them and numerous other systems, which they are dependent upon to operate.

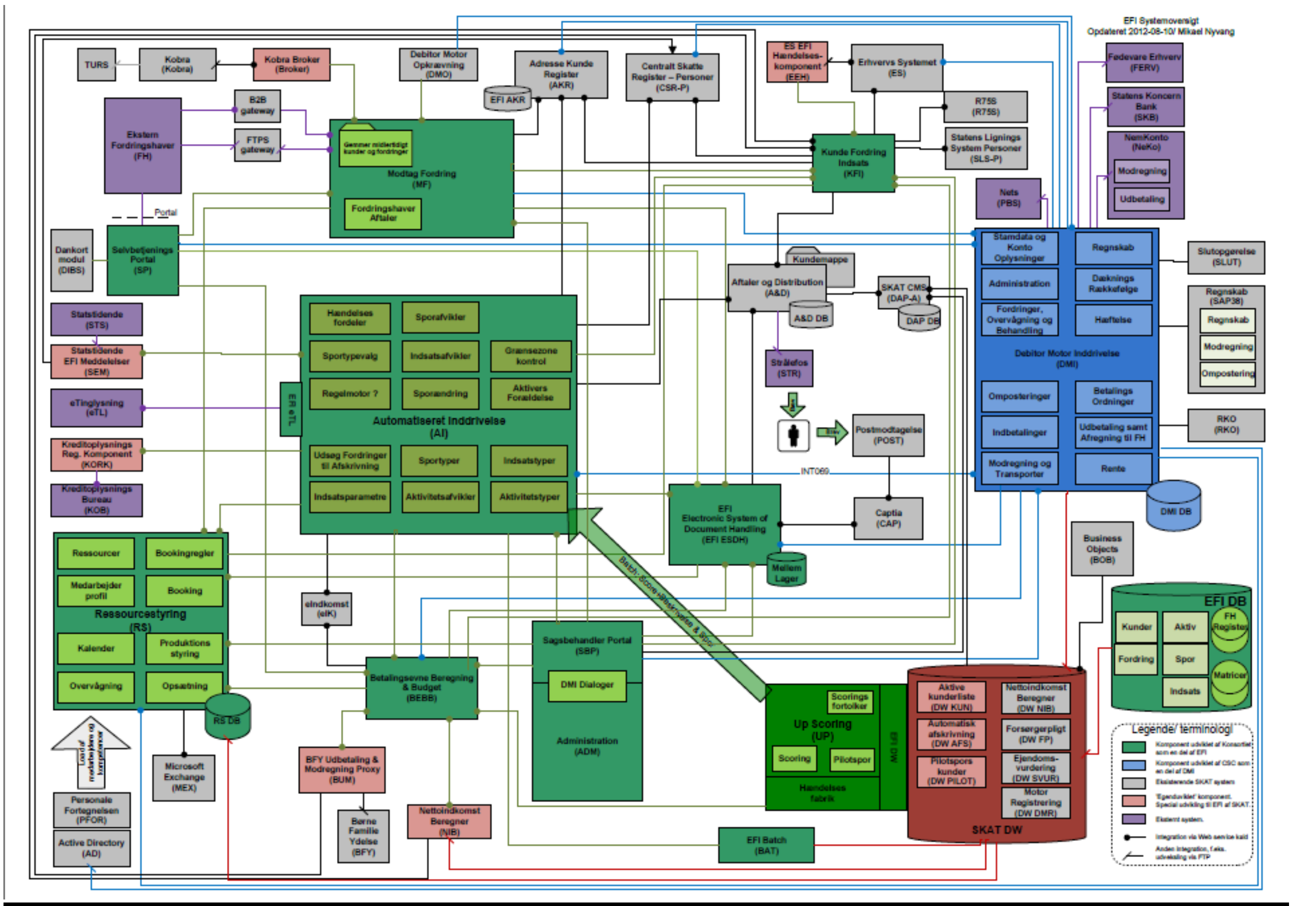


Figure 7 EFI+DMI Integrations

5.2.4 Accenture Assessment

Standard architectural advice is to implement cohesive functionality within a single application. If this approach is not taken and there is a tight coupling between the applications implementing the overall functionality, it is often not possible to make significant changes to any of the applications without impacting others. It is our finding that this is the case with EFI and DMI. We also found that it is currently not possible for the EFI Programme to test EFI or DMI without the other.

Our experience is that choosing to not implement functions in one system but to split them into separate applications has a number of consequences including the following:

- Within a single application, relational Integrity and ACID (Atomic, Consistent, Isolated and Durable) transactions can be used to enforce application wide data integrity with minimal cost, as the technical functionality to enforce this integrity is part of the database and application server platform. By comparison, with SOA style interfaces between applications, designers and developers must design, build, and test reliability and integrity for each interface. Additionally, there can be functional impacts from non-transactional interfaces requiring “compensating transactions” and other overheads to ensure end-to-end reliability and integrity. It is more complex to ensure reliability and consistency in a SOA than in a single application

- Within a single application, there is normally a single consistent data model. There is no requirement to interface or duplicate data from other systems, to meet the processing requirements of the application. Interfacing data, particularly in a fine-grained way as often happens with tightly coupled systems can introduce performance problems. Duplicating data introduces data synchronization problems (master data management) and other overheads.
- Project management of a single new IT application is complex. Project management of multiple, integrated IT applications is more complex. When IT applications are tightly coupled this becomes even more complex as correspondingly rigorous technical project management of the application developments is required. It is our opinion that the technical management and coordination of EFI and DMI projects was insufficient to deliver this rigorous coordination quickly, resulting in extensive ongoing change to align both Applications.

5.3 Solution Blueprint Review

5.3.1 Purpose

The purpose of the solution blueprint within an IT system is analogous to an architect's drawings for a complex building (e.g. a skyscraper). It provides specifications and guidance for each team on how their components integrate with the overall solution. Without this guidance, components will not integrate correctly and changes may be required. Following the analogy, it is cheaper to change the position of a window when it is on an architect's drawing than after the concrete has been poured and also there is less impact to the overall structure of the solution.

The purpose of the review was to assess the overall structure of the System as it existed in early 2015.

5.3.2 Key Findings

Following is our key finding of the solution blueprint review:

- There was no solution blueprint for EFI+DMI.
- During the design of EFI and DMI there was no documented end-to-end process description of how the overall solution should work to process claims from start to finish.
- The high-level designs that did exist (use cases) were separate for EFI and DMI. These were not integrated and they were also not used to validate that the end-to-end solution worked.
- Assessments and reports by neutral external experts between 2012-13 prior to the go-live date identified the lack of design as a concern. These concerns were communicated to senior EFI Programme representatives.

5.3.3 Assessment Details

As part of our analysis, we requested a solution blueprint or any other related high level descriptions of the system (e.g. Application Description, Technical Architecture

Description, Business Process Description) from the EFI Programme manager both in a workshop on 18th May 2015 and in subsequent emails. However, we were informed that no solution blueprint had been created. In fact, the EFI Programme manager questioned the need for a solution blueprint. This is documented in the “End to end design and coordination” meeting of 18th May 2015.

5.3.4 Accenture Assessment

As a result of a lack of an overall design, the integration of EFI and DMI, which was essential for either to function, was worked out during detailed design, build and test, requiring extensive change to both Applications, after design should have completed. The extensive change can be seen from the “change log” for the web services used for integration between EFI and DMI.

Without an overall design, extensive change and rework was required to many areas, particularly related to the integration of EFI and DMI, impacting the schedule and overall structure of the system.

The overall design is currently not documented adequately, impacting maintenance and change efforts.

5.4 Interface Specifications

5.4.1 Purpose

In an IT system, interfaces are used to connect different applications together. For example, EFI might require details from the CPR system and these details would be retrieved via an interface. Interfaces are known to be somewhat difficult areas within integrated systems and some of this difficulty is due to the precise agreement on technology and functionality that is required for between the systems for correct operation. Our experience has also shown us that changes to interfaces can be extremely disruptive and they have the potential to impact the Applications on both sides of the interface.

The main purpose of our review was to examine the interface specifications and to establish whether the level of detail and structure was sufficient to ensure a successful interactions between EFI+DMI and the systems they interact with.

5.4.2 Key Findings

Following are our key findings from this analysis:

- The interfaces between EFI and DMI and the systems they interact with including the ones within the rest of the SKAT were insufficiently designed. This led to a number of problems such as:
  - Extensive, ongoing change being required over a long period to fully evolve the interfaces
  - EFI and DMI Applications taking more time and effort to finalise
  - Making the testing of realistic paths through the code difficult due to being unable to create accurate program stubs. Stubs are normally used in large system implementations to test components or applications well before end-to-

end (complete system) testing is performed. Realistic stubs, that enable testing of more than the simple path depend on detailed designs

- There is evidence that some of the interfaces do not work reliably and/or deliver integrity. For example claims where data for the claim exists in EFI but not in DMI indicates a lack of reliability on an interface between the Applications. This was also confirmed by the EFI Programme manager in an interview. There is a lack of reconciliation reports across the System so the extent of this issue is unclear
- For most of these interfaces, there is little or no prospect of reusing the actual interface, or the interface design outside of EFI+DMI.
- When EFI or DMI requires replacement, there is a very large body of work required to re-implement the same (or new) interfaces again and build new components.

5.4.3 Assessment Details

Interface Specification
EFI and DMI have 297 web service interfaces in total, with 49 of these between EFI and DMI (many of the other interfaces are internal to sub-applications within EFI). Based on discussions with the EFI Programme manager on 23rd Feb 2015, it was understood that these interfaces (mostly web services) should create a "service oriented architecture" within SKAT, presumably to support future flexibility in collections. Based on our assessment of a sample of interfaces, we found that the documentation for each interface comprises:

- A brief description of the purpose of the interface. This is typically 3-4 lines of description
- Technical description of the interface (e.g. WSDL)
- Request and response data structures and attribute to value mappings
- In some cases (approximately 25% of a random sample) a list of the use cases in which the interface is used

In addition to this, we also found that the individual descriptions in System Architect and the services for each main area of EFI are briefly in the technical design (DDSB's) and that the general approach to using services and technical implementation details was to provide this information in the document Detaljeret delsystembeskrivelse for Teknisk Arkitektur i EFI-projektet. However, we could find no other DMI service documentation.

Therefore, it is our opinion that the interface specifications did not provide the detail necessary for rigorous testing of interfaces or services, and that therefore they were insufficiently specified.

The Services which have been spot checked in System Architect Viewer are the following:

Servicenames	Version
DOForventetIndbetalingOpret	1.3
DPDokumentHent	1.10

DWHæftelsesforholdTilAfskrivningMultiModtag	1.1
EFIBetalingEvneBFYModtag	1.3
EFIBetalingEvneKøretøjModtag	1.2
EFIETILBilHæftelserHent	1.0
EFIMatriceOpslag	1.0
EFINettoIndkomstÆndringHændelseModtag	1.4
EFIVirksomhedÆndringHændelseModtag	1.3
IAIndsatsBobehandlingHent	1.2
IASporOverblikHent	1.4
IMSporSkabelonGem	1.3
KFIAktivSlet	1.0
KFIFordringHent	2.19
KFIIndsatsAktivTilføj	1.0
KFIKundeStamoplysningerHent	1.4
MFFordringAfskrivUnderret	1.12
MFFordringReturner	1.3
MFFordringOpret	1.22
RSFindAftalerTilOmbooking	1.1
RSMedarbejderOrgEnhedArbejdsstedList	1.0
RSOpgaveHent	1.3
RSOpgavekøSøg	1.1
RSOrganisatoriskEnhedOpret	1.0
RSRessourceSlet	1.0
DMIBetalingOrdningList	1.7
DMICheckUdbetalingStatusListeModtag	1.2
DMIFordringForespørgBesvar	1.9
DMIFordringÆndr	1.14
DMIHæftelseForældelseList	1.11

Table 5 Services

We performed a spot check of 30 services in System Architect on the SKAT Sharepoint. The highest number of changes we found was 23, then we found 20, 15 and 13 respectively (refer to table above). Based our experience, we have found that a high rate of change such as this indicates an unstable and rapidly evolving design.

Given that each interface is the point at which two or more applications integrate, any interface change can impact both Applications which takes time and effort to do correctly. We have also found that without rigorous regression testing, interface change is risky and it is hard to predict the actual impact on the changes to both Applications. We also know from test analysis within PPSM that the EFI Programme did not and does not perform structured regression testing.

Service Interfaces in a Service Oriented Architecture (SOA)
Normally when we are developing a SOA system of similar scale, our service interface specifications typically describe the functionality available within the interface, the functionally valid combinations of data, details of the business errors that are possible (e.g. customer not found, insufficient funds) and details of technical errors that are possible (e.g. service timed out).

Additionally, a SOA is intended to provide flexibility through the reuse of functionality provided as services. For this reuse to occur, services must provide a function that is of use to multiple service consumers. The service interface needs to also be well designed and future proofed to avoid changes to the interface technical contracts. In our opinion, although the System uses technologies associated with SOA, EFI+DMI is not service oriented.

In a typical SOA, standardised approaches are defined and used for reliability, audit and other non-functional aspects of service interface behaviour. Although we did find standards for many of these in EFI/DMI, these cannot have been enforced in design, build and test because if they had been enforced, problems that currently exist in the System such as a lack of reliable behaviour across interfaces could not exist.

5.4.4 Accenture Assessment

Interface specifications need to provide sufficient detail to enable both sides of the interface contract build and test matching client and server components. This requires the detail and clarity to define the typical successful path and also all the failure paths. Using the specification, it should be possible to create stubs (software to simulate the other side of an interface for testing) to exercise all necessary paths through the code. The specifications in EFI+DMI did not include this detail.

It is our opinion that this lack of detail was a substantial root cause for the interfaces evolving over a long period (2010 – 2013). Changing interfaces are usually disruptive to a project, as they require changes to client, server and the test stubs and test cases for both. Typically our experience is that in comparable system development, there is an intensive effort to identify and specify interfaces at the beginning of a project (e.g. between months 3-12 of a 3 year project) with minimal changes required thereafter.

There are web service calls from EFI to DMI and DMI to EFI. In our opinion, calls in an SOA should be hierarchical, not circular. Given the number of changes to many “services” within the System, the low level of service reuse and the lack of a clear SOA structure it is our opinion that EFI+DMI does not provide an SOA.

5.5 End to End Testing Approach

5.5.1 Purpose

The purpose of this section is to;

- Describe the normal testing process in a SOA and/or distributed systems
- Compare how the EFI+DMI testing process against this

As noted in the findings in Section 5.2.2, this assessment considers EFI+DMI to be a single System composed of two integrated Applications, EFI and DMI.

Normally we refer to "End-to-end testing" as the process of testing an entire system from beginning to end. Whereas "Application" testing is the process of testing one element of the overall System in detail. The following examples from a supermarket illustrates the type of testing that is normally performed in each type of testing.

End-to-End Tests

This scenario tests a simple end-to-end scenario of a customer purchasing goods from a supermarket:

1. Customer enters supermarket
2. Customer selects items from shelves
3. Customer goes to checkout
4. Customer pays for goods
5. Customer leaves supermarket

Application Tests

This scenario tests the simple path and exception flows for the card payment application:

Simple path

1. Customer pays with card
2. Payment is received

Exception flows

1. Connection to bank is "down"
2. Card is declined
3. Card is stolen
4. PIN is entered incorrectly
5. PIN is entered incorrectly a third time

The Application tests can, and should, be performed with the application under test isolated from the overall System. It would be highly inefficient to perform all the "application tests" only as part of the "end-to-end tests". Taking this approach, each card payment application test would only be tested as part of the end-to-end tests.

5.5.2 Key Findings

Our key findings in relation to testing are:

- Analysis of the testing records confirmed with the EFI Programme test manager shows that 53.8% of the originally planned tests were executed. 52.3% of the original tests (80.5% of non-descoped tests at go-live) were recorded as passing testing. This resulted in a substantially untested System being put into production use
- There are missing isolated testing stages. For example: User Acceptance Test should be testing a system that has already passed UAT of the component applications. Instead, the EFI+DMI UAT attempts to test in a single phase of testing
    - The design of EFI (ODSB output)
    - The design of DMI (FGD output)
    - The System integration test of EFI + DMI Applications
    - The User Acceptance Test of the combined System
- The EFI Programme is unable to test either EFI or DMI in isolation. This introduces complexity in managing EFI and DMI as discrete projects or applications
- Assessments and reports by neutral external experts between 2012-13 prior to the go-live date identified the lack of a testing strategy to address key risks as a key concern. These concerns were communicated to senior EFI Programme representatives. It is not known what action, if any, was planned or completed based on these reports.

5.5.3 Assessment Details

<u>Testing Status at go-live</u>

The testing process analysis performed as part of the PPSM analysis identified that circa 60% of the original test cases had been executed. As part of the analysis for this report, some further analysis was performed on the same data from the EFI Programme QC system, provided by the EFI Programme Test Manager. The following tables summarise this analysis.

Amount of planned tests executed in any phase	
Originally planned tests	**22,519**
Executed test cases:	
V1.00	3,024
1.05 SKAT Fokuseret FKT	6,541
1.06 Dialog FKT	2,127
1.07 Fokuseret Bobehandling	421
Total tests executed in any phase	**12,113**
Percentage executed tests in any phase	**53.8%**

Table 6 Amount of planned tests executed in any phase

Original tests passing testing	
Originally planned tests	**22,519**
Tests passing	
V1.00	2,857
1.05 SKAT Fokuseret FKT	6,438
1.06 Dialog FKT	2,078
1.07 Fokuseret Bobehandling	394
Total passing testing in any phase	**11,767**
Percentage passing testing in any phase	**52.3%**

Table 7 Orignial tests passing testing

Non-descoped tests at go-live	
Test cases in scope at go-live	**11,073**
Tests passing	
1.05 SKAT Fokuseret FKT	6,438
1.06 Dialog FKT	2,078
1.07 Fokuseret Bobehandling	394
In scope tests passing testing	**8,910**
% tests passed in go-live scope	**80.5%**

Table 8 Non-descoped tests at go-live

Notes

- The figures above have been reviewed with the EFI Programme test manager. No corrections were identified as necessary.
- Normally regression tests are performed to validate that tests that previously passed continue to pass. This practice was not performed by the EFI Programme. As a result, the figures above may overstate the percentages passing testing, because without regression test it is not a safe assumption that tests passing in an earlier phase continue to pass testing.

Description of Normal End to End Testing Process

The "V Model" is a widely used model for defining the verification and validation processes for an IT system. The model is based on the fact that:

- The functional and technical requirements are the foundation for all design, build and testing activities.
- The requirements flow through successive phases (the "V" shape in black lines in the following diagrams).

- The flow is complete and correct and that there are verification and validation activities which ensure requirements have passed before they are move to another phase. This validation and verifications is normally achieved through;
  - The audit of requirements using a Requirements Traceability Matrix (RTM) or equivalent tool.
  - Testing against the specification of what should have been created. E.g. Assembly Test would test some or all of the Application against a corresponding design.

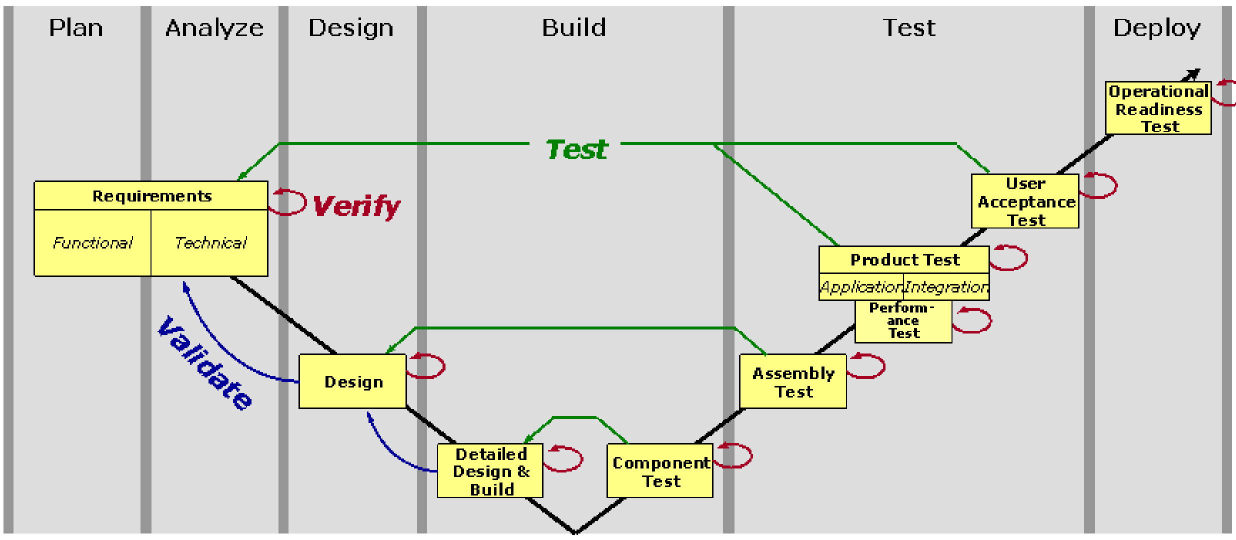


Figure 8 The V Model

In an SOA, or any highly distributed system, the “V Model” is also used. However, multiple “Vs” are required. This variant model is sometimes called a “W model”, “Dual Vee” model or “Many Vs” model.

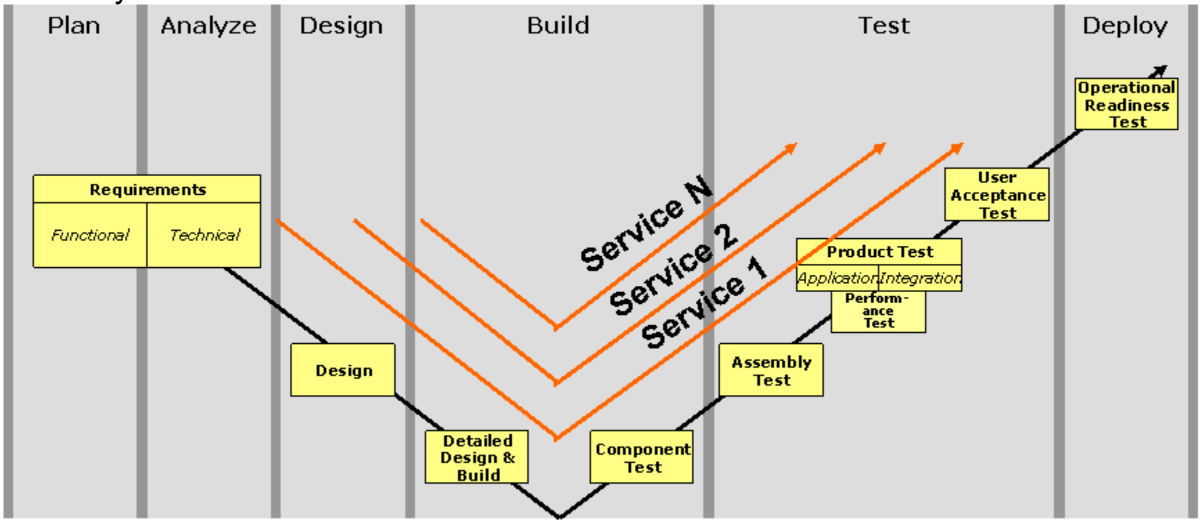


Figure 9 Many Vs Model

Comparison with Approach Used by the EFI Programme to Test EFI+DMI
The purpose of this model is to describe that each Service within the SOA should be designed, built and tested on its own (one V) before being tested "end-to-end" (another V). In the case of EFI and DMI, there should have been a thorough test of the individual Applications against documented requirements, before they were tested together, as experience has shown that testing Applications individually, prior to integration, is more effective. The reason it is important to perform testing of individual services or Applications before testing the complete System is that it is better to find errors earlier than later, where the costs of repairing these error is larger due to the time taken to find them.

Our assessment found that the EFI Programme is presently unable to test EFI or DMI in isolation and that this limitation is caused by the following main reasons:

- There is insufficient documentation to describe what each interface should do to enable it to be tested standalone.
- The EFI Programme testing team does not perform standalone testing of EFI or DMI. The analysis has not definitively determined why, although this approach was considered but not actioned.
- As the overall System is excessively coupled, there are a far larger than normal number of interfaces between EFI and DMI

As a consequence of the above, the EFI Programme performs all testing of EFI on the combined EFI+DMI System. This has the following consequences:

- All EFI Programme testing requires a more complex environment. As of April 2015, there is only a single usable environment for testing and this environment cannot be used for testing the most complex cases. The EFI Programme has estimated a cost of DKK 30-40M to create a Pre-Production environment for testing.
- It is more difficult to state whether EFI or DMI, as a singular Application on its own, is working. This is because there is no way for the EFI Programme to test either Application on its own. When an error is detected, it may be unclear whether this is caused by EFI, DMI, an interface between them, or some other cause.

Note: the testing in the above refers to EFI Programme (System) level testing. It is understood that the development teams for EFI and DMI do test the Applications standalone using stubs, however, as noted in section 5.4, as the interfaces are not specified in sufficient detail, this testing frequently fails to detect issues that surface later in integrated testing. Although test reports were requested from the EFI Application project team, no evidence of the outcome of unit tests was provided (although these clearly exist). No evidence (test approaches, acceptance criteria or test completion reports) was provided of the standalone Application testing.

The following diagram illustrates at a summary level the points above. It depicts:

- The overall "System V", with the "end-to-end" requirements, processes and overall design not documented
- Separate "EFI V" and "DMI V", showing the activities performed.

- The only User Acceptance Test is performed on the overall integrated System. There is no UAT of EFI or DMI alone.
- The EFI and DMI Original Requirements and use-cases were not tested. They were not used systematically as the basis of test conditions for UAT.
- The most structured attempt to test the System was based on using the EFI and DMI functional designs (ODSBs and FGDs) as the basis for creating test conditions. These did not provide an end-to-end specification of required behaviour as would be required for a successful UAT.

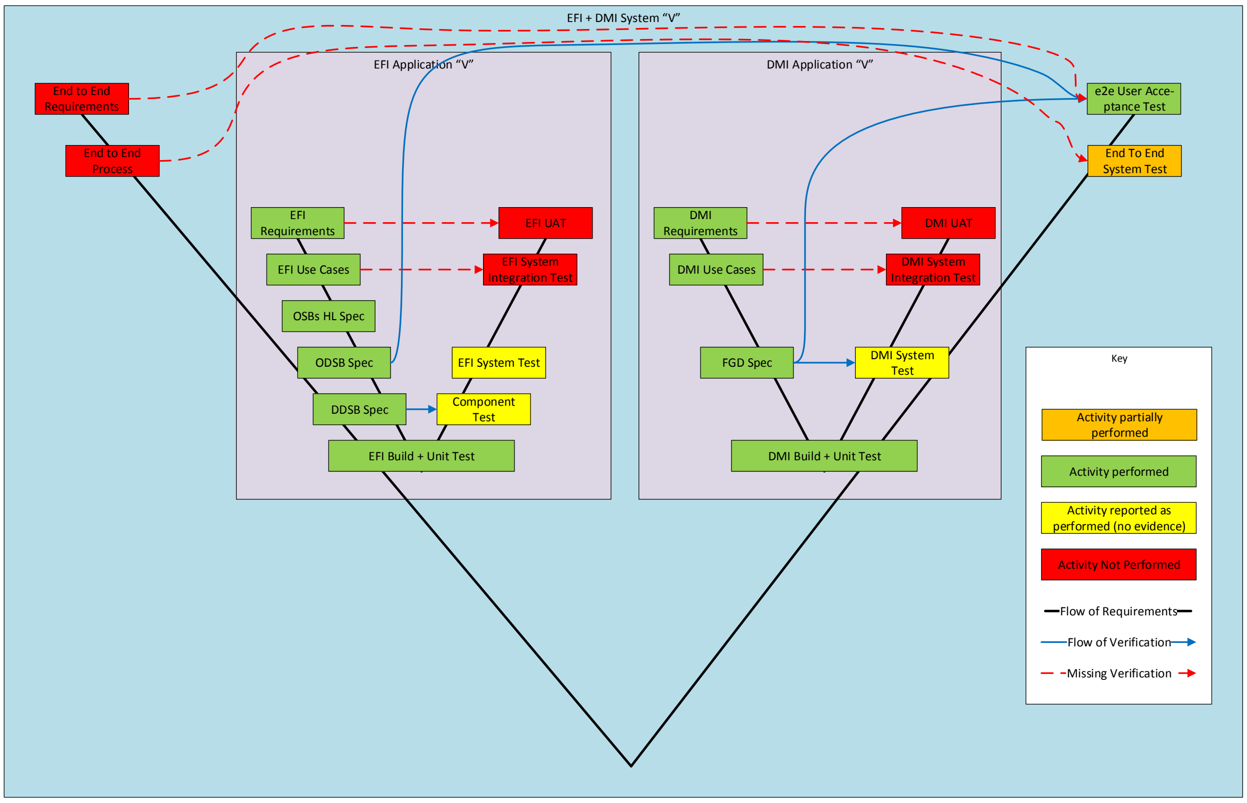


Figure 10 EFI+DMI V Model

5.5.4 Accenture Assessment

- Because the EFI Programme does not test the Application before testing the System the EFI Programme is unable to adequately perform stage containment, or to determine whether the Application is working before combining it into the overall System. This reduces test efficiency and increases the duration of testing, as well as making attribution of responsibility for defects more difficult.
- Because there are no testing phases between the tests performed by the development teams and the UAT, the UAT scope has to cover the entire breadth of the EFI+DMI System to a very detailed level (i.e. all the tests above). This is highly inefficient, taking more duration and resulting in lower quality than a phased testing approach.

- Remediating the EFI and DMI testing to a more normal approach would require
  - System (end-to-end) requirements and designs, and matching test conditions
  - Application requirements and designs that are sufficiently detailed to test against, with matching test conditions.
  - Multiple additional test environments, some of which would be integrated end-to-end
  - Additional stubs implementing the functional complexity to allow thorough testing
  - Executing all the tests until all required tests pass successfully

Performing the tasks above is equivalent to a rebuild.

5.6 EFI Code Review

5.6.1 Purpose

The purpose of a code review is to examine a body of source code and check it for good coding practices. The assessment included an automated code review of the entire EFI code base and a manual code review of a sample of areas.

An automated code review will reliably detect many problems at a source code level. This can provide useful insight to the maintainability of the source code, to the consistency of internal documentation (comments) and the complexity and to some extent the structure of the code. Most organisations now incorporate automated code reviews into their SDLC. The manual code review provides additional review and insight.

However, any code review cannot determine whether the source code performs the correct business function. Verification that the code performs the required tasks depends on a definition of what is required and testing against that definition.

5.6.2 Key Findings

Our key findings in relation to the EFI code review are:

Automated Code Review

- Overall quality of the EFI (Java) code is average to good. There is no evidence that technical code quality, on its own, is a significant cause of EFI issues.
- There are commenting and cohesion issues that will impact maintainability. More maintainable code is easier, safer and less costly to modify and test over the lifetime of the System.

Manual Code Review

- Overall, the manual code review supports the automated review findings that the code is of average quality. Although there are issues in the code, these alone are not a major cause of problems with EFI.
- The maintainability of the code base is somewhat below average for a code base of this scale. This will increase the lifetime cost of changes, and increase the time taken to make changes.

5.6.3 Assessment Details

Automated Code Review Assessment Details

The automated code review analysed the entire EFI V2.80 code base (March 2015), and dependencies. This analysis evaluated 1,403,693 lines of code (964,076 effective lines) across 11,667 source code files and 1,643 packages.

It assesses the EFI code against a number of technical focus areas. In each focus area, better code will be more maintainable. More maintainable code is easier, safer, and less costly to modify and test over the lifetime of the System. The assessment also provides a comparison of the EFI code base against similarly sized code bases from similar organisations.

The automated code review also provides input to the second half of the code review, which is performed manually, where the high-level structure and architecture of the code is examined.

Area	Fact	Evidence
Overall EFI Technical Code Quality	In summary, on most measures, by comparison to similar code bases, the code is either of good or average quality.	The proportion of issues in the code, checked with a number of separate tools all indicate that while the code base has many issues that could be improved, the density of these issues in the code base is similar to or better than average code bases of this size.
EFI Code Complexity	The EFI source code is large in code quantity but the level of complexity is unusually low for a system of this magnitude.	The automated code analysis shows that a mere 0.17% of all the files, have a high level of complexity. This is at least an order of magnitude less than comparable solutions, and is surprising given the complexity of the Needs. This could mean a number of things including: a) The EFI project has been very careful to avoid writing complex code, and to separate logic out into separate classes and methods – although the cohesion level could indicate otherwise (see next) b) There is less complexity in the business logic for EFI than in comparable systems c) Where many systems have complex functionality due to large volumes of validation or business

Area	Fact	Evidence
		rules, which adds to the average complexity, a similar volume of logic is not present in EFI One hypothesis would be that only a portion of the business logic actually required for a fully working EFI solution is actually written and in the code base, thereby reducing the average complexity.
EFI Code Maintainability	Systems such as EFI, which have an expected life span of 15-20 years, must be maintainable in order to support the evolution of the system. There are a number of issues within the code base that will impact this maintainability and make it more difficult to make required changes to the code and familiarise new developers with the code. This analysis has identified initiatives which can be taken in order to improve the maintainability of the EFI code.	**Commenting:** Commenting within code makes it much easier to understand and maintain - especially for developers new to the application or part of the application. In EFI the comment ratio is low - only 30% of files have a good level of commenting, which is contrary to best practices. Commenting should also be used to link requirements, specifications and code, making maintenance easier by enabling clear traceability from requirement to code. **Unused code:** Unused code within an application is source code that is never executed by the application. Unused code can be confusing and time consuming to understand thereby wasting developer time in understanding the code. Our analysis shows that approximately 20% of methods (which represent functionality / what the code does) are not used and should be removed. This measure is approximate as there are some circumstances in which code may be assessed as unused incorrectly (e.g. sections of dependencies or frameworks that are not used). **Cohesion:**

Area	Fact	Evidence
		Cohesion is a measure of how well functionality is split up in an application. Ideally developers separate different core functionalities into different packages and files, making it easier to update or replace functionality without affecting other functionality. It also makes the code easier to understand. Our analysis identified a lack of cohesion which could be problematic with regards to maintainability. Consequences of changes to parts of EFI can be difficult to predict where cohesion is low, increasing the likelihood of adverse effects.

Table 9 Code Review Areas

Manual Code Review Assessment Details

The manual code review examined a number of areas, including the code corresponding to the sample areas investigated in Section 5, and some areas highlighted by the automated code review.

The review approach was to manually inspect the code, and review the code considering a number of focus areas including overall structure, control flow, traceability to requirements and designs, error handling, reliability, interfaces, and some other technical aspects.
The manual assessment examined the files and classes itemized in 0.

Structure and Traceability

The code base has a modular structure, and as noted in the automated review, in most cases, the level of complexity is low.

Typically, with a large and complex system, there is a high level of traceability from functional designs to code with the traceability demonstrated by including notes in the specifications and comments in the code to cross-reference each. There is a limited level of such traceability within the EFI design (ODSBs, DDSBs) and code base. This does not provide a complete and reliable linkage such that it is always clear where a design was implemented in code and vice versa,

Normally, in the construction of similar scale systems, the validation that each requirement is implemented in code is performed by completing the requirements traceability matrix to ensure all requirements are implemented. This was not performed for the EFI code base.

There is some traceability for defect fixes and changes to the code. In some modules, there is a high proportion of the code that is commented as “EFIDEFECT” indicating that a change was made to the source code at this point to correct a defect or make a change. In portions of the code, it appears that almost half the code has been changed as a result of defects.

Although it is usually possible to trace the structure of the code, and identify that a requirement has been at least partially implemented in a location, without the traceability, it is not possible to do the reverse. That is, it is not possible to be confident that all the logic to do with some functionality has been located.

Control Flow

The flow of control is the process by which the overall execution of the system is controlled. In the majority of the system, it is straightforward to understand the control flow. However, there are locations in which this is not the case. For example, the orchestration of treatments is controlled by CSV files, in effect, creating a simple process definition language. Many complex systems include some level of unique customisation like this. Although this is usual, without detailed documentation on the detailed approach used, this can make it more complex to maintain, as new development staff face a learning curve to understand system specific approaches.

Error Handling and Logging

Error handling refers to the way that unexpected events are managed within the code. Unexpected events include incorrect data being encountered, non-availability of interfaces and functional errors such as a logical error resulting from an unexpected condition.

The error handling strategy varies between classes, and also between the main components of EFI. In some parts, the error handling is rigorous, in others less so.

The logging strategy also varies. There is less logging than be expected in critical areas of the code, e.g. in persistence attempts and Web service calls. Finding the root cause of erroneous behaviour may be difficult in parts of EFI.

There are places in the code (e.g. for Treatments) where method calls will fail silently – with regards to exception handling, logging and method return value. This will also make it difficult to identify erroneous application behaviour. In the spot checks there is a clear tendency to neglect code commenting – even on the class level. This makes it much more time consuming than necessary to understand the structure of the application, and more difficult to maintain.

Interfaces and Reliability

EFI makes extensive use of web services to integrate discrete components of EFI into the overall application. Web services technologies (SOAP, WSDL) alone do not enable reliable behaviour and coordination of work between components. The designer and developer must design the solution to ensure reliable operation in all circumstances.

A number of standard approaches have been defined, based on the standard approach of idempotence. It is necessary for the designer and developer to ensure that in each case the business logic is fitted to the correct approach to ensure reliable operation. There are cases in the code where it is noted a non-reliable approach is taken. There are also production incidents reported related to lack of reliability across interfaces.

5.6.4 Accenture Assessment

It is our opinion that the manual and automated code reviews supports the hypothesis that essential business logic is missing from the actual live code base. Code is missing in the areas of exception processing, validation, logging and other areas. In some cases, this is due to errors in coding process. However, in many cases, it is our opinion that the focus by the EFI Programme during design and testing on focussing on the "simple path" will have allowed this missing code (and therefore functionality) to have remained undetected until the code encountered real world data in production, triggering a failure.

5.7 DMI Code Review

5.7.1 Purpose

The purpose of the DMI Code Review is the same as the EFI Code Review, namely to provide insight to the code quality and the structure of the Application.

5.7.2 Key Findings

Our key findings in relation to the DMI code review are:

- The DMI custom code with an 8.8 findings in every 100 lines of code is rated as: Slightly Below Average.
- DMI uses a low proportion of SAP package functionality.
  - Core SAP concepts have been re-implemented in custom code. This prevents DMI from using large sections of SAP functionality now or in the future without a complete restructure and rewrite of DMI.
  - Core SAP product functionality has not been used, and instead has been custom coded in DMI ABAP code, EFI Java code or elsewhere. In many cases (e.g. Mass processing, Locks, CRM) this has delivered far less functionality, at higher cost with reduced maintainability.
- DMI is the financial engine at the core of the System. Unusually for a financial application it does not enforce Referential Integrity. This has allowed DMI to store invalid records. This reduces the data quality in DMI and the overall integrity of financial processing in SKAT Collections.

5.7.3 Assessment Details

<u>Approach: Functional Review</u>

The assessment consisted of an overarching functional review alongside a technical ABAP Code Review. The assessment was completed with information from workshops with project and EFI Programme resources, documentation review and access to a DMI Sandbox environment. The assessment was completed in the current DMI landscape with the following
characteristics:

- SAP ECC 6.0 - Enhancement Package 4
  - Support Pack (FI-CA and IS-PS-CA) – level 8

- SAP PI 7.1

<u>Low Usage of SAP Package Functionality</u>
DMI represents an unusual approach to an SAP Revenue implementation. There is a low usage of SAP package functionality (standard objects, programs and mass processes) combined with an unusually high percentage of custom objects (Programs and Tables).

The consequences of this are that the licensed SAP product functionality is relatively lightly used and the Application is generally harder to understand for SAP professionals, as DMI is in fact, mostly a custom Application, rather than an SAP application.

Because DMI makes light use of SAP product functionality, SKAT will benefit little from future upgrades of the SAP product as DMI makes little use of this functionality. For example, although SAP has screens, reports and other functionality, the SKAT users do not actually use this.

Because DMI is essentially a custom coded Application in ABAP, it cannot be maintained by an SAP professional, without them having to learn the internal structure of DMI (not SAP PSCD). As the level of documentation that would be required to understand this structure does not exist (or has not been revealed to the analysis team), this means that currently DMI can only be maintained or altered by the original DMI team.

The following are the major conceptual differences between DMI and typical implementations of SAP PSCD.

- Business Partner
  - The concept of unique taxpayer is housed outside of SAP. This renders address, bank details and other standard SAP Business Partner objects useless.
- Contract Object (CO)
  - This essential object within the SAP Tax and Revenue Management (TRM) master data model is missing in the DMI Application. This is commonly used for revenue clients while the CO concept is rarely used in FI-CA implementations for other industries (e.g. utilities).
  - The Contract Object concept allows another layer of data segregation, it also allows the usage of Inbound Correspondence and FACTs functionality.

DMI is unusual compared to other implementations mainly because the lack of usage of SAP standard functionality and mass processing.

- <u>Use of custom code</u> - Documentation states that DMI is utilizing at least 90% custom code, based on our experience this is a rare occurrence within SAP implementations.
- <u>Standard functionality & objects</u> – DMI does not utilize standard tables or programs to manage certain core processes, for example: Locks, Clearing, Dunning, Bank

Returns, Instalment Plans, and Promise to Pays. This not only excludes the usage of tables, but also the delivered function modules that can be used for: extraction, history and checks. DMI also opted out of utilizing information containers like Correspondence History to store interactions with the taxpayer. The net result of this is that DMI has custom coded entire functionality instead of extending or aligning to what already exists in the product, and taken architectural decisions that prevent use of this product functionality without a migration process.

- Mass processing – SAP PSCD is built around the concept of Batch Mass Processing. This allows the application to process the large amount of information in a parallel and organized fashion. The DMI Application is only using two mass processes: Payment Run and Simple Revenue Distribution. The following are examples of mass processes that are commonly used, on a daily basis, across other TRM implementations: Clearing, Billing and Invoicing (Object and Return), Dunning, Write-off, Interest run, External Collection Agency Processing, Correspondence print.
- End user access to SAP Screens – End users are not allowed to access SAP screens. End-users access to SAP is via a layer of custom objects that could have been avoided by adapting to standard screens where possible.
- It is outside the scope of this review to consider all the requirements and design decisions in DMI, and their fit within SAP Product. Based on the sample subset of Needs reviewed, there was potential to use more of the SAP functionality than was actually used.

The following processes normally fall outside the SAP PSCD standard functionality. It is our experience that custom extensions are required to fulfil common requirements in these areas and as such it was inevitable for DMI to add custom code. Nevertheless extensions in these areas should be limited in scope to avoid usage decline on SAP processes. For example in the case of Contingent Liabilities, the extensions contributed to an increased usage of custom code in other areas.

- External offsets – SAP was not built around the concept of temporary liabilities which is required to fulfil the requirements of external offsets. Solutions normally entitle custom table(s) to store the temporary liabilities and history information, along with custom program(s) that enable the matching logic. Once the match is identified SAP Standard functionality can potentially be used to continue the process.
- Contingent liabilities – While the concept of additional payee(s) is part of SAP, it normally does not comply with the complex processing rules within Revenue agencies. Based on our experience shared liability models normally require custom objects to track and operationalize the process. Similar to external offsets, the custom objects can be used to help SAP standard functionality to take over the transactions.
- Interest calculation – Monthly (or regular) calculation of interest in SAP can create an overwhelming number of postings for Revenue agencies, while the standard functionality (calculation and history) should be used, the triggering of interest is

normally adjusted to only calculate at certain times requiring some level of custom code (e.g. only calculate interest on payment receipt or forecast).

Detail of Documentation

- The existing documentation does not provide information on individual custom objects. Instead, the documentation was built around packages and broadly defines objectives, which complicates the understanding and the integration between the programs, function modules and tables.
- The web service (a series of objects which serve as the communication method between DMI and EFI) definition relays only on short descriptions to explain the functionality. The lack of detail creates scope definition problems and testing challenges.

Requirements Reassessment – Mapping to SAP PSCD

The purpose of this exercise was to examine actual DMI Needs, and examine the extent to which implementation of these Needs should fit within the SAP PSCD product. This was to compare with the actual implementation of DMI, which is 80-90% custom ABAP (i.e. outside the SAP PSCD product).

The 96 Needs that were available as of May 18th 2015 (independently of the completion status) were reviewed and compared against SAP PSCD functionality. The Needs are the same Needs used in section 5.1, and are a sample of Needs in the core areas of the System. These come from 3 categories:

- Receiving Claims
- Calculating Account Balance
- Setting up Payment Plans

Requirement Category	Custom Object	Event	Standard
Receiving Claims	8	20	7
Account Balance	15	5	31
Payment Plans	3	5	2
Total	26 (27%)	30 (31%)	40 (41%)

Table 10 Requirement Categories

An event is used to enhance standard SAP functionality beyond configuration. It enables customers to use core processes while adding specific business rules. A custom object refers to a customer specific item that is needed to complete a business process. For example an interface, report, correspondence or extension.

The analysis above shows that approximately 70% of Needs could be covered by standard SAP processes (including configuration and events). As implemented, DMI implements approximately 10% of Needs using standard functionality. This review strongly suggests that standard SAP TRM functionality could have been used more extensively within DMI.

Caveat: Complexity of implementation is increased when requirements are reviewed jointly, as this exercise only analysed a limited amount of requirements, these percentages should be only indicative for DMI. A reasonable expectation is that the *Custom Object* percentage could rise by up to 10%.

Maintainability of DMI
Based on the analysis of the SAP structure, documentation and functionality the following are reasonable expectations for maintaining DMI:

- Functional upgrades (mainly delivered by enhancement packages):
  - The effort to perform upgrades should be lower than a traditional SAP implementation, as most of the objects will be untouched by SAP's enhancements, as a consequence testing scope should be reduced.
  - Based on the small amount of standard SAP code that is being used, functional upgrades and additional SAP functionality will have little or no advantages for the EFI Programme going forward, as standard SAP product functionality mostly isn't used.
- Support packages (SP)
  - The EFI Programme should continue to keep the SP level in the recommended latest -1 (minus one) version. This will allow prompt support from SAP. SP levels on DMI have not been updated in 3 years. The latest SAP SP level is 15, leaving DMI at least 6 levels behind, which is atypical for SAP customers.
- General maintenance
  - Updates or fixes to DMI will most likely need to be performed by a third party who understands the complex integration of custom objects.
  - Lack of functional documentation means that maintenance of the Application will need to start at the code-level and move up, increasing the cost of ownership.
  - Within DMI there is an added layer of services to transform XML that will need to be maintained, given the documentation status, this will probably open the door to ongoing scope discussions.

Lack of Referential Integrity
Referential Integrity is a database level technology that can be used to strictly enforce relationships between data, and ensure data complies with business rules. This helps ensure the integrity of business records. For example, a rule can be created to require a Liability to be related to a Claim, and the database will prevent the creation of a Liability that does not belong to a Claim. It will simply not be possible to create records that break the relational integrity rules. Referential Integrity provides a second line of defence, above application validation logic, to prevent Application errors resulting in corrupt data. Referential Integrity is almost invariably used within enterprise systems to enforce data integrity rules and maintain the quality of data in enterprise systems.

The assessment has not had technical administrative access to the production DMI Application. However, based both on interviews with DMI technical staff and analysis of the DMI data transfers to the SKAT Data Warehouse, it has been identified that DMI does not have referential integrity enabled on key data relationships. This is highly unusual, as the net effect of this is to allow the Application to store records that break business rules (i.e. demonstrably incorrect business records). There are rare situations where RI may be partially disabled, to add a modest increment in performance, but there is no indication that this was required in this case.

There are documented data integrity issues within DMI data in the SKAT Data Warehouse.

Approach: DMI Semi-Automated Code Review
The review was performed using a proven Code Review tool for SAP (ABAP) code. A random sample of programs was selected. In total, the 78 selected programs comprised 11.5% of the 675 custom programs in the environment. The sample include 54 main programs and 24 include programs. These are listed in Section 9.2.8.

The automated tool was executed on the sample, and the findings were reviewed by an experienced ABAP developer. The findings were categorised by fix priority and change complexity to enable the impact of these issues to be properly understood.

Findings
The tool logged 1,624 findings in the 21,209 lines of code reviewed, or 8.8 findings in every 100 lines of code.

The following table shows the count and percentage of findings distributed by type, showing the main concentration of findings within hardcoding and Incorrect Logic categories.

	Hard coding	Incorrect Logic	Performa nce Issue	Security	Code Inspector	Maintain ability	Modulariz ation	Related Object	Total
Count	547	471	258	170	138	25	14	1	1,624
%	34%	29%	16%	10%	9%	2%	1%	0%	100%

Table 11 DMI Code Review Findings by Category

The following table shows the distribution of findings based on priority and complexity. Fix priority describes the importance of addressing the findings, while Change complexity describes the effort that will take to address them.

	Fix Priority				Change Complexity			
	High	Medium	Low	Total	High	Medium	Low	Total
Count	225	1,136	263	1,624	16	250	1,358	1,624
%	14%	70%	16%	100%	1%	15%	84%	100%

Table 12 Distribution of Findings Based on Priority

The results suggests a priority to review the code with 84% of findings in the High/Medium category, but it also suggests that the risk could be addressed with minor effort as 84% of the findings can be addressed quickly.

The results were then compared to other applications utilizing the same tool in order to provide a comparative assessment to the EFI Programme. The DMI custom code with an 8.8 findings in every 100 lines of code is rated as slightly below average.

5.7.4 Accenture Assessment

- The design of DMI and the lack of detailed current documentation means that DMI can only be maintained by the current development team. Because DMI does not use standard SAP approaches, it is not maintainable by SAP professionals without DMI experience or an extensive knowledge transfer
- DMI will benefit relatively little from future SAP upgrades, as it makes relatively little use of SAP functionality
- Without referential integrity, DMI is able to store logically incorrect business records. Data analysis (see Section 5.12) confirms that incorrect business records exist in DMI
- The analysis shows that approximately 70% of Needs could be covered by standard SAP processes (including configuration and events). As implemented, DMI implements approximately 10% of Needs using standard functionality. This review strongly suggests that standard SAP TRM functionality could have been used more extensively within DMI

5.8 Complexity of Design

5.8.1 Purpose

The purpose of the design complexity assessment was to understand the original aspirations for EFI+DMI, and compare it to similar systems. The analysis team has experience of state level collections systems serving similar and larger numbers of citizens. It is useful to compare EFI+DMI System to these other systems to:

- See if there are differences between the EFI+DMI System and other systems that are successfully in operation
- Form a rough assessment of the comparative complexity of the System. This can provide insights into the indicative effort required to create the System

5.8.2 Key Findings

Our key findings in relation to the overall design complexity are:

- It is our opinion that the EFI Programme's ambition for the EFI + DMI collections System was higher than comparable systems across many aspects, as assessed below.
- The scope of the System was not well defined. The complexity of the needs was not well described in the Original Requirements and use cases, or at any subsequent

phase. It is our opinion that the complexity of implementing the variety of debts and legal persons, combined with the extent of automation was never fully understood. This resulted in essential functionality being omitted from the System.

- IT requirements are usually the first time that new policy has to be worked through to its logical conclusions in black and white instruction and this often generates legislative requirements. In our experience with major public sector IT system development in many European countries, a track of legal change frequently accompanies the IT change, to limit /simplify the complexity of requirements by changing the legislation.

5.8.3 Assessment Details

Aspect	EFI Programme Approach	Typical Approach
Range of debts collected	A single System for collecting all debts across the public sector. However, it is clear that there many dimensions of complexity which must be handled • 490 claim types (280 in use) • 697 claimants • 12 types of legal person	Most countries currently operate separate public debt collection systems at an agency level. Within Revenue Agencies, there are frequently 2-3 separate debt collection systems for different classes of debts. EFI attempted to collect a wider range of debts, and from a larger number of claim owners than comparable organisations. The Norwegian National Collection Agency (NCA) performs only part of the function that EFI was intended to perform and is a dedicated agency created for this function. It collects on 188 claim types (less than EFI) and has 35 claimants (much less than EFI). NCA does not collect the claims for the largest and most complex government organizations. The functional complexity and the volume that NCA handles cannot be compared to the EFI/DMI scope.

Aspect	EFI Programme Approach	Typical Approach
Level of Automation	The EFI Programme attempted to automate a wide range of activities in the initial scope of the System. This included low frequency, high value, high complexity debt collection tasks (e.g. insolvency). It also included complex automated solutions, where a less sophisticated semi-automated solution would be adequate. E.g. EFI attempted to automate booking the time of the debt collector, room bookings and resource (e.g. vehicle bookings). However, the result of the automation does not work – e.g. appointments are scheduled too close together in time, without an understanding of locations and travel time. This is likely because the needs complexity of full automation was too high, and the actual implementation too simple. Note: the completion status of the resource management module is currently in dispute.	Similar systems typically balance the level of automation and frequency and value of the tasks being automated. Automation is usually focused on high frequency, low value tasks, low-medium complexity tasks. Other solutions typically include mechanisms to (a) take work aside for manual processing where required AND (b) to re-introduce the work into the automated flow when the manual exception has been resolved. Most organisations (public and private sector) do not attempt the comprehensive level of automated resource management that EFI attempts. Instead, simpler mechanisms (e.g. push and pull queues, with some matching of work complexity against worker skills) are frequently used. Where a complex resource management system is required, this is typically implemented as a package solution.

Aspect	EFI Programme Approach	Typical Approach
Legal Complexity	An assessment of the legal requirements for SKAT Collections is outside the scope of this report. However, it is clear that there many dimensions of complexity which must be handled • 490 claim types (280 in use) • 697 claimants • 12 types of legal person To some extent, the EFI Programme has been able to abstract these into groups of Claims that can be handled in the same way, although it is not clear this analysis is fully correct. Although it was an intention at the inception of EFI that the legal requirements for debt collection should be simplified, in fact this never happened. As a result, the EFI Programme has attempted to code and automate an enormous variety of Original Needs, without simplification.	The normal approach is a combination of A) Simplification of the legal requirements in parallel with the system implementation. B) Reducing the scope of the automated IT solution to address the high volume, low – medium value work where IT is of greatest value. This assessment has not examined the relative legal complexity of SKAT Collections compared to NCA Collections.

Aspect	EFI Programme Approach	Typical Approach
Costs, benefits, scope and requirements	EFI was described by 389 Original Requirements in total, and approx. 700 pages of use cases. In many cases, the Original Requirements were vaguely described e.g. as "Der må ikke foretages fysiske sletninger af data med mindre andet er specifikt angivet for et givet begreb" ("Physical deletion of data must not be performed unless otherwise specified").	Typical enterprise systems are described by detailed requirements. E.g. for a large custom built solution, 3,000 – 5,000 requirements is typical. For a comparable package solution, 2,000 – 4,000 requirements is typical. In the creation of almost all IT systems, there is a cost/benefit analysis performed at an early stage, where requirements are examined and included or excluded based on the cost to implement and the benefits provided. This is usually informal initially, followed by increasing formality and structure as documented estimates are created for the entire solution. In most organisations, a budget is defined for a solution, and the scope is managed to fit within the allocated budget. Before the budget is fixed, detailed requirements are documented and analysed to confirm the budget and scope match.

Table 13 Assessment Details

5.8.4 Accenture Assessment

In our opinion, the EFI Programme attempted to build an extremely sophisticated and complex debt collection System. In a number of major aspects, it is more complex than systems in other countries.

The System is currently incomplete (see Section 5.1 and the separate functional report) and there is no clear definition of the scope of the System.

The implications of this assessment in our opinion are:

- That the scope of the System, as originally envisioned without simplification, would probably have required building one of the world's most complex public sector collections systems.
- That SKAT's needs for collections should be simplified significantly, thereby reducing the overall scope. This will reduce the complexity and cost of any rebuild or replacement.

5.9 Solution Architecture Observations

5.9.1 Purpose
The purpose of this section is to document findings related to the overall architecture and principles of the System and EFI Programme.

5.9.2 Key Findings
Our key findings in relation to the overall Solution Architecture are:

- The split of functionality across EFI and DMI was defined at a high level by the Original Requirements and use cases for EFI and DMI. The details of the integration were established through the integration design (see Section 5.4). In our experience, this is an unusual approach to implementing a System for debt collection, as these systems are typically implemented as a single application, providing simplicity, improved performance, reduced maintenance costs and more integrated functionality (360 degree view). The analysis team are not aware of another debt collection system that has functionality split in a similar way to EFI/DMI.
- The EFI Programme adopted a principle of flexibility in designing EFI+DMI. In our experience, flexibility is not normally considered an architecture principle. As used in the EFI Programme, this “principle” contradicts several established architecture principles such as KISS (Keep It Short and Simple) and YAGNI (You Aren’t Going to Need It).
- EFI and DMI has been designed with the aspiration of providing future flexibility. Some flexibility has been provided, however
    - It is not clear this level of flexibility was required. Most of the flexibility has not in fact been used to date. The System is intended to be more flexible than comparable systems in other countries that are working satisfactorily in production.
    - It is not clear all the functionality controlled by the flexibility works, as the level of testing this has received is not exhaustive. It is therefore likely that if functionality or combinations of functionality that have not been used to date are configured into use that new defects will emerge, and there will be data corruption resulting in the inability to provide an unequivocal account of what actions have been done and why.

5.9.3 Assessment Details

Overall Structure
The essential difference between the EFI Programme approach and the approach taken by other agencies the assessment team have experience of, is that the EFI Programme separated core Collections functions across multiple Applications. For example, to create a payment plan, calculate payment ability, or create a salary deduction plan and many other situations, EFI must retrieve detailed financial data from DMI, perform logic in EFI and DMI, and update both Applications in a consistent way. This is highly inefficient in many ways, for example:

- It will take more effort to develop the required functionality as additional to the required business logic, there are numerous web service interfaces interposed between the components performing the logic.
- At runtime, the logic will perform at a fraction of the speed of a single integrated system.

This split of functionality across EFI and DMI was defined at a high level by the Original Requirements and use cases for EFI and DMI. The details of the integration were established through the integration design (see Section 5.4).

The assessment of the EFI+DMI SOA in Section 5.4 also relates to the overall architecture.

Flexibility as a Principle

During the analysis to prepare this report, it was repeatedly stated in meetings with EFI Programme management that a key principle for the design of EFI+DMI was flexibility, e.g. in the meeting of 29th June 2015 with the EFI Programme manager.

Many well-respected lists of software architecture principles do not specifically list flexibility as a principle. Instead, they list other principles, which are intended to deliver a minimal but well-structured application that can be maintained, extended, and scaled in the future. This delivers the goal of a flexible and maintainable system.

A particular problem with designing a system to be flexible, is that unless there are specific requirements defining the type of flexibility required, then the designers are attempting to predict the future. This can result in over-engineering a system or building flexibility that is never required in practice.

Including additional functionality to anticipate possible (but unconfirmed) future requirements is directly against widely accepted architecture principles such as YAGNI and KISS.

Architecture and Flexibility

As noted above in the comments on SOA within 5.4, and as stated by the EFI Programme manager, it was intended that EFI and DMI would offer great flexibility as a platform for SKAT Collections. The System incorporates mechanisms to enable extensive configuration of the functionality to support possible future scenarios.

All non-trivial applications include some level of flexibility. At the simplest level, this may be e.g. the ability to change a VAT percentage rate without requiring to change source code, rebuild and redeploy the system. However, in our experience, it is necessary to balance a number of factors including:

- What parameters must be configurable
- The level of testing required when parameters are changed

- The complexity of the configuration mechanisms

Our experience is that other organizations with high value, high complexity systems decide that the more complex changes to configurations require a formal test within a testing environment before use in production. This decision may reduce the benefits from a customization module (of whatever technology), as changes are, in effect, treated as similar to software code changes requiring a test and release cycle.

There is no evidence the EFI Programme tested every configuration change thoroughly in a test environment before use in production. There is therefore a risk that production operation has been incorrect as a result of this execution of untested configurations in live use.

EFI/DMI and Configuration Design
In designing EFI, the EFI Programme maximized configurability, and much of this configuration could be performed by administrative users. This configuration has the potential to completely change the behavior and processing of EFI, including introducing legally incorrect behavior. This configuration is not tested rigorously in SIT02 (environment for System Integration Test), which introduces additional risks of unexpected and incorrect behavior.

When designing an application or system to be future proof, there are complex cost/benefit decisions to be made to balance designing the system to build in anticipated but unknown flexibility now and implementing the minimum functionality with no thought to the future. Most organizations strive to:

- Implement a system that implements the current known requirements, with a structure and architecture that enables, but does not implement, future expansion. A term sometimes used for this approach is “minimum viable product”. This approach is based on implementing a working “Version 1.0” into production, learning from this, and then extend and upgrade from Version 1.0 through successive versions and releases (1.1, 1.2, 2.0, etc.).
- Simplify requirements to reduce the complexity and therefore the cost of implementing the system. In the public sector, this frequently involves parallel programmes of legal and system changes

Where a configuration mechanism is implemented, this is in many cases far more complex to design, build, and test than a hard coded mechanism, or a less sophisticated mechanism with some manual steps. As examples:

- The configurable and general purpose treatment tracks engine at the core of EFI enables EFI administrators to dynamically configure infinite combinations of treatments for cases, based on a variety of parameters. In practice, comparable collections systems in other countries are based around a small number, typically 5-10, treatment tracks.

- The design of the treatment services does not enable the flexibility the business requires in practice and is not able to provide fundamental functionalities such as adding of new claims to an existing treatment.
- Although the System was intended to be flexible to suit a wide variety of future scenarios with little or no code changes, in fact, this is not proving to be the case. Real world changing needs are in fact requiring extensive and complex changes to the EFI and/or DMI code. Examples of these changing needs includes changes to the "PEF" customer processing and changes to process claims for a customer singly, rather than in a group).

Flexibility of EFI+DMI Treatments, Compared to Similar Collections Systems
The EFI Programme designed a System which allowed administrative users to create almost any combination of highly automated treatments.

By comparison, a recent state level collections system, serving over 10M citizens implemented three treatment tracks, which include some configuration (business rules) regarding the delays at each step depending on the claim type. This provided standard processing paths, with flexibility to rapidly and safely adjust the key drivers of treatment success.

Many collections systems provides flexibility to adjust the treatments, which enables the rapid adjustments of existing treatments to apply slight variations and thus enable new treatments. By comparison, EFI has separate services for each treatment track with little to no re-usability. This means that the implementation of a new treatment results in the development of a new service as there is no way to just add another "type" and then configure the business rules.

With EFI, often the addition of new customer types results in the need to create new paths and create separate modules in the system for the processing of these customers, which has an impact on system complexity and makes it more difficult to maintain in the long term. Other agencies have systems that enable the addition of new tax payer/debtor types and the catering of the flows within minimal code changes.

The DMI SAP Application backbone provides a narrow set of flexibility and workflows. Normally, agencies adjust their processes and requirements to fit within the available flexibility. This is critical to success with COTS software. However, DMI has instead custom-built extensive structures within SAP to give the illusion of a fully flexible/customizable Application. In practice, this means that key SAP functionality is simply not in use.

5.9.4 Accenture Assessment

- Although the intention was to create a highly flexible System, which could accommodate future changes easily, changes currently being planned for implementation will most likely require modifications in multiple places throughout EFI and DMI.
- In our opinion, the lifetime maintenance costs of EFI and DMI will be higher than those of comparable systems with less flexibility and functionality because EFI+DMI

includes two platforms rather than one and is intrinsically more larger and more complex.

- A less flexible System, thus less complex, would have been cheaper to build and easier to test, both to date and into the future, because construction and testing effort is driven by complexity.
- There is a significant risk that changes to the “configuration matrices” controlling elements of the functionality have errors, as these changes are not tested. There should be testing to verify that the configuration works as expected. Complex configuration should be treated like code changes.

5.10 Focus on Simple Path

5.10.1 Purpose

The purpose of this assessment was to identify whether the EFI Programme had systematically focused on the “simple path” through the System, and failed to implement other paths required for correct operation.

5.10.2 Key Findings

Our key findings in relation to the simple path are:

- The design and implementation of EFI/DMI System, in design, build and test, has focused on the “simple path”. Although the simple path may often work, real inputs and real cases require exception paths, which are inadequately handled and may fail to operate correctly.

5.10.3 Assessment Details

In a software system, the simple path is the default path through the system, with no exceptions.

As examples:

- The default path for a Claim entering EFI+DMI is for the Claim to arrive electronically, be accepted and be stored in EFI and DMI.
- The default path for a Payment Plan is for the plan to start, for the debtor to pay every instalment on time and for the plan to complete.

It is possible for design, build and test to focus on the simple path. When projects take this approach, they focus on trying to show that the System can work. When the System is eventually faced with real world conditions, the System is unable to process correctly.

In the examples above

- EFI does not enforce validations to ensure that only valid claims can enter the System. On the simple path, a correct claim can enter the System and be received correctly. However, when presented with invalid claims, EFI also accepts these, causing problems later on. The approach normally taken by systems is to implement strong validation on interfaces, especially external interfaces, to prevent invalid data ever entering the system.

- EFI is able to process a payment plan correctly if every payment is made on time. However, in reality, frequently debtors will miss a payment, incur additional debts during the course of the plan etc. In these scenarios EFI may not operate correctly.

The basis of this assessment are the following observations

- EFI was designed to permit entry to incorrect data - e.g. claims without a due date – that were legally incorrect and could not be processed. The analysis team was told by the EFI Programme management that it was assumed that the public agencies sending claims to EFI would never send incorrect data, and SKAT users would not make errors on input fields – e.g. entering a CPR number in the claim value field. Both of these errors have occurred in practice. In our experience, it is highly unusual in systems of comparable size to have validation this lax on any interface, let alone an external interface. There is no evidence there was any plan in place to ensure this assumption remained true.
- In many use cases and designs, the description or alternate paths were missing detail necessary to handle scenarios that occurred during production operation.
- It is clear from our requirements trace analysis that the test cases have focused on testing the simple path.
- Both from the automated code analysis (complexity measures) and also from the manual inspection of the code, it is clear that the code is relatively simple. Part of the explanation for this is that the logic that should be present to process the alternate paths is not present, resulting in more simple code than usual.
- EFI was designed without the possibility of ever having “expired claims” within the System, and was not designed to process correctly if a claim happened to expire while in the System. There are currently many expired claims within the System. There is no evidence there was a robust approach planned to ensure this could not happen.

5.10.4 Accenture Assessment

- In most areas analysed across EFI+DMI, the System works to some extent for simple cases and simple inputs. However, the System fails to operate correctly in many cases when presented with real world inputs.
- As designing and implementing the exception paths typically comprises 80% of the effort in an IT system in our experience and as these are substantially incomplete in EFI, there is a large effort required to remediate these if EFI/DMI is to work.
- Given the lack of detailed documented Original Needs there can be no accurate assessment of the completeness. The Original Requirements and use cases do not give a detailed and complete view on how the System should work, and therefore do not describe a finished System.

- As the cause was systematic, the reasonable conclusion is that if examined, gaps will be discovered in all areas.

5.11 Event based processing

5.11.1 Purpose

The purpose of this assessment is to examine whether the event based architecture of EFI was ever likely to work in practice. This is primarily based on the explanations of how the System worked provided in the requirements gathering workshops and the meetings listed in Section 7.1.1, together with some source code review.

5.11.2 Key Findings

Our key findings in relation to the event based processing are:

- The event based processing approach, used for "monitoring" results in excessive numbers of events and changes from a customer perspective. In practice, this means that if the automation were enabled as originally designed and implemented customers would receive excessive numbers of communications and events from EFI+DMI.
- Although no detailed analysis has been performed by the analysis team of the effort to change this processing, it is probable that resolving this issue would require major changes as it would be a fundamental change to EFI+DMI.

5.11.3 Assessment Details

EFI is architected to take an event based approach to monitoring collections. The System is intended to detect debtor's events, and react accordingly. E.g., the System is intended to monitor payment ability. When a debtor is able to pay, the System should automatically enable salary deduction, and vice versa.

As designed, the system would send a single customer multiple letters during many common scenarios. No batch approach was chosen to group changes on a customer's case. E.g., a debtor with salary deduction will receive a letter each time the deduction percentage changes, which can occur often for customers with frequent changes in payment ability.

This is based on our analysis of the following documents:

1) Modtag Fordring ODSB 1 External system interface and MF Component
2) Overordnet Delsystembeskrivelse for Inddrivelsesmotor
3) ODSB_for_Hændelsesfabrikken_v_3 5 2
4) Overordnet Delsystembeskrivelse for indsatsen Lønindeholdelse EFI_OP_00
5) Overordnet Delsystembeskrivelse for indsatsen betalingsordning EFI_OP_00
6) Overordnet Delsystembeskrivelse for indsatsen Bobehandling EFI_OP_00
7) Overordnet Delsystembeskrivelse for EFI ESDH og AandD Integration
8) Overordnet Delsystembeskrivelse for Betalingevneberegning og Budget
9) SAD - Software Architecture Document for EFI-IPO
10) EFI – Source code

A more typical approach in an event based system such as this is to aggregate outputs to customers in some way (e.g. daily, monthly). Additionally, in many scenarios, some level of smoothing is required on input data to ensure a suitable outcome is delivered at a business level – e.g. averaging payment ability over a period of time.

5.11.4 Accenture Assessment

The approach of event based processing and creation of outputs and the lack of selective controls on processing would likely be a block on fully automated processing until this entire approach is restructured.

5.12 Data Quality Impacts on Application

5.12.1 Purpose

The technical analysis has primarily focussed on the technical and functional architecture of the System and Applications. However, the System and data have impacts on each other.

The purpose of this section is to describe the implications of some findings from the data analysis on the System, and vice versa.

5.12.2 Key Findings

Our key findings in relation to the data and application are:

- There was an assumption that the System would receive valid data from the internal and external systems feeding EFI+DMI with claims, and also from all users inputting data manually. This assumption is identified because (a) the System does not do normal levels of validation and (b) it was stated in requirements workshops that the EFI Programme had no mandate to control incoming information. This does not appear to be a reasonable assumption, based on our experience of other systems, and the fact that non-valid data has been input to the System. This resulted in the System failing to process correctly when presented with incorrect production data.
- EFI does not implement rigorous external validation on input data. This is a highly unusual approach in our experience and is in violation of SOA and normal application principles.
- EFI+DMI is partially or completely unable to process records that are missing key fields.

The following table describes the Key Findings related to the data within the System, categorized against a number of areas.

Area	Definition	Observation	Source	Consequences for Data Analysis / EFI System
Consistency	To what extent is the data stored in a well-defined and described format	DMI has a well-structured description of the data model and the individual tables are described in the document. For EFI, we were not able to establish if a data model description document exists.	The list of data description documents we received for both EFI and DMI is documented in the Appendix of the Data Analysis report. The set of documents we received does not include an EFI data model description document. We received a set of SQL and DDL files for the tables in EFI.	This does not impact the findings of the data analysis. Its impact is in the additional effort that was required to perform analysis because it was necessary to reverse engineered a Data Dictionary and ER Diagrams from these SQL and DDL files using Oracle SQL Developer Data Modeler as a pre-condition to starting analysis.
Integrity	To what extent are data relationships defined and adhered to	We have identified inconsistencies between the central DMI tables in the Data Warehouse. These inconsistencies would suggest that Referential integrity is turned off at database level in DMI.	We identified 8.658 claim rows on the "DMI Fordring" table in the Data Warehouse that do not match any claim rows on "DMI Inddrivelse". (this figure is less than 1% of the total number claims in this table) We identified 2,943 claims rows on the "DMI Inddrivelse" table in the Data Warehouse that do not match any claim rows on the "DMI Haeftelse" table. (this figure is less than 1% of the total number claims in this table)	The consequences of this inconsistency is that key dates governing collection will be missing for these claims. The consequences of this inconsistency is that Collections will not be carried out on these claim. These rows were excluded from data analysis.
Uniqueness	To what extent is data unique (or is data duplicated)	In general, data is not duplicated between EFI and DMI beyond what is necessary to ensure that the two Applications must be able to function. However we understand through interviews with EFI Programme Representatives that a record is registered in both EFI and DMI databases when the claim is first received, and understand that inconsistences have been observed.	During interviews with Programme Representatives we understand that reconciliation activities have been carried out between claims sent from the KOBRA application (a SKAT application and claimant) and the claim stored in DMI, which identified inconsistencies. In addition a deeper dive as part of the same reconciliation suggested that the claim was also stored in EFI when first received and that there were also inconsistencies between this data and the claim stored in DMI.	The received claim data stored in DMI was used as-is for data analysis. We did not have access to the relevant tables in KOBRA, EFI or DMI so were unable to perform a separate validation or quantification of the number of instances where this occurred, or therefore establish the consequences of this on data analysis or on the EFI System.
Completeness	To what extent is the data set complete (or is there missing data)	We have identified instances in incompleteness in the data relating to expiration date.	We queried the DMI extract tables haeftelse_attribut and haeftelse_forældelse in the Data Warehouse extract from August 6th 2015 and identified 17,368,847 instances where both tables had a missing expiration date for the same liability. This is 17% of the total liabilities and all records are interest liabilities. In subsequent interviews with EFI Programme Representatives to understand the nature of this issue, we learned that the date is calculated 'on-the-fly' by the EFI System (EFI/DMI	A workaround was used to reduce the impact on data analysis. The logic for this workaround was provided by Programme Representatives - if the date was missing from the attribute table and it is a subclaim then the expiration date of its main claim is used instead for the purpose of analysis. We did not assess if the EFI System uses this logic.

Area	Definition	Observation	Source	Consequences for Data Analysis / EFI System
			Applications) when this data is retrieved from the DMI database.	
Uniqueness	To what extent is data unique (or is data duplicated)	The expiration date is stored in two places in DMI – the current expiration date is stored both in the expiration date transaction table is also stored in the liability table. The current expiration date can be derived from the transaction table using a rule set.	We compared the DMI tables haeftelse_attribut and haeftelse_forældelse in the Data Warehouse extract from August 6th 2015 and identified 15,816,632 instances where a liability expiration date was present on both tables but the values did not match for the same record. This is 15% of the total liabilities.	This impacted the selection of the 'correct' expiration date. Following interviews with Programme Representatives we understand that haeftelse_attribut table is more reliable, therefore we used this date value during data analysis instead of the haeftelse_forældelse date value.
Completeness	To what extent is the data set complete (or is there missing data)	We have not been able to establish if all the information needed to trace the source of an update to the expiration date registered in DMI. Although, for some payment allocations it will be possible to reverse engineer the probable expiration date update in order to trace the likely update as it is registered in DMI, it is unlikely that this will apply to all updates.	We have identified the location of the record stored in DMI of the expiration date updates, and of the reason codes related to the updates, however we have not been able to identify the location of a record indicating the treatment, treatment step or payment that resulted in the expiration date update in DMI or in EFI.	This had an impact on Retracer analysis of claims where the expiration date had been updated as a result of a treatment or treatment step. The Retracer application was unable to establish if the data had broken rules in this scenario. The consequence is that these claims were marked in the grey category
Consistency	To what extent is the data stored in a well-defined and described format	The expiration date data field in DMI is used to capture information other than the date when the claim will expire.	The expiration date is set to 31-12-9999 or 31-12-8888 in DMI to indicate special cases like bankruptcy proceedings, where the expiration of the claim is suspended. Although this use of a date which is far into the future will give the effect of preventing the claim from expiring, we have not been able to identify the logic for determining what will happen to the actual expiration date of the claim when the suspension is released in the future.	This had an impact on Retracer analysis of claims where a suspension has been lifted. The Retracer application was unable to establish if the data had broken rules in this scenario. The consequence is that these claims were marked in the grey category.

Table 14 Key Findings Related to Data in the System

5.12.3 Assessment Details

A limited investigation of specific data quality aspects was carried out by the data analysis team with the specific purpose of evaluating the consequences of data quality issues that have a direct impact on the analysis findings.

The consequences of these findings to the data analysis results are listed in the table in section 5.12.2. The consequences of these findings to processing of this data within the EFI System, comprising EFI and DMI are documented to the extent that it was possible to establish this during data analysis.

Data arrives into EFI from several hundred upstream systems, which submit Claims to EFI. Additionally data is received from internal SKAT systems. End users may also manually input data. A large quantity of data was also received during the data migration, through the same input mechanisms.

Data analysis has identified a number of significant issues within EFI and DMI. These include

- Fields required for processing that are missing. A high profile example is missing “due dates” on Claim records. Without this date, critical Application logic cannot work. There are many other examples.
- Missing record to record relationships. An example of this is liabilities without a corresponding claim.

Examples of both these issues are listed in the table in Section 5.12.2 along with the number of instances identified.

During the manual code analysis by the Accenture team, it was frequently noted that deep within the internal logic of EFI, there was Application code to work around missing data.

Related to Section 5.10, a fundamental design assumption is that data entering the System will be correct. The analysis has not identified a documented source of this assumption, however, it has been described by EFI Programme representatives in workshops (e.g. the Modtag Fordring requirements gathering and the data migration assessment), in meetings with EFI Programme management, and it is clear from the code review that there is a lower than normal level of validation.

Unfortunately, this assumption has been proven incorrect, although no action has yet been taken to remediate this. Without correct data, and with the “simple path” assumption, EFI is largely defenceless.

Typical approaches used in enterprise systems are

- Validate incoming data rigorously at interfaces, and prevent non-processable data reaching the System.

- Implement mechanisms at interfaces (e.g. batch skips, message error queues) to "filter" broken data before entry. This includes external interfaces, and may include internal interfaces also.
- Assume input data will be incorrect in every possible way, and ensure comprehensive protection is implemented.

EFI+DMI does not include mechanisms to control incorrect/broken data in a systematic way. In some cases, it implements work around logic (e.g. inferring a missing field from other data) that may or may not be correct. This may risk making a claim non-collectable.

Another example is when claims are submitted in a foreign currency, EFI stores neither exchange rate nor amount in original currency, which may cause problems explaining collected amount to debtor, especially with fluctuating currencies or errors in conversion.

5.12.4 Accenture Assessment

At present, even if EFI+DMI Applications performed perfectly with perfect data, given the data quality issues within the System it appears that the net result could be similar to the current situation with large scale problems.

A different approach should be considered in future to ensure high quality data within the System.

A further consequence of these findings is related to future attempts to cleanse data to improve the quality. Arising from the observations and investigation of the Data Quality that are documented in this Section, we have identified some technical challenges that will arise during data cleansing.

Data Quality Finding	Impact on Data Cleansing	Suggested approach to Addressing
We have identified instances in incompleteness in the data relating to expiration date.	When Data Cleansing is undertaken, to complete the expiration date data, it will be necessary to identify an alternative source from which establish a correct expiration date.	During Retracer analysis representatives from KA established a 'minimum' expiration date, which is the earliest possible expiration date for the claim under analysis. This could be adopted for data cleansing in the absence of available data. Another suggested approach is to apply the logic that is currently executed by the EFI System when it retrieves a liability record containing a blank expiration date. The approaches should be compared to establish which method produces the appropriate answer on how to fix the incompleteness.
The expiration date is stored in two places in DMI – the current expiration date is stored both in the expiration date transaction table is also stored in the liability table.	When Data Cleansing is undertaken, to harmonise the expiration date data, it will be necessary to establish which of the two dates currently stored is the correct expiration date.	During Retracer analysis representatives from KA established a 'minimum' expiration date, which is the earliest possible expiration date for the claim under analysis. This could be compared to the existing stored dates for the purpose of taking a decision on which on which value to use for the harmonised date.
We have not been able to establish if all the information needed to trace the source of an update to the expiration date registered in DMI.	When Data Cleansing is undertaken, to fix the expiration date data, it will be necessary to establish what effect	Although, for some payment allocations it will be possible to reverse engineer the probable expiration date update in order to trace the likely update as it is

Data Quality Finding	Impact on Data Cleansing	Suggested approach to Addressing
	payments, and treatments had on the value of the expiration date.	registered in DMI, an approach has not yet been formulated that applies to all updates.
It has not been possible to establish the configuration settings that were in place at any given time in the past when data was updated. (This finding is documented in the Technical Report.)	When Data Cleansing is undertaken, to cleanse data affected by a Quality Center defect that was subsequently fixed, it will be necessary to determine when the production configuration in EFI/DMI was updated to apply this fix for the defect as a pre-requisite to establishing which data records were updated before the fix was in place and therefore need cleansing.	Further investigation is needed to establish if there is an alternative method to establish which data records were updated before a specific configuration change was in place, and therefore are in need of cleansing.

Table 15 Data Quality Impact Overview

6 Appendix: List of Defined Terms

Term	Definition
ACID	Atomic, Consistent, Isolated and Durable. These are the fundamental guarantees a database provides when using transactions to ensure data integrity.
ADM	Accenture Delivery Methods (ADM) is a development methodology that supports business process analysis, application requirements and use case analysis, application design, technical architecture development, testing, and the deployment of a system. See Section 10 for an overview of ADM.
Application	An Application is an executable software program that performs a function or group of functions. It is typically composed of a single technology, and may be integrated with other applications. The term “Application” is used to describe EFI and DMI. EFI and DMI are technically separate and are built on different technologies in separate projects.
Automation	Automation is the practice of using applications and other IT to perform tasks that would otherwise be performed manually, i.e. by people. A typical example is the issue of reminder letters to debtors.
Business Process	A Business Process (or Business Process Model) is a formal description of how a business function is performed from beginning to end. The description should indicate the activities performed by users and applications, and the interfaces between each.
Code / Source Code	The “code” or “source code” refers to the human readable instructions that are either compiled and/or interpreted to form object code that can be executed by a computer system.
Configuration	Configuration refers to any data that is used to control the behaviour of a system. Configuration data may be held in files, databases or elsewhere. An example could be a fee amount held in a database. This allows the fee to be changed, without requiring changes to source code. Changes to configuration data should be tested as these can completely change the behaviour of a system.
COTS	Commercial Off The Shelf [software]. COTS refers to the use of software applications (packages) to implement a business solution. Complex business solutions will require the COTS software to be customised, often to a significant extent.
CRM	Customer Relationship Management.
Database	A database is a platform application that enables data to be stored and retrieved. Additionally, a database can be used to guarantee the integrity of the data stored within certain constraints, using ACID transactions and Relational Integrity.
Design	Design is the process of converting requirements or a higher level design into a more detailed design. Designs are typically decomposed over multiple levels from a Solution Blueprint through high level design to low level design.
DMI	“Debitormotor Inddrivelse”
EFI	“Et Fælles Inddrivelsessystem”
EFI Programme	The programme of work (a number of related projects) to deliver a new debt collection System for SKAT. The EFI Programme included the EFI project, DMI project and a number of other smaller projects or packages of work to integrate EFI and DMI with other systems within SKAT.
End to end	End to end (e2e) refers to a complete process and/or the supporting IT system. For example: • The end-to-end process of handling a Claim includes initial receipt, performing collections treatment(s) and finally closing the Claim. • The corresponding IT system(s) to enable this process includes all the IT systems and applications that integrate to perform the overall function.
Flexibility	Flexibility refers to the EFI Programme principle to implement a system that enables future business requirements to be accommodated primarily by configuration of the system, rather than source code changes.
Interface	An interface is the point where an application connects to something external to the application. Interfaces may be implemented with a wide variety of technologies including files, database tables, web services and more.
IT	Information Technology
KISS principle	Keep It Short and Simple – principle suggesting focus on simple solutions that meet the requirements
Maintainability	Maintainability is a non-functional characteristic of a system. It refers to the ability to make changes to an application over its lifetime to accommodate changing requirements.
Need	As used in this document, the Needs are the detailed documented requirements established through requirements gathering workshops for the selected sample areas. The Needs were gathered on the basis that they described what the users originally expected or required the system to do.
Original Need	The Original Needs were all that was intended that was required in order for the System to work as necessary to support SKATs debt collection business service. The Original Needs are not documented as detailed documented requirements.
Original Requirement	The Original Requirements are the requirements that were used, together with the Use Cases, as the start of the design, build and test processes for EFI and DMI, found in these documents: • *EFI 02 Leverandørens kravopfyldelse FA v1_00* • *EFI 02 Leverandørens kravopfyldelse S v1_00*

Term	Definition
	(See section 9.2.1 and 9.2.6 for details and hyperlinks.)
Package Software	Package Software: see COTS
Referential Integrity (RI)	Referential Integrity (RI) is the ability to enforce basic business rules at the database level. RI helps ensure the integrity of data by guaranteeing that fundamental constraints on data (i.e. business records) are met. As an example, RI can enforce that every Claim has at least one Liability, or that every Debtor has an Address.
Requirement	A requirement is a formal statement of what a system must do, in order to be working correctly. Typically, the scope of a business application is defined by a number of requirements. Large systems often comprise 3,000 – 10,000 requirements. An example could be "The decision letter to debtor must in all cases contain size, period, type, and due date of all claims covered by the decision."
Requirements Traceability Matrix (RTM)	A Requirements Traceability Matrix (RTM) is a) A list of all the requirements comprising the system b) An audit for each requirement of where the requirement was satisfied in design, build and test.
SAP	SAP is package software for performing many common business functions.
SDLC	Software Development Life Cycle. SDLC is used to describe the process of creating a software system.
Service	A Service (or web service) is a self-contained unit of functionality and data that performs some useful function. An example Service could be a service that allows users or other applications to check the registration number (CVR, CPR or AKR) for a customer.
Simple path	In a software system, the simple path is the default path through the system, with no exceptions. E.g. case processing of a single, simplistic claim with low complexity and no errors along the way.
SOA	Service Oriented Architecture (SOA) is the concept of creating systems based on integrating Services that provide self-contained functionality.
Specification	See Design.
System	A system is one or more applications that perform an overall business function. An example system is an email system that performs all receiving, storing and sending email (although this may be comprised of a number of discrete applications). The term "System" is used to describe the integrated combination of EFI and DMI, which provides the overall debt collection IT function for SKAT.
Test	A Test is a documented procedure that can be performed to validate compliance with a requirement. As an example, with reference to the definition of Requirement above, the corresponding test would validate that in all cases the decision letter contained the necessary details and that these were correct.
Test Stub	In any large system, some testing will have dependencies on external components, where the "remote side" of the interface is required to perform the test. It is usual to perform some testing where the "remote side" of the interface is performed with a fake system that returns sufficiently real responses to enable testing. These fake remote systems are termed "Test Stubs".
Use Case	A Use Case is a description of the steps that a number of users (actors) must perform in order to complete a business scenario. Use Case descriptions can be used, together with other designs, to provide a design for a system or application. Use Cases typically describe the "simple path" and any number of alternate paths through the system. Separate sets of use cases were used to describe the functionality for the EFI and DMI applications.
V Model	The "V Model" is a widely used model for defining the verification and validation processes for an IT system, in which each design and build output is validated via a matching testing (validation) step.
Web Service	A Web Service is a Service that provides a machine to machine interface. The interface technologies used were originally HTTP, SOAP and WSDL, but are now commonly considered to include REST.
WSDL	Web Services Description Language
YAGNI	You Aren't Going to Need It – principle to avoid "Rolls Royce" solutions when you need a "Ford"

Table 16 Defined Terms

7 Appendix: List of Meetings

7.1 Inventory of Workshops and Interviews

7.1.1 General

Date	Description	Participants	Location
23.02.2015	Introduction	• SKAT Personnel • Accenture	SKAT
03.03.2015	Introduction to SKAT BI	• SKAT Personnel • Accenture	SKAT
04.03.2015	EFI Architecture	• SKAT Personnel • Supplier • Accenture • Valcon	SKAT
10.03.2015	Testing overview	• SKAT External Consultants • Accenture	SKAT
11.03.2015, 18.03.2015, 26.03.2015	SKAT TA walkthrough	• SKAT Personnel • AccentureACN analysis team	SKAT
11.03.2015	EFI / DMI Processing	• SKAT Personnel • Accenture	SKAT
13.05.2015	EFI / DMI coordination and integration	• SKAT Personnel • Accenture	SKAT
18.05.2015	End to end design and coordination	• SKAT Personnel • Accenture	SKAT
18.06.2015	Walkthrough of test cases in Quality center	• Accenture • SKAT External Consultant	SKAT
29.06.2015	Requirements tracing / overall design decisions / negative path / system requirements	• SKAT Personnel • Accenture • SKAT External Consultant	SKAT

Table 17 Meetings

7.1.2 Modtag Fordring (Receive Claim)

Date	Description	Participants	Location
16.04.2015	Workshop 1: Modtag Fordring / Receive Claim	SKAT TA State attorneys ACN analysis team	SKAT
22.04.2015	Workshop 2: Modtag Fordring / Receive Claim	SKAT TA State attorneys ACN analysis team	SKAT
30.04.2015	Workshop 3: Modtag Fordring / Receive Claim	SKAT TA State attorneys ACN analysis team	SKAT
11.06.2015	Review with SKAT PM	SKAT PM ACN analysis team	SKAT
02.06.2015	Workshop with State Attorney	State attorneys ACN analysis team	State attorney's office
23.06.2015	Review with technical resources at SKAT	SKAT ACN analysis team	SKAT

Table 18 Workshops and Reviews - Modtag Fordring

7.1.3 Kundesaldi (Client Account Balance)

Date	Description	Participants	Location
05.05.2015	Workshop 1: Kundesaldi I DMI / Client Account Balances	SKAT TA State attorneys ACN analysis team	SKAT
07.05.2015	Workshop 2: Kundesaldi I DMI / Client Account Balances	SKAT SME SKAT BPO SKAT TA State attorneys ACN analysis team	SKAT
12.05.2015	Workshop 3: Kundesaldi I DMI / Client Account Balances	SKAT SME State attorneys ACN analysis team	SKAT
08.06.2015	Review meeting	SKAT PM ACN analysis team	SKAT
17.06.2015	Review meeting	SKAT PM ACN analysis team	SKAT
18.06.2015	Review meeting	SKAT SME ACN analysis team	SKAT
24.06.2015	Review meeting	SKAT SME CAN analysis team	SKAT

Table 19 Workshops and Reviews - Kundesaldi

7.1.4 Betalingsordning (Payment Plans)

Date	Description	Participants	Location
30.04.2015	Workshop 1: Payment plans	SKAT BPO SKAT TA State attorneys CAN analysis team	SKAT
07.05.2015	Workshop 2: Payment plans	SKAT BPO State attorneys ACN analysis team	SKAT
12.05.2015	Workshop 3: Payment plans	SKAT BPO State attorneys CAN analysis team	SKAT
19.05.2015	Workshop 4: Payment plans	SKAT BPO State attorneys, CAN analysis team	SKAT
02.06.2015	Open questions regarding requirements on payment plans	SKAT TA ACNanalysis team	SKAT
15.06.2015	Review of requirements for payment plans with SKAT PM	SKAT PM ACN Analysis team	SKAT
15.06.2015	Meeting about liability types for payment plans	State attorneys ACN analysis team	SKAT
25.06.2015	Review of the requirements for payment plans with SKAT TA	SKAT TA SKAT BPO ACNanalysis team	SKAT
01.07.2015	Review of the requirements for payment plans with SKAT BPO	SKAT BPO ACNAnalysis team	SKAT

Table 20 Workshops and Reviews – Betalingsordning

7.1.5 Lønindeholdelse (Salary Deduction)

Date	Description	Participants	Location
18.05.2015	Workshop 1: Lønindeholdelse / Salary Deduction	SKAT BPO SKAT TA State attorneys ACN analysis team	SKAT
19.05.2015	Walkthrough of basic Lønindeholdelse / Salary Deduction processes	ACN analysis team SKAT TA	SKAT
21.05.2015	Workshop 2: Lønindeholdelse / Salary Deduction	SKAT BPO SKAT TA	SKAT

		State attorneys ACN analysis team	
22.05.2015	Walkthrough of aging rules for Lønindeholdelse / Salary Deduction	ACN analysis team SKAT TA	SKAT
27.05.2015	Workshop 3: Lønindeholdelse / Salary Deduction	SKAT BPO SKAT TA State attorneys ACN analysis team	SKAT
12.06.2015	Review of Lønindeholdelse / Salary Deduction RTM	SKAT PM ACN analysis team	SKAT
17.06.2015	Review of Lønindeholdelse / Salary Deduction RTM	SKAT PM ACN analysis team	SKAT
23.06.2015	Walkthrough of current RTM	SKAT TA ACN analysis team	SKAT

Table 21 Workshops and Reviews - Lønindeholdelse

8 Appendix: Examples of Missing and Incomplete Needs

The table below show a few examples of missing and incomplete Needs for all areas. For a complete list, see the Needs requirement traceability matrix.

Area	ID	Need Description	Use Case score	Design Score	Code score	Test Case score	Notes
Receive claim	1.04.3	If the claim is not an Arrest/Urgent, then the period for when the claim originated must be equal to or earlier to the date received by SKAT	1	1	1	1	This is one of many examples, on **cross validation criteria** on received claims that must be met in order for SKAT to accept a claim. Many validation criteria have been elicited during the workshops in the form of new Needs.
Receive claim	1.13.4	If provided, the Dunning dates (Dunning date 1 and 2) must be in the past. I.e. Reminder date 1 and Reminder date 2 must be earlier than the current date	1	1	1	1	This is one of many examples of validation on **single information elements** in received claims that must be met in order for SKAT to accept a claim. Many validation criteria have been elicited during the workshops in the form of new Needs.
Receive claim	1.36.3	If a IOU (gældsbevis) date is provided, then it must be a valid date	NA	1	1	1	This is one of many examples, on **data format validations** on received claims that must be met in order for SKAT to accept a claim. Many validation criteria have been elicited during the workshops in the form of new Needs.
Receive claim	1.36.1	When a claim is submitted, it must be possible to specify if a debtor has signed an IOU for the claim. The field is optional	1	1	1	1	This is one of many examples, on **information need criteria** on received claims that must be met in order for SKAT to accept a claim. Many of such criteria have been

Area	ID	Need Description	Use Case score	Design Score	Code score	Test Case score	Notes
							elicited during the workshops in the form of new Needs.
Client Account Balance	1.3.29	Dette afhænger af, hvilke form for henstand, der er tale om. Aftalt henstand udskyder således forældelsen. Henstand i henhold til gældsinddrivelsesbekendtgørelsens § 6 er ikke forældelsesafbrydende. Henstand i henhold til skatteforvaltningslovens § 51 er forældelsesafbrydende.	1	5	N/A	1	No statements in the UC or testing done in this area. The state attorney has not definitively confirmed the legislation on this area since the Need is dependent on which type of grace period. More time is needed to investigate in detail.
Client Account Balance	1.30	Ved indbetalinger skal dækningsrækkefølgen som udgangspunkt følges. For CVR Hæftere er rækkefølgen: 1. Bøder 2. Moms, told, A-skat og AM-bidrag, punktafgifter, selskabs- og acontoskat mv., renter, gebyrer mv.	1	1	N/A	1	There is no documentation regarding the order of coverage for CVRs and also no testing done. The state attorney initially agrees but needs more time to do a proper investigation as the order of coverage legally should not only be determined based on the types of claims but also based on the treatment through which the claims are covered. The business process owner however states there are no Needs on the order of coverage for CVRs.
Payment plan	1.6	Claims which are expired at the day of creating the payment plan can't be expired. Legally it is not allowed to include information about expired claims in decision letters.	1	1	N/A	1	There are no information regarding the rules to include claims that are soon to be expired or already expired in a payment plan. SKAT stakeholders assumes that there should never exist expired claims in the System. SKAT stakeholders are also not unanimous on how expired claims that are already part of a payment plan should be handled.

Area	ID	Need Description	Use Case score	Design Score	Code score	Test Case score	Notes
Payment plan	1.18	"The content of the decision letter for created payment plans must include at least: - decision of the payment ability (details of payment ability budget excluded) - size of installment (interests not included) - frequency - due date of the first installment - the date of when the decision letter is sent out - claims (claim type, claim period, claimant, amount) - identification number of the payment plan - total amount for all claims in the payment plan. - reasoning for the size of the payment ability (what was it based on (e.g. reference to the law))	N/A	N/A	N/A	N/A	SKAT stakeholder don't know what the original requirement is on the content of the letter. The letter has been changed a fair amount of times since the EFI System went live. They are still working on the content of the letter both from a legal and business perspective.
Salary deduction	1.3.10	The System must validate that all unread correspondence registered in the System from the debtor is processed before any decisions are made to a debtor's salary deduction treatment.	1	1	1	1	It is a legal Need, that the System must check whether a budget or complaint (often based on a notification received by a debtor) has been sent to EFI, and whether this information has been "processed" before sending out a decision letter on starting salary deduction. If a budget sent in by a debtor has not been processed, SKAT, in general, risks starting or changing a treatment on an incorrect payment ability basis. However, this validation is currently not implemented.
Salary deduction	2.4.1	Automatically suspending (Bero) an automatic salary deduction, must not affect the aging of the claims included by the decision. This is a preliminary assessment of the Need. The state attorneys are currently investigating the legal Need related to bero.	1	2	1	1	The Use Case documentation, ODSB and code describes that suspension (Bero), in conflict with this Need, affects aging on all claims included in the decision.

Area	ID	Need Description	Use Case score	Design Score	Code score	Test Case score	Notes
							During the workshop, we discussed an inherent risk associated with this Need. If Bero does not interrupt aging, there is a risk that a caseworker sets the Bero period for so long, that claims will expire before resuming salary deduction. There is to our knowledge currently no constraints implemented in the System to avert this from happening.
Salary deduction	4.4.4	When a main claim expires, all sub claims: opkrævnings and inddrivelses-interests and inddrivelses-fees related to that main claim will also expire.	1	1	5	1	The Use Case documentation and ODSB does not differ between main claims and sub claims, hence this Need is not described. Furthermore, we have not been able to identify any Test Cases that tests this Need. Still this functionality is implemented. This is a large risk, since aging rules in regards to the main claims and various sub claims (accumulated interest and simulated interest for types: inddrivelses-interests and opkrævnings-interests, fees and fines) differs and are highly complex.

Table 22 Examples of Missing and Incomplete Needs

9 Appendix: Documents Examined

9.1 EFI + DMI System Level

9.1.1 Analysis

None Identified

9.1.2 Design

Description	Date / Version
08 EFI MF DMI - Processer.vsd	Last changed 3/7/2013

Table 23 Design Documents Examined

9.1.3 Test

The following set of testscripts were assessed when tracing requirements and reviewing approach to testing:

Description	Date / Version
FASE2_EFI	N/A

Table 24 Test Documents Examined

9.2 EFI

9.2.1 Analysis

9.2.1.1 EFI Requirements & Use Cases Analysed

Description	Date / Version
EFI 02 Leverandørens kravopfyldelse FA v1_00	N/A
EFI 02 Leverandørens kravopfyldelse S v1_00	N/A
EFI 01_02 Forr processer og akt beskrivelser v1_00	N/A
UC 99.1.1 Modtag fordring via WEB	04.06.2010
UC 99.1.2 Modtag fordring System til System	14.04.2010
UC 99.1.4 Opret fordring	14.04.2010
UC 06.1.1 Opret eller rediger betalingsordning - Brugergrænseflade	27.10.2010
UC 6.1.3 EFI Use-case Send afgørelse om fastsættelse af betalingsordning – Brugergrænseflade	28.04.2010
UC 6.1.6 EFI Use-case opdater betalingsordning med ny fordring – EFI	28.04.2010
UC 60.1.15 Overvåg	18.04.2010
UC 03.1.1 Varsko kunde om lønindeholdelse	05.08.2010
UC 03.1.2 Iværksæt lønindeholdelse	05.08.2010
UC 03.1.4 Slet lønindeholdelse	05.08.2010
UC 03.1.6 Opdater lønindeholdelse med yderligere fordring	05.08.2010
UC 03.1.7 Rediger iværksat lønindeholdelse	05.08.2010
UC 03.1.8 Berostil lønindeholdelse, helt eller delvis	05.08.2010
UC 03.1.9 Genoptag lønindeholdelse	05.08.2010
UC 50.2.1 Ryk for andet end betaling	05.08.2010
UC 60.1.16 Sagsbehandl	19.05.2010
UC 04.4.3 Genberegn forældelse	Version by May 2015
UC 15.1.1 Anmod om anerkendelse af fordring	Version by May 2015
Proces 03.1 Iværksættelse af lønindeholdelse	Version by May 2015

Table 25 EFI Requirements & Use Cases Analysed

9.2.2 Design

9.2.2.1 EFI Design Specifications

Description	Date / Version
Overordnet Delsystembeskrivelse for indsatsen betalingsordning EFI_OP_00270415	27.04.2015
Overordnet Delsystembeskrivelse for Indsatser EFI_OP_00	06.11.2013
Overordnet Delsystembeskrivelse for Betalingevneberegning og Budget041214	28.07.2014

Overordnet Delsystembeskrivelse for indsatsen Lønindeholdelse EFI_OP_00	25.03.2015
Overordnet Delsystembeskrivelse for Sagsbehandlerportalen092015	14.12.2014
Overordnet forretningsmæssig beskrivelse_EFI_OP_00	21.05.2010
Modtag Fordring ODSB 1 External system interface and MF Component	05.08.2014
Modtag Fordring ODSB 2 Receive Debts Dialogues	19.06.2015
Modtag Fordring ODSB 3 Claimants and agreements	08.03.2013
Modtag Fordring ODSB 4 DMI Dialogues250315	25.03.2015
Modtag Fordring ODSB 5 Alternative Liabilities	02.07.2015
Regel matricer (EFI)	N/A
Fordringstype_fordringsoplysninger	N/A
ODSB_for_Hændelsesfabrikken_v_3 5 2	15.11.2012
Overordnet Delsystembeskrivelse for Inddrivelsesmotor	21.05.2015
Overordnet Delsystembeskrivelse for indsatsen Bobehandling EFI_OP_00	14.04.2015
Overordnet Delsystembeskrivelse for EFI ESDH og AandD Integration	05.03.2015

Table 26 EFI Design Specifications

9.2.3 Code

9.2.3.1 Code Base for Automated Analysis

efi-2.80.zip (md5 checksum: f2bb1dd6f9f8cf1b5ac4d13e1d3c86e4)

File	Package	File	Package
MfservicesServices	dk.skat.efi.wls.mf	IALoenindeholdelseAfgoerelseTypeEnum	dk.skat.efi.wls.ia.da.domain.indsatser.typer
Mfserviceservicesimpl	dk.skat.efi.wls.mf	IAMeddelelsePakke	dk.skat.efi.wls.ia.da.domain.indsatser.typer
fromxmlhelper	dk.skat.efi.wls.mf.adapter	IAOpgave	dk.skat.efi.wls.ia.da.domain.indsatser.typer
toxmlhelper	dk.skat.efi.wls.mf.adapter	IAProcent	dk.skat.efi.wls.ia.da.domain.indsatser.typer
mfexception	dk.skat.efi.wls.mf.common.exception	IASamarbejdPart	dk.skat.efi.wls.ia.da.domain.indsatser.typer
mftekniskexception	dk.skat.efi.wls.mf.common.exception	IASENummer	dk.skat.efi.wls.ia.da.domain.indsatser.typer
mfsekvensnummergenerator	dk.skat.efi.wls.mf.common.sekvensnummer	IAUdlaegIndsatsOpgaveTypeEnum	dk.skat.efi.wls.ia.da.domain.indsatser.typer
mfsekvensnummergeneratorimpl	dk.skat.efi.wls.mf.common.sekvensnummer	IAAdresse	dk.skat.efi.wls.ia.da.domain.indsatser.typer
aftaledao	dk.skat.efi.wls.mf.dao.aftale	IAAlmindelig	dk.skat.efi.wls.ia.da.domain.indsatser.typer
aftaledaoimpl	dk.skat.efi.wls.mf.dao.aftale	IAAlternativ	dk.skat.efi.wls.ia.da.domain.indsatser.typer
AendrFordringDao	dk.skat.efi.wls.mf.dao.fordring	AdresseAendretConverter	dk.skat.efi.im.converters.haendelser
AendrFordringDaoImpl	dk.skat.efi.wls.mf.dao.fordring	AnbefaletSporSkabelonConverter	dk.skat.efi.im.converters.haendelser

AlternativKontaktDao	dk.skat.efi.wls.mf.dao.fordring	BeloebConverter	dk.skat.efi.im.converters.haendelser
AlternativKontaktDaoImpl	dk.skat.efi.wls.mf.dao.fordring	BEODaekningAendretConverter	dk.skat.efi.im.converters.haendelser
FordringAktionDao	dk.skat.efi.wls.mf.dao.fordring	BEORateAendretConverter	dk.skat.efi.im.converters.haendelser
FordringAktionDaoImpl	dk.skat.efi.wls.mf.dao.fordring	BerostilLoenindeholdelseConverter	dk.skat.efi.im.converters.haendelser
LeveranceDao	dk.skat.efi.wls.mf.dao.fordring	BetalEvneFaldetConverter	dk.skat.efi.im.converters.haendelser
LeveranceDaoImpl	dk.skat.efi.wls.mf.dao.fordring	BetalEvneFaldetVarigtConverter	dk.skat.efi.im.converters.haendelser
OpretFordringDao	dk.skat.efi.wls.mf.dao.fordring	BetalEvneNulConverter	dk.skat.efi.im.converters.haendelser
OpretFordringDaoImpl	dk.skat.efi.wls.mf.dao.fordring	BetalEvneSBetalEvneAendretConverter	dk.skat.efi.im.converters.haendelser
Afregningoplysninger	dk.skat.efi.wls.mf.domain.aftale	BetalEvneSLFaldetConverter	dk.skat.efi.im.converters.haendelser
Aftale	dk.skat.efi.wls.mf.domain.aftale	BetalEvneSLStegetConverter	dk.skat.efi.im.converters.haendelser
AlternativAdresse	dk.skat.efi.wls.mf.domain.aftale	BetalEvneStegetConverter	dk.skat.efi.im.converters.haendelser
BerigelseValideringFelt	dk.skat.efi.wls.mf.domain.aftale	BetalEvneStegetVarigtConverter	dk.skat.efi.im.converters.haendelser
FordringHaverFordringType	dk.skat.efi.wls.mf.domain.aftale	BetalingOrdningOprettetConverter	dk.skat.efi.im.converters.haendelser
FordringOplysninger	dk.skat.efi.wls.mf.domain.aftale	BetalingsordningMisligeholdtConverter	dk.skat.efi.im.converters.haendelser
FordringTypeAftale	dk.skat.efi.wls.mf.domain.aftale	BFSAfsoningAflysConverter	dk.skat.efi.im.converters.haendelser
Modregningoplysninger	dk.skat.efi.wls.mf.domain.aftale	BFSAfsoningOpdaterConverter	dk.skat.efi.im.converters.haendelser
OplysningerOmModregningPerFordringstype	dk.skat.efi.wls.mf.domain.aftale	BFSGensendVarselConverter	dk.skat.efi.im.converters.haendelser
AftaleBerigelseKode	dk.skat.efi.wls.mf.domain.aftale.enums	BFSKorrektionSendConverter	dk.skat.efi.im.converters.haendelser
AftaleKanSkalEjKode	dk.skat.efi.wls.mf.domain.aftale.enums	BFSOpdaterPolitikredsConverter	dk.skat.efi.im.converters.haendelser
FordringFeltKode	dk.skat.efi.wls.mf.domain.aftale.enums	BFSSendAnmodningConverter	dk.skat.efi.im.converters.haendelser
FordringhaverAftaleType	dk.skat.efi.wls.mf.domain.aftale.enums	BFSSendVarselConverter	dk.skat.efi.im.converters.haendelser
FordringhaverArt	dk.skat.efi.wls.mf.domain.aftale.enums	BFSVarselAendretConverter	dk.skat.efi.im.converters.haendelser

FejlAdvis	dk.skat.efi.wls.mf.domain.common	BobGemKontaktConverter	dk.skat.efi.im.converters.haendelser
Kunde	dk.skat.efi.wls.mf.domain.common	BobSletKontaktConverter	dk.skat.efi.im.converters.haendelser
KundeStruktur	dk.skat.efi.wls.mf.domain.common	BookingSvarConverter	dk.skat.efi.im.converters.haendelser
MFAkteringNote	dk.skat.efi.wls.mf.domain.common	BosagAendrAutomatiskConverter	dk.skat.efi.im.converters.haendelser
MFKundeStruktur	dk.skat.efi.wls.mf.domain.common	BosagAendrConverter	dk.skat.efi.im.converters.haendelser
Note	dk.skat.efi.wls.mf.domain.common	ErkendFordringFristOverskredetConverter	dk.skat.efi.im.converters.haendelser
Periode	dk.skat.efi.wls.mf.domain.common	ErkendFordringGenstartConverter	dk.skat.efi.im.converters.haendelser
RenteBeregningModel	dk.skat.efi.wls.mf.domain.common	ErkendFordringKundehenvendelseConverter	dk.skat.efi.im.converters.haendelser
Rettighedshaver	dk.skat.efi.wls.mf.domain.common	ErkendFordringSagsbehandlerErkenderConverter	dk.skat.efi.im.converters.haendelser
ValutaBeloeb	dk.skat.efi.wls.mf.domain.common	ETLConverter	dk.skat.efi.im.converters.haendelser
AlternativKontaktType	dk.skat.efi.wls.mf.domain.common.enums	FordringOprettetConverter	dk.skat.efi.im.converters.haendelser
FejlAdvisCode	dk.skat.efi.wls.mf.domain.common.enums	FordringSaldoAendretConverter	dk.skat.efi.im.converters.haendelser
FordringAktionKode	dk.skat.efi.wls.mf.domain.common.enums	ForkyndelseDatoAendrConverter	dk.skat.efi.im.converters.haendelser
FordringArt	dk.skat.efi.wls.mf.domain.common.enums	FristOverskredetCirkulaerskrivelseEjModtagetConverter	dk.skat.efi.im.converters.haendelser
FordringReturAarsag	dk.skat.efi.wls.mf.domain.common.enums	FristOverskredetModtagelseAfAdkomsterklaeringConverter	dk.skat.efi.im.converters.haendelser
HaeftelseFormKode	dk.skat.efi.wls.mf.domain.common.enums	GenoptagSporSkifteConverter	dk.skat.efi.im.converters.haendelser
KundeNummerType	dk.skat.efi.wls.mf.domain.common.enums	HaeftelseForaeldelseConverter	dk.skat.efi.im.converters.haendelser
MFQueryEnum	dk.skat.efi.wls.mf.domain.common.enums	HaendelseCommonAttributesConverter	dk.skat.efi.im.converters.haendelser
SubsidiaerHaeftelse	dk.skat.efi.wls.mf.domain.common.enums	HaendelseConverter	dk.skat.efi.im.converters.haendelser
AlternativKontakt	dk.skat.efi.wls.mf.domain.fordring	HaendelseConverterImpl	dk.skat.efi.im.converters.haendelser
BasisAendrInfo	dk.skat.efi.wls.mf.domain.fordring	HaendelseFilterConverter	dk.skat.efi.im.converters.haendelser

BasisOpretInfo	dk.skat.efi.wls.mf.domain.fordring	HaendelseModtagConverter	dk.skat.efi.im.converters.haendelser
FeltVaerdier	dk.skat.efi.wls.mf.domain.fordring	HaendelseModtagConverterImpl	dk.skat.efi.im.converters.haendelser
FordringAktion	dk.skat.efi.wls.mf.domain.fordring	HenstandAendretConverter	dk.skat.efi.im.converters.haendelser
FordringFeltVaerdier	dk.skat.efi.wls.mf.domain.fordring	IHaendelseConverter	dk.skat.efi.im.converters.haendelser
Fordringhaver	dk.skat.efi.wls.mf.domain.fordring	IMETLAnmeldelseStatusCheckConverter	dk.skat.efi.im.converters.haendelser
HaeftelseForhold	dk.skat.efi.wls.mf.domain.fordring	IndkomsttypeAendretConverter	dk.skat.efi.im.converters.haendelser
IdentifiedKunde	dk.skat.efi.wls.mf.domain.fordring	IndsatsFordringFjernetConverter	dk.skat.efi.im.converters.haendelser
IndberetLeverance	dk.skat.efi.wls.mf.domain.fordring	IndsatsFordringTilfoejConverter	dk.skat.efi.im.converters.haendelser
OpretFordring	dk.skat.efi.wls.mf.domain.fordring	KFIAdresseConverter	dk.skat.efi.im.converters.haendelser
AarsagStruktur	dk.skat.efi.wls.mf.domain.fordring	KOBVarslFristAendretConverter	dk.skat.efi.im.converters.haendelser
HaeftelseStruktur	dk.skat.efi.wls.mf.domain.underret	KundemoedeAendrConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackService	dk.skat.efi.wls.mf.inboundservice.dmi	KundemoedeGennemfoertConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackServiceImpl	dk.skat.efi.wls.mf.inboundservice.dmi	LoenIndholdelseBegrundelseConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackRequest	dk.skat.efi.wls.mf.inboundservice.dmi.request	LoenIndholdelseGensendIvaerksaetConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackResponse	dk.skat.efi.wls.mf.inboundservice.dmi.response	MeddelelseIkkeModtagetConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetService	dk.skat.efi.wls.mf.inboundservice.ekstern	MeddelelseIkkeSendtConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetServiceImpl	dk.skat.efi.wls.mf.inboundservice.ekstern	MeddelelsePakkeConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetResponse	dk.skat.efi.wls.mf.inboundservice.ekstern.response	MoedeAendrConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetRequest	dk.skat.efi.wls.mf.inboundservice.ekstern.request	MultiHaendelseModtagConverter	dk.skat.efi.im.converters.haendelser
FordringOpretService	dk.skat.efi.wls.mf.inboundservice.shared	MultiHaendelseModtagConverterImpl	dk.skat.efi.im.converters.haendelser

FordringOpretServiceImpl	dk.skat.efi.wls.mf.inboundservice.shared	NoPayloadConverter	dk.skat.efi.im.converters.haendelser
SuperInboundXmlService	dk.skat.efi.wls.mf.inboundxmlservice	OpgaveOpretConverter	dk.skat.efi.im.converters.haendelser
SuperInboundXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice	OpretOpgavePayloadConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackXmlService	dk.skat.efi.wls.mf.inboundxmlservice.dmi	RykBetalingsFristAendretConverter	dk.skat.efi.im.converters.haendelser
FordringAsynkronOpretCallbackXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice.dmi	SagsbehandlerErkenderConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetXmlService	dk.skat.efi.wls.mf.inboundxmlservice.ekstern	ScoringConverter	dk.skat.efi.im.converters.haendelser
FordringIndberetXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice.ekstern	StartIndsatsConverter	dk.skat.efi.im.converters.haendelser
OIOFordringIndberetXmlService	dk.skat.efi.wls.mf.inboundxmlservice.ekstern.oio	StopIndsatsConverter	dk.skat.efi.im.converters.haendelser
OIOFordringIndberetXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice.ekstern.oio	StopSporSkifteConverter	dk.skat.efi.im.converters.haendelser
SuperOIOInboundXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice.ekstern.oio	UdlaegAktivAndelsboligSendRykkerConverter	dk.skat.efi.im.converters.haendelser
FordringOpretXmlService	dk.skat.efi.wls.mf.inboundxmlservice.portal	UdlaegAktivAndelsboligTinglysningAendrConverter	dk.skat.efi.im.converters.haendelser
FordringOpretXmlServiceImpl	dk.skat.efi.wls.mf.inboundxmlservice.portal	UdlaegAktivAndelsboligTinglysningFristAendrConverter	dk.skat.efi.im.converters.haendelser
AftaleService	dk.skat.efi.wls.mf.internalservice	UdlaegAktivFjernConverter	dk.skat.efi.im.converters.haendelser
AftaleServiceImpl	dk.skat.efi.wls.mf.internalservice	UdlaegAktivForaeldelseDatoAendrConverter	dk.skat.efi.im.converters.haendelser
FordringAktionService	dk.skat.efi.wls.mf.internalservice	UdlaegAktivTinglysConverter	dk.skat.efi.im.converters.haendelser
FordringAktionServiceImpl	dk.skat.efi.wls.mf.internalservice	UdlaegBladDanConverter	dk.skat.efi.im.converters.haendelser
FordringHaverService	dk.skat.efi.wls.mf.internalservice	UdlaegEjGennemfoertConverter	dk.skat.efi.im.converters.haendelser
FordringHaverServiceImpl	dk.skat.efi.wls.mf.internalservice	UdlaegKladdeGemConverter	dk.skat.efi.im.converters.haendelser
FordringService	dk.skat.efi.wls.mf.internalservice	UdlaegPolitieftersoegningAnmodConverter	dk.skat.efi.im.converters.haendelser

FordringServiceImpl	dk.skat.efi.wls.mf.internalservice	UdlaegTilsigelseSendConverter	dk.skat.efi.im.converters.haendelser
FordringValiderService	dk.skat.efi.wls.mf.internalservice	HaendelseDao	dk.skat.efi.im.dao
FordringValiderServiceImpl	dk.skat.efi.wls.mf.internalservice	HaendelseDaoImpl	dk.skat.efi.im.dao
KFIMultiOpretAendrFordringService	dk.skat.efi.wls.mf.internalservice	HaendelseFilterDao	dk.skat.efi.im.dao
KFIMultiOpretAendrFordringServiceImpl	dk.skat.efi.wls.mf.internalservice	HaendelseFilterDaoImpl	dk.skat.efi.im.dao
KundeService	dk.skat.efi.wls.mf.internalservice	IndsatsDao	dk.skat.efi.im.dao
KundeServiceImpl	dk.skat.efi.wls.mf.internalservice	IndsatsDaoImpl	dk.skat.efi.im.dao
SuperInternalServiceImpl	dk.skat.efi.wls.mf.internalservice	KundeDao	dk.skat.efi.im.dao
SuperService	dk.skat.efi.wls.mf.internalservice	KundeDaoImpl	dk.skat.efi.im.dao
SuperServiceImpl	dk.skat.efi.wls.mf.internalservice	SporDao	dk.skat.efi.im.dao
ValiderOgBerigHaeftelsesforholdService	dk.skat.efi.wls.mf.internalservice	SporDaoImpl	dk.skat.efi.im.dao
ValiderOgBerigHaeftelsesforholdServiceImpl	dk.skat.efi.wls.mf.internalservice	SporRegelDao	dk.skat.efi.im.dao
FordringAktionOpretFordringSchedulerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	SporRegelDaoImpl	dk.skat.efi.im.dao
AbstractOpretAendrEFIFordringerSchedulerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	SporSkabelonDao	dk.skat.efi.im.dao
FordringAktionSchedulerTask	dk.skat.efi.wls.mf.internalservice.scheduling	SporSkabelonDaoImpl	dk.skat.efi.im.dao
FordringAktionSchedulerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	SporSkabelonIndsatsSkabelonDao	dk.skat.efi.im.dao
FordringAktionTilbagekaldSchedulerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	SporSkabelonIndsatsSkabelonDaoImpl	dk.skat.efi.im.dao
OpretDMIFordringerTask	dk.skat.efi.wls.mf.internalservice.scheduling	HaendelsesWork	dk.skat.efi.im.dispatchers.haendelse
OpretDMIFordringerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	HaendelsesWorkExecuter	dk.skat.efi.im.dispatchers.haendelse
OpretEFIFordringerSchedulerTaskImpl	dk.skat.efi.wls.mf.internalservice.scheduling	HaendelsesWorkExecuterImpl	dk.skat.efi.im.dispatchers.haendelse
OpretEFIFordringSchedulerTask	dk.skat.efi.wls.mf.internalservice.scheduling	HaendelsesWorkTaskImpl	dk.skat.efi.im.dispatchers.haendelse

FordringAsynkronOpretService	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMIndsats	dk.skat.efi.im.domain
FordringAsynkronOpretServiceImpl	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMKundeData	dk.skat.efi.im.domain
DMIFordringSynkronOpretService	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMKundeLaas	dk.skat.efi.im.domain
DMIFordringSynkronOpretServiceImpl	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMSpor	dk.skat.efi.im.domain
DMIFordringTilbagekaldService	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMSporRegel	dk.skat.efi.im.domain
DMIFordringTilbagekaldServiceImpl	dk.skat.efi.wls.mf.outboundservice.dmi.fordring	IMSporRegelType	dk.skat.efi.im.domain
DMIFordringOpretRequest	dk.skat.efi.wls.mf.outboundservice.dmi.fordring.request	IMSporSkabelon	dk.skat.efi.im.domain
DMIFordringAsynkronOpretResponse	dk.skat.efi.wls.mf.outboundservice.dmi.fordring.response	IMSporSkabelonIndsatsParameter	dk.skat.efi.im.domain
DMIFordringSynkronOpretResponse	dk.skat.efi.wls.mf.outboundservice.dmi.fordring.response	IMSporSkabelonIndsatsSkabelon	dk.skat.efi.im.domain
KFIFordringMultiOpretService	dk.skat.efi.wls.mf.outboundservice.kfi	MultiOpretHaendelse	dk.skat.efi.im.haendelseadmin
KFIFordringMultiOpretServiceImpl	dk.skat.efi.wls.mf.outboundservice.kfi	MultiOpretHaendelseImpl	dk.skat.efi.im.haendelseadmin
KFIFordringMultiOpretRequest	dk.skat.efi.wls.mf.outboundservice.kfi.request	OpretHaendelse	dk.skat.efi.im.haendelseadmin
FordringMultiOpretResponse	dk.skat.efi.wls.mf.outboundservice.kfi.response	OpretHaendelseImpl	dk.skat.efi.im.haendelseadmin
AlternativKontaktSoegXmlService	dk.skat.efi.wls.mf.outboundxmlservice.akr	BudgetHent	dk.skat.efi.services.bebb.adapters
AlternativKontaktSoegXmlServiceImpl	dk.skat.efi.wls.mf.outboundxmlservice.akr	EFIBetalingEvneAendrAdapterImpl	dk.skat.efi.services.bebb.adapters
DMIFordringAsynkronOpretXmlService	dk.skat.efi.wls.mf.outboundxmlservice.dmi.fordring	EFIBetalingEvneAsynkronHentAdapterImpl	dk.skat.efi.services.bebb.adapters

DMIFordringAsynkronOpretXmlServiceImpl	dk.skat.efi.wls.mf.outboundxmlservice.dmi.fordring	EFIBetalingEvneBfyModtagAdapterImpl	dk.skat.efi.services.bebb.adapters
DMIFordringSynkronOpretXmlService	dk.skat.efi.wls.mf.outboundxmlservice.dmi.fordring	EFIBetalingEvneBudgetAendrAdapterImpl	dk.skat.efi.services.bebb.adapters
DMIFordringSynkronOpretXmlServiceImpl	dk.skat.efi.wls.mf.outboundxmlservice.dmi.fordring	EFIBetalingEvneBudgetSendAdapter	dk.skat.efi.services.bebb.adapters
KFIFordringMultiOpretXmlService	dk.skat.efi.wls.mf.outboundxmlservice.kfi	EFIBetalingEvneBudgetSendAdapterImpl	dk.skat.efi.services.bebb.adapters
KFIFordringMultiOpretXmlServiceImpl	dk.skat.efi.wls.mf.outboundxmlservice.kfi	EFIBetalingEvneEjendomModtagAdapter	dk.skat.efi.services.bebb.adapters
OIOMFFordringIndberetXmlService	dk.skat.efi.wls.mf.outboundxmlservice.oio	EFIBetalingEvneEjendomModtagAdapterImpl	dk.skat.efi.services.bebb.adapters
OIOMFFordringIndberetXmlServiceImpl	dk.skat.efi.wls.mf.outboundxmlservice.oio	EFIBetalingEvneForsoergerpligtGenberegnAdapter	dk.skat.efi.services.bebb.adapters
AktivitetCommon	dk.skat.efi.wls.aa.loen	EFIBetalingEvneForsoergerpligtGenberegnAdapterImpl	dk.skat.efi.services.bebb.adapters
AktivitetConstants	dk.skat.efi.wls.aa.loen	EFIBetalingEvneHentAdapterImpl	dk.skat.efi.services.bebb.adapters
Berostilloenindeholdelseheltellerdelvist	dk.skat.efi.wls.aa.loen	EFIBetalingEvneKoeretoejModtagAdapter	dk.skat.efi.services.bebb.adapters
BerostilloenindeholdelseheltellerdelvistImpl	dk.skat.efi.wls.aa.loen	EFIBetalingEvneKoeretoejModtagAdapterImpl	dk.skat.efi.services.bebb.adapters
Bookressourcertilloenindeholdelse	dk.skat.efi.wls.aa.loen	EFIBetalingEvneNettoIndkomstAendr	dk.skat.efi.services.bebb.adapters
BookressourcertilloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	EFIBetalingEvneNettoIndkomstListAdapter	dk.skat.efi.services.bebb.adapters
Bookressourcetilgenoptagelseafloenindeholdelse	dk.skat.efi.wls.aa.loen	IBfyModtag	dk.skat.efi.services.bebb.adapters
BookressourcetilgenoptagelseafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	IBudgetHent	dk.skat.efi.services.bebb.adapters
Bookressourcetilgenoptagloenindeholdelseredigerloendeindeholdelseellerbooksagsbehandler	dk.skat.efi.wls.aa.loen	ILoenSimuler	dk.skat.efi.services.bebb.adapters
BookressourcetilgenoptagloenindeholdelseredigerloendeindeholdelseellerbooksagsbehandlerImpl	dk.skat.efi.wls.aa.loen	INettoIndkomstAendr	dk.skat.efi.services.bebb.adapters

Bookressourcetilmeddelelseomanmodningomloenoplysninger	dk.skat.efi.wls.aa.loen	INettoIndkomstAendringHaendelseModtag	dk.skat.efi.services.bebb.adapters
BookressourcetilmeddelelseomanmodningomloenoplysningerImpl	dk.skat.efi.wls.aa.loen	INettoIndkomstList	dk.skat.efi.services.bebb.adapters
Bookressourcetilnedsaettelseafloenindeholdelse	dk.skat.efi.wls.aa.loen	LoenSimuler	dk.skat.efi.services.bebb.adapters
BookressourcetilnedsaettelseafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	NettoIndkomstAendringHaendelseModtag	dk.skat.efi.services.bebb.adapters
Bookressourcetilopdateringafloenindeholdelsemedyderligerefordring	dk.skat.efi.wls.aa.loen	DokumentOpretAdapter	dk.skat.efi.services.dp.adapters
BookressourcetilopdateringafloenindeholdelsemedyderligerefordringImpl	dk.skat.efi.wls.aa.loen	MeddelelseSendAkterAdapter	dk.skat.efi.services.dp.adapters
Bookressourcetilvarslingafstigningafloenindeholdelse	dk.skat.efi.wls.aa.loen	IAIndsatsBetalingOrdningHentAdaptor	dk.skat.efi.services.ia
BookressourcetilvarslingafstigningafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	IAIndsatsLoenindeholdelseHentAdaptor	dk.skat.efi.services.ia
Booksagsbehandlertilhaandteringafmeddelelsesfejl	dk.skat.efi.wls.aa.loen	AddRemoveClaimController	dk.skat.efi.portal.sag.controller.actions.addremoveclaim
BooksagsbehandlertilhaandteringafmeddelelsesfejlImpl	dk.skat.efi.wls.aa.loen	BetalingsordningController	dk.skat.efi.portal.sag.controller.actions.betalingsordning
Forhoejloenindeholdelsesprocent	dk.skat.efi.wls.aa.loen	MFCreateDebtController	dk.skat.efi.portal.sag.mf.controller.createdebt
ForhoejloenindeholdelsesprocentImpl	dk.skat.efi.wls.aa.loen	MFCreateDebtFacade	dk.skat.efi.portal.sag.mf.facade
Genoptagloenindeholdelse	dk.skat.efi.wls.aa.loen	MFCreateDebtFacadeImpl	dk.skat.efi.portal.sag.mf.facade.impl
GenoptagloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	AktivitetCommon	dk.skat.efi.wls.aa.beo
Indsatsfordringtilfoejfjern	dk.skat.efi.wls.aa.loen	BEOAfbrydBetalingsordningImpl	dk.skat.efi.wls.aa.beo
IndsatsfordringtilfoejfjernImpl	dk.skat.efi.wls.aa.loen	BEOBookRessourceTilBetalingsordningImpl	dk.skat.efi.wls.aa.beo

Ivaerksaetloenindeholdelse	dk.skat.efi.wls.aa.loen	BEOBookRessourceTilMeddelelseOmAEndretBetalingsordningImpl	dk.skat.efi.wls.aa.beo
IvaerksaetloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	BEOBookSagsbehandlerTilMeddelelsesfejlImpl	dk.skat.efi.wls.aa.beo
Ivaerksaetloenindeholdelsemedyderligerefordringer	dk.skat.efi.wls.aa.loen	BEOBookSagsbehandlerTilOpfoelgningImpl	dk.skat.efi.wls.aa.beo
IvaerksaetloenindeholdelsemedyderligerefordringerImpl	dk.skat.efi.wls.aa.loen	BEOIndsatsFordringTilfoejFjernImpl	dk.skat.efi.wls.aa.beo
Ivaerksaetnedsaettelseafloenindeholdelse	dk.skat.efi.wls.aa.loen	BEORedigerBetalingsordningImpl	dk.skat.efi.wls.aa.beo
IvaerksaetnedsaettelseafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	BEOSendMeddelelseOgIvaerksaetBetalingsordningImpl	dk.skat.efi.wls.aa.beo
Ivaerksaetstigningafloenindeholdelse	dk.skat.efi.wls.aa.loen	RuleBEOBetalingordningSkalVaereAktiv	dk.skat.efi.wls.aa.beo.rules
IvaerksaetstigningafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	RuleBEOTvungenSkalHaveBetalingsevneFrivilligSkalHaveRatebeloeb	dk.skat.efi.wls.aa.beo.rules
LoenPct	dk.skat.efi.wls.aa.loen	AktivitetsAfviklerImpl	dk.skat.efi.wls.aa
Nedsaetloenindeholdelsesprocent	dk.skat.efi.wls.aa.loen	IAktivitetsAfvikler	dk.skat.efi.wls.aa
NedsaetloenindeholdelsesprocentImpl	dk.skat.efi.wls.aa.loen	AbstractAktivitet	dk.skat.efi.wls.aa.common
Opdaterfristformodtagelseafangivelse	dk.skat.efi.wls.aa.loen	AbstractAktivitetBase	dk.skat.efi.wls.aa.common
OpdaterfristformodtagelseafangivelseImpl	dk.skat.efi.wls.aa.loen	AktivitetBase	dk.skat.efi.wls.aa.common
ReplayGuard	dk.skat.efi.wls.aa.loen	AktivitetConditionChecker	dk.skat.efi.wls.aa.common
Sendmeddelelseomanmodningomloenoplysninger	dk.skat.efi.wls.aa.loen	AktivitetFilterExecuter	dk.skat.efi.wls.aa.common
SendmeddelelseomanmodningomloenoplysningerImpl	dk.skat.efi.wls.aa.loen	AktivitetFordringList	dk.skat.efi.wls.aa.common
Sendvarselomloenindeholdelse	dk.skat.efi.wls.aa.loen	AktivitetsAfviklerContext	dk.skat.efi.wls.aa.common
SendvarselomloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	AktivitetsAfviklerContextImpl	dk.skat.efi.wls.aa.common
Sendvarslingomstigningafloenindeholdelse	dk.skat.efi.wls.aa.loen	IAktivitet	dk.skat.efi.wls.aa.common

SendvarslingomstigningafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	IConditionCheckerResult	dk.skat.efi.wls.aa.common
Sletloenindeholdelse	dk.skat.efi.wls.aa.loen	IFilterExecuterResult	dk.skat.efi.wls.aa.common
SletloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	RuleCommonData	dk.skat.efi.wls.aa.common
Stopvarselogbooksagsbehandler	dk.skat.efi.wls.aa.loen	FordringTilfoejFjernAktivitet	dk.skat.efi.wls.aa.common.delegate
StopvarselogbooksagsbehandlerImpl	dk.skat.efi.wls.aa.loen	FilterData	dk.skat.efi.wls.aa.common.filter
Udskydivaerksaettelseafloenindeholdelse	dk.skat.efi.wls.aa.loen	IFilter	dk.skat.efi.wls.aa.common.filter
UdskydivaerksaettelseafloenindeholdelseImpl	dk.skat.efi.wls.aa.loen	IFilterResult	dk.skat.efi.wls.aa.common.filter
Udskydstopgrundetindkomsttype	dk.skat.efi.wls.aa.loen	PayloadChecker	dk.skat.efi.wls.aa.common.payload
UdskydstopgrundetindkomsttypeImpl	dk.skat.efi.wls.aa.loen	AbstractRule	dk.skat.efi.wls.aa.common.rules
VenteTilstandState	dk.skat.efi.wls.aa.loen	IResult	dk.skat.efi.wls.aa.common.rules
MeddelelseSendConverter	dk.skat.efi.wls.aa.loen.meddelelse	IRule	dk.skat.efi.wls.aa.common.rules
MeddelelseSendConverterImpl	dk.skat.efi.wls.aa.loen.meddelelse	RuleBOBBehandlingNyeFordringer	dk.skat.efi.wls.aa.common.rules
AfgoerelseCreator	dk.skat.efi.wls.aa.loen.model	RuleBooleanTest	dk.skat.efi.wls.aa.common.rules
AfgoerelseCreatorImpl	dk.skat.efi.wls.aa.loen.model	RuleData	dk.skat.efi.wls.aa.common.rules
IndsatsAfviklerImpl	dk.skat.efi.wls.ia	RuleErIndkomsttypeGyldigForIndsatsUndertype	dk.skat.efi.wls.aa.common.rules
IndsatsGraf	dk.skat.efi.wls.ia.indsatsgrafer	RuleFordringerDenEnkelteSaldoSkalOverstigeMinimumsbeloeb	dk.skat.efi.wls.aa.common.rules
IndsatsGrafBETALINGSORDNING	dk.skat.efi.wls.ia.indsatsgrafer	RuleFordringerDenEnkelteSaldoSkalOverstigeMinimumsbeloebFordringstype	dk.skat.efi.wls.aa.common.rules
IndsatsGrafLOENINDEHOLDELSE	dk.skat.efi.wls.ia.indsatsgrafer	RuleFordringerErOmfattetAfAndreIkkeTilladteIndsatser	dk.skat.efi.wls.aa.common.rules
IAIndsatsDAO	dk.skat.efi.wls.ia.da.dao	RuleFordringerErOmfattetBetalingsordning	dk.skat.efi.wls.aa.common.rules
IAIndsatsDAOImpl	dk.skat.efi.wls.ia.da.dao	RuleFordringerSamledeSumSkalOverstigeMinimumsBeloeb	dk.skat.efi.wls.aa.common.rules

IAIndsats	dk.skat.efi.wls.ia.da.domain.indsatser	RuleFordringerSamledeSumSkalOverstigeMinimumsBeloebFodringstype	dk.skat.efi.wls.aa.common.rules
IAIndsatsBetalingsordning	dk.skat.efi.wls.ia.da.domain.indsatser	RuleFordringerSkalHaveEnFordringsArt	dk.skat.efi.wls.aa.common.rules
IAIndsatsLoenindeholdelse	dk.skat.efi.wls.ia.da.domain.indsatser	RuleFordringerSkalHaveEnGyldigFundamentdato	dk.skat.efi.wls.aa.common.rules
IATilstand	dk.skat.efi.wls.ia.da.domain	RuleFordringerSomSkalFjernesFraAndreIkkeTilladteIndsatser	dk.skat.efi.wls.aa.common.rules
IAAktivitetsType	dk.skat.efi.wls.ia.da.domain	RuleFordringersSamledeSumSkalOverstigeGivetBeloeb	dk.skat.efi.wls.aa.common.rules
AngivelseStatus	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleFordringerTidligereIndberettet	dk.skat.efi.wls.aa.common.rules
EIndkomstBestillingStatus	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleIndsatsHaeftelsesforholdMellemKundeOgFordringTilladt	dk.skat.efi.wls.aa.common.rules
IABeregningsGrundlag	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleIndsatsHarIkkeIndsatsTypeITilstand	dk.skat.efi.wls.aa.common.rules
IALoenAngivelseBestilling	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleIndsatsTilladtForValgteFordringTyper	dk.skat.efi.wls.aa.common.rules
IALoenindeholdelseAfgoerelse	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKOBFristOverskredetVarselOmIndberetning	dk.skat.efi.wls.aa.common.rules
IALoenindeholdelseAfgoerelseFordring	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKOBFristOverskredetVarselOmIndberetningGyldighedsperiode	dk.skat.efi.wls.aa.common.rules
IALoenindeholdelseAngivelse	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKundeErPerson	dk.skat.efi.wls.aa.common.rules
IALoenindeholdelseFordring	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKundeErPersonOgHarCprNummer	dk.skat.efi.wls.aa.common.rules
IALoenindeholdelseReplay	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKundeHarAlleredeIndsatsMedSammeUndertype	dk.skat.efi.wls.aa.common.rules
Indberetningsart	dk.skat.efi.wls.ia.da.domain.indsatser.loenindeholdelse	RuleKundeSkalHaveBetalingsevneTilLoenindeholdelse	dk.skat.efi.wls.aa.common.rules

IABeloeb	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleKundeSkalHaveEnAdresse	dk.skat.efi.wls.aa.common.rules
IAEnkeltmandsVirkReference	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleKundeSkalHaveEnDanskAdresse	dk.skat.efi.wls.aa.common.rules
IAFordringRef	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleMinimumAlder	dk.skat.efi.wls.aa.common.rules
IAForventetIndbetaling	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleNot	dk.skat.efi.wls.aa.common.rules
IAHaeftelseForaeldelse	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleResult	dk.skat.efi.wls.aa.common.rules
IAHaendelse	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleUdlaegFordringerSidsteRettidigeBetalingsdatoForTilsigelseTilladt	dk.skat.efi.wls.aa.common.rules
IAIdentifikator	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleUdlaegFordringerTilladtForFortrinsberettedeFordringstyper	dk.skat.efi.wls.aa.common.rules
IAIndberetningsreference	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleUdlaegTilsigelsesAdresseSkalVaereDanskOgHaveEtPostnummer	dk.skat.efi.wls.aa.common.rules
IAKontakt	dk.skat.efi.wls.ia.da.domain.indsatser.typer	RuleUdlaegTilsigelseSkalHaveAngivetTilsigelsesform	dk.skat.efi.wls.aa.common.rules
IAKundeRef	dk.skat.efi.wls.ia.da.domain.indsatser.typer	DPFacade	dk.skat.efi.wls.aa.common.service
IAKundeRefFordringRef	dk.skat.efi.wls.ia.da.domain.indsatser.typer	DPFacadeImpl	dk.skat.efi.wls.aa.common.service
AktivitetSekvensNummerGenerator	dk.skat.efi.wls.aa.da.dao	KFIFacade	dk.skat.efi.wls.aa.common.service
AktivitetSekvensNummerGeneratorImpl	dk.skat.efi.wls.aa.da.dao	KFIFacadeImpl	dk.skat.efi.wls.aa.common.service

Table 27 Code Base for Automated Analysis

Code Samples Manually Reviewed

Area	Sub-area	packaged	Class
CORE	Treatments	dk.skat.efi.wls.ia	IndsatsAfviklerImpl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions	AbstractForretningsFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions	AbstractSystemFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	HaendelseFormatFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	IndsatsFindesEjFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	IndsatsIDFormatFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	IndsatsTypeInstantieringFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	IndsatsTypeMismatchFejl

CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	IndsatsTypeUkendtFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	KundeFindesEjFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	ResultatIkkeMatchetFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.AbstractForretningsFejl	SporSkabelonIndsatsSkabelonIDFormatFejl
CORE	Treatments	dk.skat.efi.wls.ia.exceptions.systemfejl	DAOFejl
CORE	Treatments	dk.skat.efi.wls.ia.factories	IIndsatsFactory
CORE	Treatments	dk.skat.efi.wls.ia.factories	IndsatsFactoryImpl
CORE	Treatments	dk.skat.efi.wls.ia.helpers	GetterWrappers
CORE	Treatments	dk.skat.efi.wls.ia.helpers	UtilityMethods
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGraf
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafBETALINGSORDNING
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafBOBEHANDLING
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafBOEDEFORVANDLSTRAF
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafERKENDFORDRING
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafHENSTAND
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafKREDITOPLYSBUREAU
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafKUNDEMOEDE
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafLOENINDEHOLDELSE
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafMANUELSAGSBEHANDL
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafRYKKER
CORE	Treatments	dk.skat.efi.wls.ia.indsatsgrafer	IndsatsGrafUDLAEG
CORE	Treatments	dk.skat.efi.wls.ia.management	AngivelsesoperationerVO
CORE	Treatments	dk.skat.efi.wls.ia.management	BegrundelsesVO
CORE	Treatments	dk.skat.efi.wls.ia.management	EIndkomstHentStatusVO
CORE	Treatments	dk.skat.efi.wls.ia.management	IaRestAPI
CORE	Treatments	dk.skat.efi.wls.ia.management	IndsatsParameterVO
CORE	Treatments	dk.skat.efi.wls.ia.management	KundeIndsatserVO
CORE	Treatments	dk.skat.efi.wls.ia.methodobjects	StopProcesseringAfHaendelse
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	FlytSporSkabelonIndsatsSkabelonParametreTilIndsatsNiveau.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentBoBehandlingsType.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentFordringIDerForIndsatsID.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentFundamentOplysningerForFordringerKOB.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIAIndsatsOgKundeViaKorrelationId.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsDataForFordringForKunde.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsDataForIndsatser.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsDataForKunde.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsersAktiveFordringerForKundeOpdeltPrIndsatsId.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsIDerForFordringID.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentIndsatsTyperForIndsatser.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentRetsafgiftDatoUdlaeg.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentTillaegsafgiftDatoUdlaeg.java

CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentTilstandsNavnfraTilstandsID.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentUdlaegBladInfoViaAktivId.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	HentUdlaegBladInfoViaIndsatsId.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	IForretningsProcess.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	IndsatsTypeList.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	PaakravsSkrivelseAkterAfskriv.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	PaakravsSkrivelseAkterAfskrivI.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	RetsAfgiftUdlaegBeregn.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	SignalerHaendelse.java
CORE	Treatments	dk.skat.efi.wls.ia.methodsobjects	StartIndsats.java
CORE	Treatments	dk.skat.efi.wls.ia.services;	ErkendFordringSynkronWrapper
CORE	Treatments	dk.skat.efi.wls.ia.validators	Validator
CORE	Process Engine	dk.skat.efi.wls.im.api	SporAdminApi.java
CORE	Process Engine	dk.skat.efi.wls.im.api	SporafviklerApi.java
CORE	Process Engine	dk.skat.efi.wls.im.api	SporskabelonAdminApi.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	AnbefaletSporSkabelonHaendelseModtagVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	HaendelseModtagSvarVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	HaendelseModtagVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	IndsatsTypeOgUndertypeVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	IndsatsVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	KundeDataVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	MultiHaendelseModtagVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	SporHistorikVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	SporRegelVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	SporskabelonInfoVO.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	SporVo.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain	VentendeHaendelseVO.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain.sporskabelon	SporSkabelonIndsatsParameterVO.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain.sporskabelon	SporSkabelonIndsatsSkabelonVO.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain.sporskabelon	SporSkabelonSporRegelVO.java
CORE	Process Engine	dk.skat.efi.wls.im.api.domain.sporskabelon	SporSkabelonVO.java
CORE	Process Engine	dk.skat.common.exceptions	SkatSystemMultiopretHaendelseException.java
CORE	Process Engine	dk.skat.common.exceptions	UnrecoverableStateException.java
CORE	Process Engine	dk.skat.efi.im	TransaktionsIdParser.java
CORE	Process Engine	dk.skat.efi.im.aktering	Aktering.java
CORE	Process Engine	dk.skat.efi.im.aktering	AkteringFactory.java
CORE	Process Engine	dk.skat.efi.im.aktering	AkteringImpl.java

CORE	Process Engine	dk.skat.efi.im.aktering	TitleAndTextResolved.java
CORE	Process Engine	dk.skat.efi.im.api	IMSporConverter.java
CORE	Process Engine	dk.skat.efi.im.api	MultiHaendelseModtagVo2DomainConverter.java
CORE	Process Engine	dk.skat.efi.im.api	SporAdminApiImpl.java
CORE	Process Engine	dk.skat.efi.im.api	SporafviklerApiImpl.java
CORE	Process Engine	dk.skat.efi.im.api	SporVoConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	AdresseAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	AnbefaletSporSkabelonConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BeloebConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BEODaekningAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BEORateAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BerostilLoenindeholdelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneFaldetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneFaldetVarigtConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneNulConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneSBetalEvneAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneSLFaldetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneSLStegetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneStegetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalEvneStegetVarigtConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalingOrdningOprettetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BetalingsordningMisligeholdtConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSAfsoningAflysConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSAfsoningOpdaterConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSGensendVarselConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSKorrektionSendConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSOpdaterPolitikredsConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSSendAnmodningConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSSendVarselConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BFSVarselAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BobGemKontaktConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BobSletKontaktConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BookingSvarConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BosagAendrAutomatiskConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	BosagAendrConverter.java

CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ErkendFordringFristOverskredetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ErkendFordringGenstartConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ErkendFordringKundehenvendelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ErkendFordringSagsbehandlerErkenderConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ETLConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	FordringOprettetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	FordringSaldoAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ForkyndelseDatoAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	FristOverskredetCirkulaerskrivelseEjModtagetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	FristOverskredetModtagelseAfAdkomsterklaeringConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	GenoptagSporSkifteConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaeftelseForaeldelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseCommonAttributesConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseConverterImpl.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseFilterConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseModtagConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HaendelseModtagConverterImpl.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	HenstandAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	IHaendelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	IMETLAnmeldelseStatusCheckConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	IndkomsttypeAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	IndsatsFordringFjernetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	IndsatsFordringTilfoejConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	KFIAdresseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	KOBVarslFristAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	KundemoedeAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	KundemoedeGennemfoertConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	LoenIndholdelseBegrundelseConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	LoenIndholdelseGensendIvaerksaetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MeddelelseIkkeModtagetConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MeddelelseIkkeSendtConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MeddelelsePakkeConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MoedeAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MultiHaendelseModtagConverter.java

CORE	Process Engine	dk.skat.efi.im.converters.haendelser	MultiHaendelseModtagConverterImpl.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	NoPayloadConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	OpgaveOpretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	OpretOpgavePayloadConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	RykBetalingsFristAendretConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	SagsbehandlerErkenderConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	ScoringConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	StartIndsatsConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	StopIndsatsConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	StopSporSkifteConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivAndelsboligSendRykkerConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivAndelsboligTinglysningAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivAndelsboligTinglysningFristAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivFjernConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivForaeldelseDatoAendrConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegAktivTinglysConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegBladDanConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegEjGennemfoertConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegKladdeGemConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegPolitieftersoegningAnmodConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.haendelser	UdlaegTilsigelseSendConverter.java
CORE	Process Engine	dk.skat.efi.im.converters.helpers	AdresseConverter.java
CORE	Process Engine	dk.skat.efi.im.dao	HaendelseDao.java
CORE	Process Engine	dk.skat.efi.im.dao	HaendelseDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	HaendelseFilterDao.java
CORE	Process Engine	dk.skat.efi.im.dao	HaendelseFilterDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	IndsatsDao.java
CORE	Process Engine	dk.skat.efi.im.dao	IndsatsDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	KundeDao.java
CORE	Process Engine	dk.skat.efi.im.dao	KundeDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	SporDao.java
CORE	Process Engine	dk.skat.efi.im.dao	SporDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	SporRegelDao.java
CORE	Process Engine	dk.skat.efi.im.dao	SporRegelDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	SporSkabelonDao.java

CORE	Process Engine	dk.skat.efi.im.dao	SporSkabelonDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dao	SporSkabelonIndsatsSkabelonDao.java
CORE	Process Engine	dk.skat.efi.im.dao	SporSkabelonIndsatsSkabelonDaoImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	FremtidigHaendelseCheckerTask.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	FremtidigHaendelseCheckerTaskImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	FremtidigHaendelseProcessor.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	FremtidigHaendelseProcessorImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelseRecordLogger.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelseRecordLoggerImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWork.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWorkExecuter.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWorkExecuterImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWorkTaskImpl.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWorkTimestampUpdater.java
CORE	Process Engine	dk.skat.efi.im.dispatchers.haendelse	HaendelsesWorkTimestampUpdaterImpl.java
CORE	Process Engine	dk.skat.efi.im.domain	IMGraenseZoneVaerdierType.java
CORE	Process Engine	dk.skat.efi.im.domain	IMIndsats.java
CORE	Process Engine	dk.skat.efi.im.domain	IMKundeData.java
CORE	Process Engine	dk.skat.efi.im.domain	IMKundeLaas.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSpor.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSporRegel.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSporRegelType.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSporSkabelon.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSporSkabelonIndsatsParameter.java
CORE	Process Engine	dk.skat.efi.im.domain	IMSporSkabelonIndsatsSkabelon.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	BEODaekningAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	BEORateAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	FejletHaendelseVO.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	FordringAendret.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMAdresseAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMAendretAktiv.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMAnbefaletSporSkabelonHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBerostilLoenindeholdelseHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneFaldet.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneFaldetVarigt.java

CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneNul.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneSBetalEvneAendret.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneSLFaldet.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneSLSteget.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneSteget.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalEvneStegetVarigt.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalingOrdningOprettetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalingsordningMisligeholdtHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBetalingsordningMisligeholdtVO.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSAfsoningAflysHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSAfsoningOpdaterHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSGensendVarselHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSKorrektionSendHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSOpdaterPolitikredsHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSSendAnmodningHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSSendVarselHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBFSVarselAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBobAendreBosagAutomatiskHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBobAendreBosagHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBobGemKontaktHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBobOpgOpretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBobSletKontaktHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMBookingSvarHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMErkendFordringFristOverskredetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMErkendFordringGenstartHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMErkendFordringKundehenvendelseHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMETLAnmeldelseStatusCheckHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMETLAnmeldelseSvarHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMFordringOprettetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMFordringSaldoAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMFristOverskredetCirkulaerskrivelseEjModtagetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMFristOverskredetModtagelseAfAdkomsterklaeringHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMGenoptagSporSkifteHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaeftelseAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaeftelseForaeldelseHaendelseVO.java

CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelseFilter.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelseFilterUndtagelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelseRecord.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelseTilIndsatsOpgave.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHaendelseTilIndsatsOpgaveComparator.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMHenstandAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMIndkomsttypeAendret.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMIndsatsFordringFjernetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMIndsatsFordringTilfoejHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMKreditoplysningsbureauVarselFristAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMKundeHaeftelseForaeldelseHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMKundemoedeAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMKundemoedeGennemfoertHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMLoenIndholdelseBegrundelseHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMLoenIndholdelseGensendIvaerksaetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMMeddelelseIkkeModtagetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMMeddelelseIkkeSendtHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMMeddelelsePakkeHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMModtagerReference.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMOpgaveOpretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMRykkerBetalingsFristAendretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMSagsbehandlerErkenderHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMScoringHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMStartIndsatsHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMStartIndsatsOpgave.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMStopIndsatsHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMStopIndsatsOpgave.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMStopSporSkifteHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivAndelsboligAdkomsterklaeringModtagetHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivAndelsboligSendRykkerHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivAndelsboligTinglysningFristAendrHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivFjernHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivForaeldelseDatoAEndrHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegAktivTinglysHaendelse.java

CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegbladDanHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegEjGennemfoertHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegForkyndelsesDatoAendrHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegKladdeGemHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegMoedeAendrHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegPolitieftersoegningAnmodHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	IMUdlaegTilsigelseSendHaendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	KFIAdresse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	MeddelelsePakkeBilag.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	MeddelelsePakkeOpkraevBeloeb.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	MeddelelsePakkeType.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser	RessourceBookInfo.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	Beloeb.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	BobehandlingKontakt.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	EFIEkstraInfo.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	EFIKunde.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	FordringAfbetaling.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	FordringBeloebInfo.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	Henvendelse.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	IMBEOEkstraInfo.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	IMDividende.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	KundeStruktur.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	Rate.java
CORE	Process Engine	dk.skat.efi.im.domain.haendelser.details	SamarbejdPart.java
CORE	Process Engine	dk.skat.efi.im.enums	HaendelseFilterUploadStatus
CORE	Process Engine	dk.skat.efi.im.exception	IndsatsForsøgtSlettetMensHaendelseProcesseresException.java
CORE	Process Engine	dk.skat.efi.im.exception	CirkulaerIndsatsException.java
CORE	Process Engine	dk.skat.efi.im.exception	GentagetIndsatsTypeException.java
CORE	Process Engine	dk.skat.efi.im.exception	IngenIndsatserPaaSporetException.java
CORE	Process Engine	dk.skat.efi.im.exception	SporRegelPegerPaaAktivIndsatsException.java
CORE	Process Engine	dk.skat.efi.im.exception	SporRegelPegerPaaIkkeEksisterendeIndsatsException.java
CORE	Process Engine	dk.skat.efi.im.factory	SporFactory.java
CORE	Process Engine	dk.skat.efi.im.factory	SporFactoryImpl.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	EFINotFutureDateException.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	HaendelseAdministration.java

CORE	Process Engine	dk.skat.efi.im.haendelseadmin	HaendelseAdministrationImpl.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	MultiOpretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	MultiOpretHaendelseImpl.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	OpretHaendelse.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	OpretHaendelseImpl.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	SletHaendelse.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	SletHaendelseImpl.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	TjekOmSagsbehHaendelseUnderBehandling.java
CORE	Process Engine	dk.skat.efi.im.haendelseadmin	TjekOmSagsbehHaendelseUnderBehandlingImpl.java
CORE	Process Engine	dk.skat.efi.im.management	ImRestAPI.java
CORE	Process Engine	dk.skat.efi.im.sporadmin	SporService.java
CORE	Process Engine	dk.skat.efi.im.sporadmin	SporServiceImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	HaendelseFilterHaandtering.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	HaendelseFilterHaandteringImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	HaendelsesHaandtering.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	HaendelsesHaandteringImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	OpgaveAfvikler.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	OpgaveAfviklerImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	Opgaveopretter.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	OpgaveopretterImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	SpecialHaandtering.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	SporSkifteKontrol.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	SporSkifteKontrolImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	SporSkifter.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	SporSkifterImpl.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	UdestaaendeTjek.java
CORE	Process Engine	dk.skat.efi.im.sporafvikler	UdestaaendeTjekImpl.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonadmin	SporskabelonAdminImpl.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonadmin	SporSkabelonConverter.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonvalg	GraenseZoneKontrol.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonvalg	GraenseZoneKontrolImpl.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonvalg	SporSkabelonValg.java
CORE	Process Engine	dk.skat.efi.im.sporskabelonvalg	SporSkabelonValgImpl.java
CORE	Process Engine	dk.skat.efi.im.vo	HaendelseAndKundeStruktur.java
CORE	Process Engine	dk.skat.efi.im.vo	HaendelseFilterList.java

CORE	Process Engine	dk.skat.efi.im.vo	HaendelseFilterUploadRapport.java
CORE	Process Engine	dk.skat.efi.im.vo	HaendelseFilterVo.java
CORE	Treatments	dk.skat.efi.wls.aa.beo	BEOBookRessourceTilMeddelelseOmAEndretBetalingsordningImpl
CORE	Treatments	dk.skat.efi.wls.aa.beo	BEOBookSagsbehandlerTilMeddelelsesfejlImpl
CORE	Treatments	dk.skat.efi.wls.aa.beo	BEOBookSagsbehandlerTilOpfoelgningImpl
CORE	Treatments	dk.skat.common.transactionstrategies	TryAgainWithStatusLookupStrategy
Peripheral	BeBB	dk.skat.efi.wls.bebb.api	BebbApiImpl
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	AbstractBeregnerDecorator
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	AbstractBetalingsEvneBeloeb
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	BeloebLedigTilReservation
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	BeregnAktuelLoenIndProcent
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	Beregner
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	BeregnetAarsindkomst
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.betalingevne	BeregnetLoenIndProcent
Peripheral	BeBB	dk.skat.efi.wls.bebb.beregning.budget	KundeBudget
Peripheral	BeBB	dk.skat.efi.wls.bebb.betalingevne	BebbKundeDataVoConverter
Peripheral	BeBB	dk.skat.efi.wls.bebb.betalingevne	BeregningGrundlagVoConverter
Peripheral	BeBB	dk.skat.efi.wls.bebb.betalingevne	BetalingevneAdmin
Peripheral	BeBB	dk.skat.efi.wls.bebb.betalingevne	BetalingEvneAsynkronHentWorkExecuterImpl
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BebbKundeDataVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BeregningGrundlagVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BetalingEvneVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BoernebudgetVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BudgetpostBarnVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BudgetpostLaanVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BudgetpostVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BudgetpostVoksenVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	BudgetVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	EjendomVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	KoeretoejVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	KundeBudgetterVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	NettoIndkomstPostListVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	NettoIndkomstPostVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	ReservationVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo.alt	SBetalingEvneVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	BebbKundeIdentifikation

Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	BebbKundeIdentifikationByKFIKundeId
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	BebbKundeIdentifikationCreate
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	dk.skat.efi.wls.bebb.api.vo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	BetalingEvneMultiHentKundeListeVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	BetalingEvneMultiHentKundeVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	EjendomHaendelseKundeVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	ForsoergerpligtGenberegnVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	HenvendelsesformVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	IndkomsttypeVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	KoeretoejHaendelseKundeVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	dk.skat.efi.wls.bebb.api.vo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api.vo	SReservationAendreVo
Peripheral	BeBB	dk.skat.efi.wls.bebb.api	BebbApi
Peripheral	BeBB	dk.skat.efi.wls.bebb.opretkunde	NettoIndkomstBeregnerImpl
Peripheral	BeBB	dk.skat.efi.wls.bebb.opretkunde	CustomerAsyncronCreationWorkerImpl
Peripheral	BeBB	dk.skat.efi.wls.bebb.betalingevne	BetalingEvneAsynkronHentWorkExecuterImpl

Table 28 Code Samples Manually Reviewed

9.2.4 Test

There are no “EFI” test cases, separate from DMI performed by SKAT
Test cases and test reports performed by the EFI development team were requested but not provided.

9.2.5 Change Requests

Modtag Fordring (Receive Claim)	
Change Request ID	Result of Change Request
QC 694	Affected trace score
QC 1645	Affected trace score
QC 10741	Affected trace score
Kundesaldi (Client Account Balance)	
Change Request ID	Result of Change Request
None	None
Betalingsordninger (Payment Plans)	
Change Request ID	Result of Change Request
None	None
ønindeholdelse (Salary Deduction)	
Change Request ID	Result of Change Request
None	None

Table 29 Change Requests DMI

9.2.6 Analysis

9.2.6.1 DMI Requirements & Use Cases Analysed

Description	Date / Version
EFI 02 Leverandørens kravopfyldelse FA v1_00	N/A
EFI 02 Leverandørens kravopfyldelse S v1_00	N/A
Bilag 3 4 3_Use case og supplerende beskrivelser for DMI	August 2010

Bilag 3.4 Use cases	2010

Table 30 DMI Requirements & Use Cases Analyses

9.2.7 Design

9.2.7.1 DMI Design Specifications

Description	Date / Version
SKAT_Debitormotor_RTM-CIC_DMI_DMO_DMS_Technical_requirements_19 06 2014	19.06.2014
DM RTM-CIC Drift og Vedligehold v 19Juni2014	19.06.2014
DM DMI Funktionalitetsgruppering - Fordringer Modtag	08.10.2012
DM DMI Funktionalitetsgruppering Rente 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Betalingsordning_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Dækningsrækkefølge_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Dækningsrækkefølge_BT_20130116	08.10.2010
DM_DMI_Funktionalitetsgruppering_Fordringer_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Hæftelse_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Hæftelse_Forældelse_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Indbetalinger_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Modregning_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Stamdata_ 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Administration 08 OKT 2012_v1.00	08.10.2010
DM_DMI_Funktionalitetsgruppering_Processer _08 OKT 2012_ v1.00	08.10.2010
Bilag 3.6.2 Services udtræk for DMI	February 2010

Table 28 DMI Design Specifications

9.2.8 Code

DMI Code as of May 2015 was assessed

Program Name	Package
Z_MAST_CLAM_UPDATE	ZMAST
Z_OFFS_QUER_RUN	ZOFFS
ZACCH_CUST_CALC	ZACCO
ZACCO_SPEC_ALV	ZACCO
ZACCO_SPEC_ALV_FORMS	ZACCO
ZACCO_SPEC_ALV_SCR	ZACCO
ZACCO_SPEC_DATADEF_ALV	ZACCO
ZBC_DWL_TRANSPORT_REQ	ZADMI
ZBC_SHOW_VERSION	ZADMI
ZCLAM_BO_BANK_UPDATE	ZCLAM
ZCLAM_FIX_NOTI_RECE	ZCLAM
ZCLAM_FIX_NOTI_RECE_METHODS	ZCLAM
ZCLAM_INTE_NOTIFY	ZCLAM
ZCLAM_NOCO_DIFF_DUPLICATES	ZCLAM
ZCLAM_NOCO_DIFF_DUPLICATES_F01	ZCLAM
ZCLAM_NOCO_DIFF_NOTI_CREA	ZCLAM
ZCLAM_NOCO_DIFF_NOTI_CREA_FORM	ZCLAM
ZCLAM_NOTI_CREATE	ZCLAM
ZCLAM_NOTI_DIFF	ZCLAM
ZCLAM_NOTI_DIFF_01	ZCLAM
ZCLAM_RDI_COPY	ZCLAM
ZCLAM_RDI_COPY_FORMS	ZCLAM
ZCLAM_REP_SUM	ZCLAM
ZCLAM_SETL_CORRECT_FIX	ZCLAM
ZCLAM_SETL_CORRECT_FIX_EXECF01	ZCLAM
ZCLAM_SETL_NOTIFY	ZCLAM
ZCLAM_SETL_NOTIFY_METHODS	ZCLAM
ZCLIA_AGIN_ANALYSE_ANAL_DELETE	ZCLIA
ZCLIA_AGIN_ANALYSE_ANAL_LOCK	ZCLIA
ZCLIA_AGIN_ANALYSE_ANAL_ROLLBA	ZCLIA
ZCLIA_AGIN_ANALYSE_ROLLBA_MAIN	ZCLIA
ZCLIA_AGIN_FIX	ZCLIA
ZCLIA_AGIN_FIX_DEL	ZCLIA
ZCLIA_AGIN_FIX_GET	ZCLIA
ZCLIA_AGIN_FIX_SELECTIONS_SF01	ZCLIA

ZCLIA_AGIN_FIX_VERIFY_PAR_IF01	ZCLIA
ZCLIA_CLIA_FIX_PEF	ZCLIA
ZCLIA_CLIA_FIX_PEF_STATISTIK	ZCLIA
ZCLIA_INTE_SPEC_ALV	ZCLIA
ZCLIA_INTE_SPEC_ALV_FORMS	ZCLIA
ZCLIA_INTE_SPEC_ALV_SCR	ZCLIA
ZCLIA_LOAD_COURTFEE	ZCLIA
ZCLIA_LOAD_DEPR	ZCLIA
ZCLIA_MONITORING	ZCLIA
ZCLIA_SERV_LIMIT_RECEIVE	ZCLIA
ZCLIA_SPEC_DATADEF_ALV	ZCLIA
ZCLIA_SPEC_SUM_UPDATE	ZCLIA
ZCOVE_IPAY_SIMU	ZCOVE
ZCOVE_IPAY_SIMU_DATA_DEF	ZCOVE
ZCOVE_REDO_IPAY	ZCOVE
ZCOVE_REDO_IPAY_OLD	ZCOVE
ZCOVE_REDO_IPAY_TOOLS	ZCOVE
ZCPEF_CLIA_EVAL	ZCLIA
ZFICO_RECKEY_AGGR_CLOSE	ZFICO
ZFICO_RECKEY_AGGR_INIT	ZFICO
ZFICO_RECONCILE_TABLES	ZFICO
ZFICO_RECONCILE_TABLES_CLASSES	ZFICO
ZFICO_RECONCILE_TABLES_FORMS	ZFICO
ZFICO_RECONCILE_TABLES_USERI01	ZFICO
ZFICO_WRSP_CALC	ZFICO
ZFPER_CALC_PERI	ZFICO
ZFPER_EVAL_VKONT	ZFICO
ZFPER_RECE_RECON	ZFICO
ZGENE_DOCU_FIX	ZGENE
ZGENE_DOCU_NETS_FIX	ZGENE
ZGENE_DW_EXTR_SELE	ZGENE
ZGENE_DW_EXTRACT	ZGENE
ZGENE_DW_KOBRA_SPECIAL	ZGENE
ZGENE_PERF_DISPLAY	ZGENE
ZGENE_PROC_MONI	ZGENE
ZGENE_SHOW_LOG	ZGENE
ZINST_CONV_LOAD	ZINST
ZINST_FIX_RATES	ZINST
ZINTE_BALANCE_FORWARD	ZINTE
ZINTE_MASS_PERIOD	ZINTE
ZIPAY_LOAD_REJECT	ZIPAY
ZIPAY_REJECT	ZIPAY
ZOPAY_RET_DUPL_FIX	ZOPAY

Table 31 DMI Code

9.2.9 Test

There are no "DMI" test cases, separate from EFI performed by SKAT.
Test cases and test reports performed by the development team were not requested.

9.2.10 Change Requests

Modtag Fordring (Receive Claim)	
Change Request ID	Result of Change Request
QC 694	Affected trace score
QC 1645	Affected trace score
QC 10741	Affected trace score
Kundesaldi (Client Account Balance)	
Change Request ID	Result of Change Request
DMI ÆA114	Affected two requirements, but did not affect their trace score
Betalingsordninger (Payment Plans)	
Change Request ID	Result of Change Request
None	None
Lønindeholdelse (Salary Deduction)	
Change Request ID	Result of Change Request
None	None

Table 32 Change Requests

10 Appendix: Accenture Delivery Methods

This report contains descriptions of and references to the Accenture Delivery Methodology (ADM). ADM is proprietary to Accenture and all descriptions of and references to ADM must be kept strictly confidential and the respective party of the report must be redacted prior to distribution.

ADM defines the project work that needs to be done and how that work can best be accomplished. Included within ADM are methods, estimators, and procedures.

- Proven processes, deliverables and techniques that enable global teams to define what to do and how to do it
- A comprehensive set of methods that supports multiple types of work (e.g., custom development, package implementations and outsourcing)
- Focused on the fundamentals and discipline, especially around program and project management
- Built on a common framework to promote growth of consistent skills
- Includes estimators for estimating level of effort
- Includes repeatable, step-by-step procedures to drive consistency
- Defines standard language, roles and deliverables

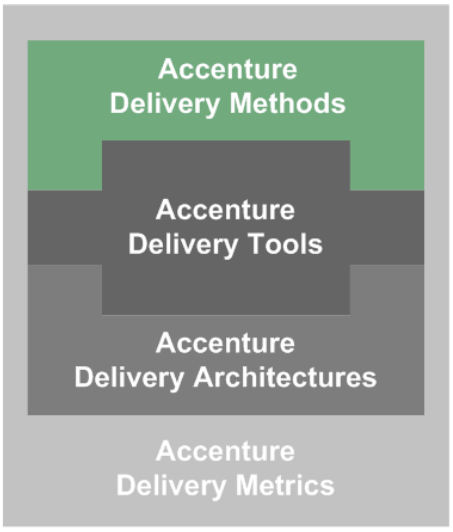

Accenture Delivery Methods
Accenture Delivery Tools
Accenture Delivery Architectures
Accenture Delivery Metrics

Figure 11 Accenture Delivery Suite

10.1 Components

ADM defines the project work that needs to be done and how that work can best be accomplished. Included within ADM are methods, estimators, and procedures.

- **Methods:** ADM defines the work to be done and how it can best be accomplished. There are 3 key types of content in the methodology: processes, work products, and roles. These 3 content types all reference each other to form the foundation of the methodology.

- **Estimators:** Enable you to create an estimate of work based on Methods' activities and tasks. Additionally, you can create a staffing estimate and plan using the roles from the Methods.

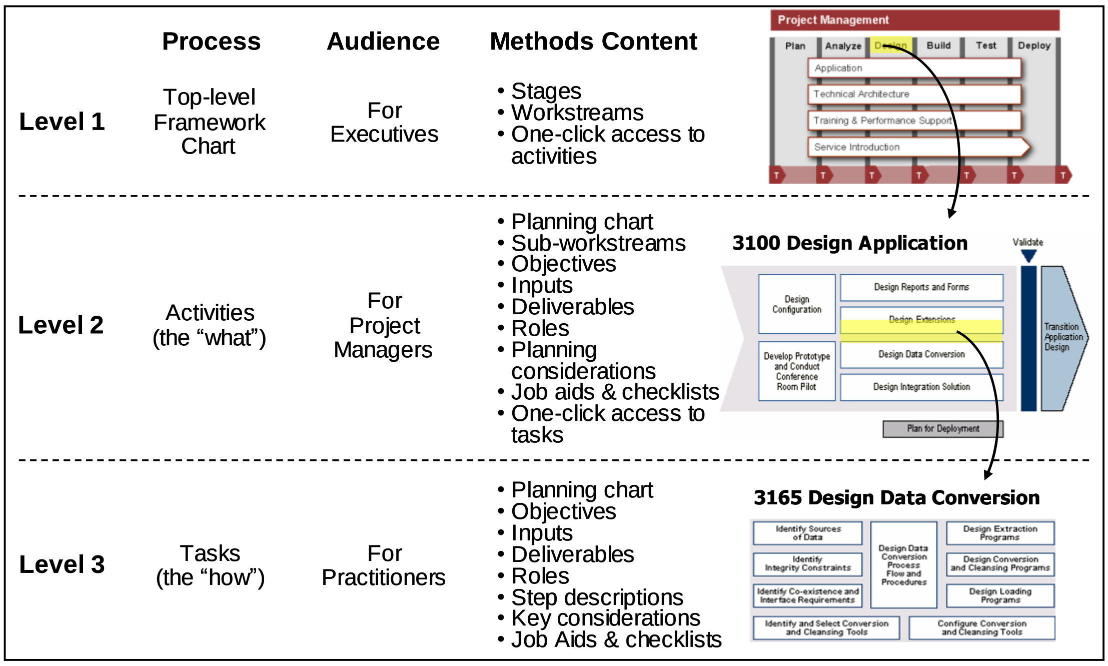
- **Procedures:** Step-by-step, role-based instructions for completing detailed functions. The tools automate procedures, making them standard and re-usable across Methods.

10.2 Comprehensive Coverage

The main benefit of ADM is its ability to be used to guide teams through daily project tasks and activities. There are multiple methods in each of four categories. All methods follow a common structure and approach. Each project chooses the method(s) that correspond with the type of work it is delivering. A method can be used alone, or in combination with other methods. Within these methods are processes, procedures, and work products that support and guide teams in delivering excellence.

10.3 Structure

The Methods provide a standard 3-level framework that you can drill down into. Within that framework are standard inputs, outputs, and roles associated with each process or task. From this common starting point, projects can tailor the processes and deliverables to meet their project-specific needs.



	Process	Audience	Methods Content
Level 1	Top-level Framework Chart	For Executives	• Stages • Workstreams • One-click access to activities
Level 2	Activities (the "what")	For Project Managers	• Planning chart • Sub-workstreams • Objectives • Inputs • Deliverables • Roles • Planning considerations • Job aids & checklists • One-click access to tasks
Level 3	Tasks (the "how")	For Practitioners	• Planning chart • Objectives • Inputs • Deliverables • Roles • Step descriptions • Key considerations • Job Aids & checklists

Project Management
Plan
Analyze
Design
Build
Test
Deploy
Application
Technical Architecture
Training & Performance Support
Service Introduction
3100 Design Application
Validate
Design Configuration
Design Reports and Forms
Design Extensions
Design Data Conversion
Design Integration Solution
Develop Prototype and Conduct Conference Room Pilot
Transition Application Design
Plan for Deployment
3165 Design Data Conversion
Identify Sources of Data
Identify Integrity Constraints
Identify Co-existence and Interface Requirements
Design Data Conversion Process Flow and Procedures
Design Extraction Programs
Design Conversion and Cleansing Programs
Design Loading Programs
Identify and Select Conversion and Cleansing Tools
Configure Conversion and Cleansing Tools

Figure 12 Levels in ADM Models

10.4 ADM for Custom Development

Overview

The Accenture Delivery Methods for Custom Development focus on the custom development of application solutions for our clients. They also support software package implementations where some degree of custom development is required.

The Accenture Delivery Methods for Custom Development are an application development methodology. This methodology supports business process analysis, application requirements and use case analysis, application's functional and technical design, technical architecture development, and the deployment of the application.

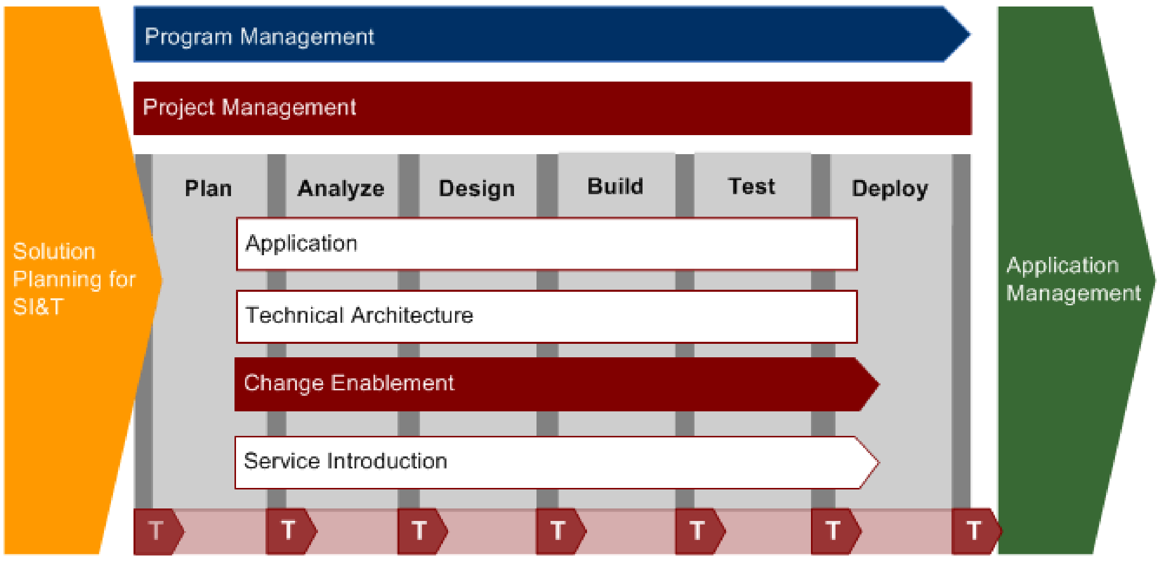


Figure 13 Accenture Delivery Methodology

Major components of this methodology include the following:

- **Plan.** This involves the up-front, project-level planning required to understand high-level requirements, define the application and technology blueprints, explore solution options, and define the solution delivery strategy.
- **Application.** This involves the tasks and deliverables needed to analyze, design, build, and test a custom-built application.
- **Technical Architecture.** This involves the tasks and deliverables needed to analyze, select and design, install and build, and test the development, execution, and operations environments.
- **Service Introduction.** This includes tasks and deliverables needed to ensure that the application has been developed to properly address the operational and support requirements.
- **Deploy.** This includes the tasks and deliverables needed to deploy the application to the users and transition the application management responsibilities to the support unit.