Luther Case Study

Project Ford

Supplier Claims Order Fulfillment Process

Luther Systems

1. Executive Summary

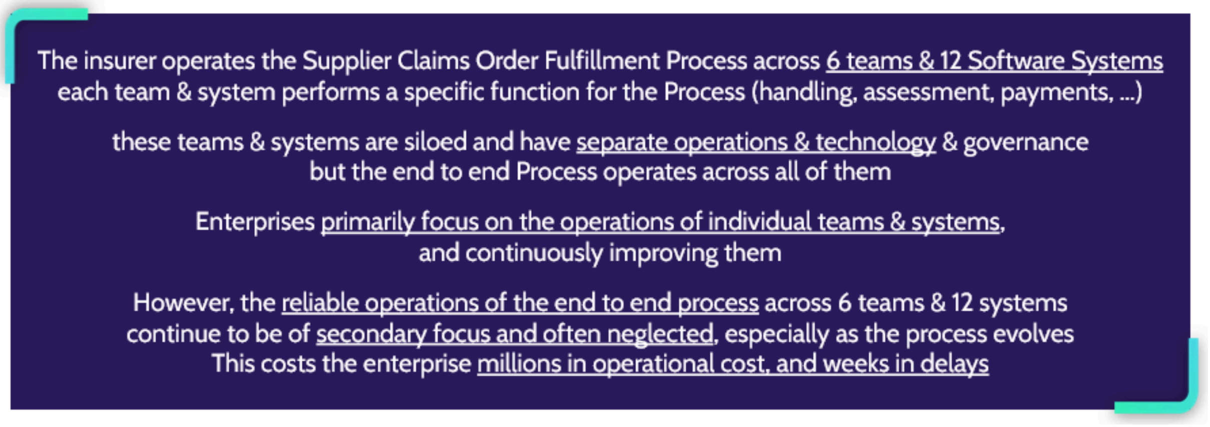
A leading FTSE 100 insurance company (Insurer) with tens of thousands of employees. They provide a range of insurance products, including general insurance, health insurance, life insurance and asset management, as well as retirement and savings solutions.

A major aspect of the Insurer's operations is fulfilling claims. For operational and financial efficiency, regulatory reporting, and to avoid fraudulent claims, each step of the claims process must be validated and approved. As one of the steps in completing the claims process, The Insurer fulfills claims from policy-holders by covering the cost of a replacement item or repair from one of a large group of suppliers. Managing suppliers is a key part of successful operations for this Insurer, and the supplier management market is worth almost $29 billion worldwide[1].

The Supplier Claims Order Fulfillment process has these steps: i) Policy holder notifies the Insurer of an incident, ii) The Insurer acknowledges claim and verifies claimant policy, iii) The Insurer's Claims Handling team sends claimant list of suppliers, iv) Claimant goes to a supplier, v) Supplier provides an assessment to the Insurer, vi) The Insurer's Supplier Assessment team requests clarifications, supplier responds, vii) The Insurer's Supplier Management team approves work, viii) Supplier's Finance team sends an invoice to the Insurer on completion, ix) The Insurer's Supplier Management team validate invoice, x) The Insurer's Payment team make payment to supplier & claim is closed.

Team	Software systems	Tasks
6	12	64

The Insurer operates the Supplier Claims Order Fulfillment process, as part of the claims value chain. This process operates across 6 teams & 12 software systems. To operate the process end-to-end, each function performs the same cycle of steps i) send data & info to the System, ii) receive response from System, iii) compute & validate response, iv) share & store execution of step, v) evaluate & initiate next step. For reliable operations, all teams & systems involved should operate the same end-to-end Process. They often don't! This leads to operational & technical challenges, which make process operations unreliable. The opportunity is providing a platform to reliably operate the end-to-end process, across all teams & systems involved.



The insurer operates the Supplier Claims Order Fulfillment Process across 6 teams & 12 Software Systems
each team & system performs a specific function for the Process (handling, assessment, payments, ...)

these teams & systems are siloed and have separate operations & technology & governance
but the end to end Process operates across all of them

Enterprises primarily focus on the operations of individual teams & systems,
and continuously improving them

However, the reliable operations of the end to end process across 6 teams & 12 systems
continue to be of secondary focus and often neglected, especially as the process evolves
This costs the enterprise millions in operational cost, and weeks in delays

[1]https://www.marketsandmarkets.com/Market-Reports/supply-chain-management-market-190997554.html

Traditional solutions to end-to-end process operations are unreliable & expensive. Enterprise Operations are generally function-first, they continue to improve functions & systems. Processes are considered secondary. The thinking is that if we have great functions & systems, the business can operate any process! Luther's platform is designed process-first, & primarily focuses on end-to-end processes. Reliable end-to-end process operations include consistent operation, and great functions & systems. Traditionally enterprises use bespoke connectors & local operations scripts for process operations, which are fragmented, siloed, and change separately, and so are ineffective for reliable process operations.

Luther's platform takes a process first approach to process operations
focusing on the reliable operations of the end to end process across all teams & systems,
instead of cobbling & stitching together the separate & siloed functions of 6 teams & 12 software systems

To remedy this, enterprises use automation tools. However they are ineffective at end-to-end process operations, due to their limited scope and scale, and stitching them together also doesn't solve the problem.

Luther's platform vertically integrates

distributed system technology
optimal resource allocation and management
real time event ordering and streaming (sharing)
deterministic event processing and execution

To make reliable end-to-end process operations possible.

Project Ford is a product built on the Luther platform using Deep Process Automation Technology to automate the process of claims fulfillment with suppliers. It processes the fulfillment of a claim from initiation to completion. The Luther Platform provides standard connectivity and a Common Operations Script shared by all participants. The platform reliably operates the end-to-end process across all teams and software systems from the common operations script.

Luther's unique value for reliable end-to-end Process Operations is providing

standard connectivity
a common operations script

across all teams and software systems.

Luther's unique value for reliable end-to-end Process Operations is providing i) standard connectivity & ii) a common operations script, across all teams & software systems. Luther's platform vertically integrates i) distributed system technology ii) optimal resource allocation & management, iii) real time event ordering & streaming, iv) deterministic event processing & execution, for reliable end-to-end process operations. Luther's platform does this by i) connecting systems to standard platform nodes, rather than to each other, and ii) teams & systems can change the common operations script but all teams & systems have to know & agree to the change, so all teams & systems involved operate the same end-to-end Process all the time!

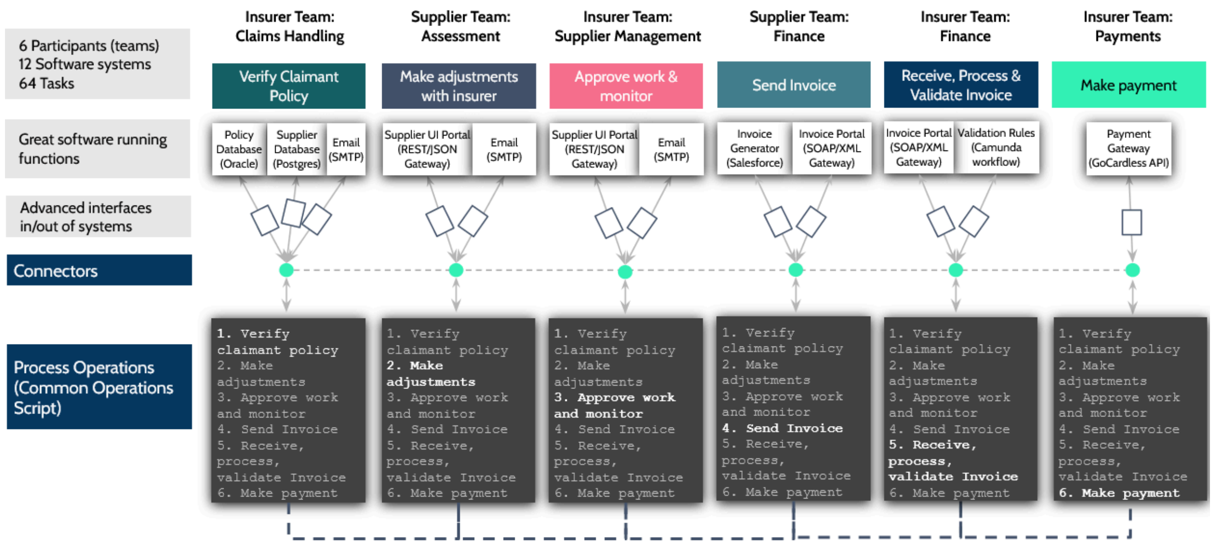

6 Participants (teams)
12 Software systems
64 Tasks
Great software running functions
Advanced interfaces in/out of systems
Connectors
Process Operations (Common Operations Script)
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy 2. Make adjustments 3. Approve work and monitor 4. Send Invoice 5. Receive, process, validate Invoice 6. Make payment

Fig 1. Project Ford Overview

To implement the platform, i) Luther's team mapped the Process, ii) Identified teams & software systems in the process, iii) allocated nodes (servers) to teams, iv) connected nodes to systems, v) set up the Platform on the nodes. vi) The Insurer's team along with Luther's team developed the Common Operations Script (code) for Process Operations, vii) the process went Live.

Process mapping 1 month	Platform setup 1 day	Implementation 2 months

Fig 2. Implementation details and timeline of Project Ford.

The results have been highly impactful. Thanks to increases in efficiency and the elimination of manual intervention, a process that traditionally took over a month can now be completed in a matter of days and operational costs have been reduced by 70%. Beyond the commercial results, this led to operational benefits in production, i) reliable operations across the process & over time, ii) Execution visibility, iii) reduction of reconciliation, iv) 5X smaller Ops teams, v) enforced compliance checks. Also, technical benefits during development, i) automated infrastructure., connectors and development environment setup, ii) focus only on development of process operations rather than technical setup and infrastructure, iii) consistent real time updates, iv) eliminate DevOps teams, v) 5X smaller Dev teams.

ROI 6X	Cost reduction 70%	Speed increase 10X

Fig 3. Estimated results of implementing the Platform for the Supplier Claims Order Fulfillment Process

Project Ford demonstrates a sleek, effective system built on the Luther platform to standardize and automate the supplier claims order fulfillment process. The network could be further expanded to encompass other suppliers, but the Luther platform could also be utilized to further streamline other areas of the leading Insurer's operations.

2. The Process

2.1. Process Operations

Different teams have different operations, rules and governance and they also utilize and operate a variety of software systems in different ways. Each system operates a specific function for the process. To operate the process end to end, each function performs the same cycle of steps: i) send data & information to the System, ii) receive response from the System, iii) compute & validate response, iv) share & store execution of step, v) evaluate & initiate next steps.

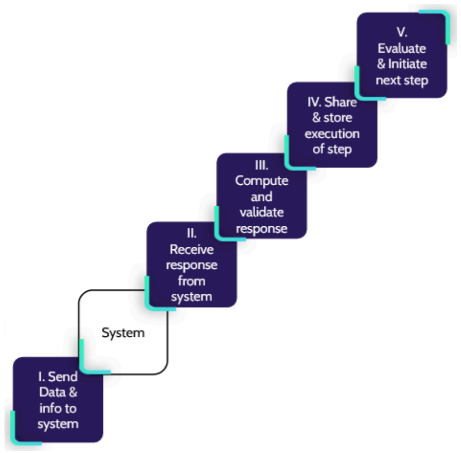

I. Send Data & info to system
System
II. Receive response from system
III. Compute and validate response
IV. Share & store execution of step
V. Evaluate & Initiate next step

Fig 4. These are the requirements that repeat for all functions across the end-to-end Process Operations.

Enterprises operate a set of specific functions based on their objective. For example, an insurance company's functions help it to provide insurance. While the functions and systems may change, the process remains the same. However, expecting processes to be efficient because of efficient individual tools simply does not work for enterprises. Luther empowers enterprises with a process-first approach.

Efficient software systems and functions are not enough to reliably operate a process end-to-end

To understand why efficient functions and tools do not create an efficient process, it is important to understand the following. Tasks are simple events that are localized to one team involving one or two software systems, for example retrieving data from a database. Workflows are more complex, involving 10-20 tasks between one or two teams and two or three software systems. An example of a workflow would be onboarding a new employee. Processes are complex, involving 50+ tasks, 3 or more teams and multiple software systems. Managing order fulfillment for customer claims from start to finish is a process.

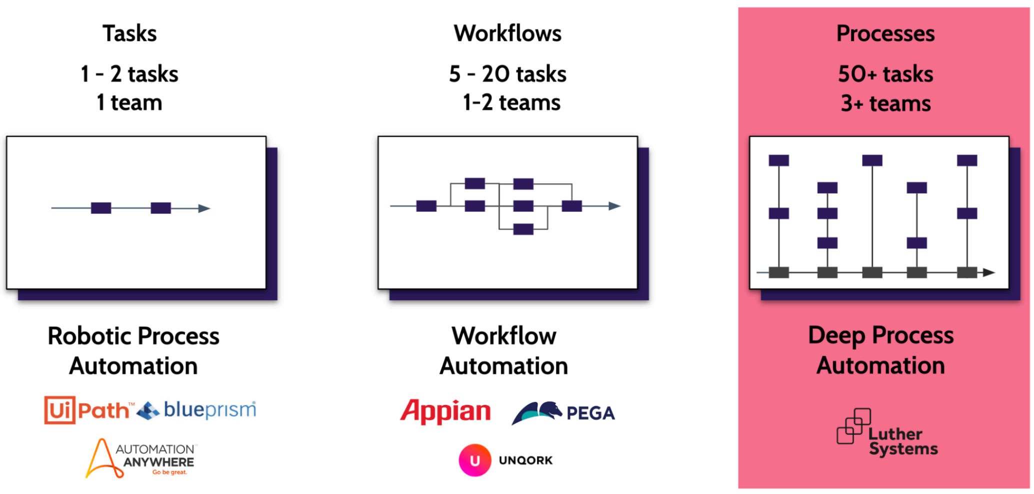


Fig 5. Different tools are used to automate different levels of complexity.

Generally, enterprise operations are function-driven i.e. enhancing the performance of individual functions (tasks and workflows). Tasks often have dedicated software systems and are operated by specific teams. By building and maintaining efficient and effective systems, almost any function can be efficiently operated. As a result, tasks are highly efficient. However, large enterprises operate thousands of processes each day, and each process is made up of many tasks. Enterprises generally have similar core processes, with some variation on the details. For example, every insurance company has the core processes of underwriting risk, collecting premiums and paying out claims.

"If we have great functions, services and systems, we're enabling the business to build and operate any process they want!"

Enterprises continue to optimize and improve, and incorporate better functions and systems. Example functions include claims handling, claims assessment, customer onboarding, finance, payments, settlement, fraud, compliance, reconciliation. Example software systems include databases, CRMs, RPA, Workflow tools, cloud services, microservices, data lakes, and others.

Enterprises naturally gravitate to a "function-first" view where processes are secondary to these functions and systems, as they are considered ever changing, and functions and systems can enable any process that the business may envision. The problem is, functions do not necessarily create an efficient process. Process-first thinking is required for this, and since the processes in many industries are regulated and unchanging, process-first thinking makes more sense.

Each enterprises generally operates a specific set of value chains and process, in particular in regulated industries, as explicitly stated by their primary activities. An insurance company insures !

For each enterprise most processes are already known and don't change.
For most processes, the majority of the process operations are already known and don't change.

It's time to take a Process first approach in the enterprise !

Luther's platform is designed process-first. For efficient enterprise operations, effective end-to-end process operations are as important as effective individual services and systems, and the Luther platform is designed especially for operating explicitly enumerated end-to-end process operations while still accounting for functions.

Enterprise Operations are generally function-first.

They continue to improve functions & systems. Processes are considered secondary.
If we have great functions and systems, the business can operate any process !

Luther's platform is designed process-first.

Primary focus on end-to-end processes.
Reliable end-to-end process operations include consistent operations, & great functions & systems

2.2. Supplier Claims Order Fulfillment in Context

Value streams are collections of processes organized by enterprises that operate specific enterprise objectives. In an insurance context, one example of a value stream could be all processes involved with "claims"[2]. Together, these can involve hundreds or thousands of tasks, many teams, and many software systems. Efficiently operating value streams can take up all operations of an enterprise.

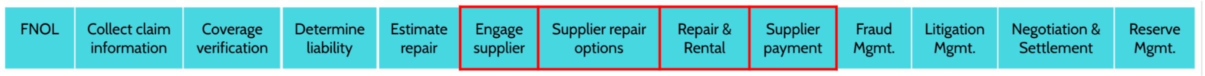


Fig 6. Claims is a value stream. It contains many processes, including supplier management

Supplier order fulfillment is a crucial activity of medium-to-large enterprises that work with multiple suppliers to deliver products to customers. Its primary goal is to ensure that a company's supply chain for its customers is efficient, cost-effective, and capable of meeting its operational needs. Globally, products and systems that aid in supplier order fulfillment are worth almost $29 billion, and this market is expected to grow by 11% each year[3]. An effective supplier claims order fulfillment process is critical to ensure smooth operations. Fast completion times on invoice processing ensures that both suppliers and customers are satisfied and help to avoid compliance violations. This process is carried out at immense scale by the largest insurance companies, who use multiple specialist suppliers providing different categories of product.

[2] https://www.insurancethoughtleadership.com/claims/rethinking-claims-value-chain
[3] 1. https://www.marketsandmarkets.com/Market-Reports/supply-chain-management-market-190997554.html

The multinational Insurer has many suppliers that provide replacement items to customers who have filed claims. The Insurer processes over 400,000 supplier invoices a year relating to household claims. Approximately 20% are put on hold as a result of existing validation processes. The average time for a supplier invoice to be settled is over 30 days. As the world becomes increasingly digitized, companies look to streamline their operations with efficient digital processes to automate administrative tasks. Currently, much of the Insurer's supplier claims order fulfillment process does operate digitally. However, owing to the large number of participants involved and the need for consistent operations across the participants involved, current automation solutions have been unable to effectively automate the operations of this process.

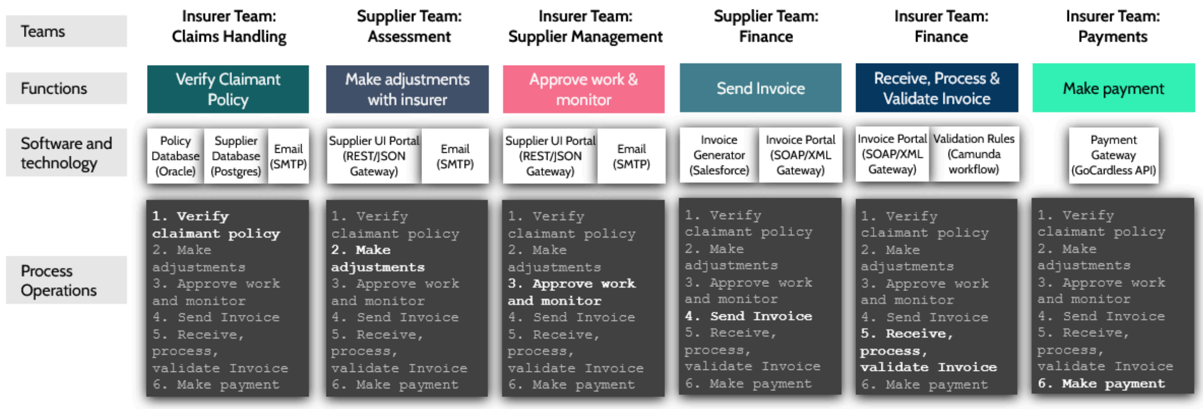
Despite this scale, the process is unstandardized and its operations are fragmented across different participants. To ensure consistent operations, compliance with regulations, avoidance of errors in fulfillments and payments, and prevention of fraud, claims, orders, invoices, and payments must be processed, validated, and approved by different teams. Each team performs a specific function within the end-to-end process.

2.3. Supplier Claims Order Fulfillment Process before

Team	Software systems	Tasks
6	12	64

Let's first discuss the process of processing a claim and the participants involved.

1. The policy holder notifies the insurer of an incident that may result in a claim
2. The insurer acknowledges the claim and verifies the claimant has a valid policy with them
3. The insurer's Claims Handling team provides claimant with list of relevant suppliers
4. The claimant goes to a supplier
5. The supplier provides an assessment to the insurer's Claims Handling team
6. The insurer's Supplier Assessment team respond requesting clarifications,
7. The insurer's Supplier Mgmt team approves work & makes adjustments as they arise
8. The supplier's Finance team sends an invoice to the insurer once the work is completed
9. The insurer's Supplier Management team process and validate the invoice
10. The insurer's Payment team make the payment to supplier & the claim is marked as closed



Teams	Insurer Team: Claims Handling	Supplier Team: Assessment	Insurer Team: Supplier Management	Supplier Team: Finance	Insurer Team: Finance	Insurer Team: Payments
Functions	Verify Claimant Policy	Make adjustments with insurer	Approve work & monitor	Send Invoice	Receive, Process & Validate Invoice	Make payment
Software and technology	Policy Database (Oracle); Supplier Database (Postgres); Email (SMTP)	Supplier UI Portal (REST/JSON Gateway); Email (SMTP)	Supplier UI Portal (REST/JSON Gateway); Email (SMTP)	Invoice Generator (Salesforce); Invoice Portal (SOAP/XML Gateway)	Invoice Portal (SOAP/XML Gateway); Validation Rules (Camunda workflow)	Payment Gateway (GoCardless API)
Process Operations	**1. Verify claimant policy** 2. Make adjustments 3. Approve work and monitor 4. Send Invoice 5. Receive, process, validate Invoice 6. Make payment	1. Verify claimant policy **2. Make adjustments** 3. Approve work and monitor 4. Send Invoice 5. Receive, process, validate Invoice 6. Make payment	1. Verify claimant policy 2. Make adjustments **3. Approve work and monitor** 4. Send Invoice 5. Receive, process, validate Invoice 6. Make payment	1. Verify claimant policy 2. Make adjustments 3. Approve work and monitor **4. Send Invoice** 5. Receive, process, validate Invoice 6. Make payment	1. Verify claimant policy 2. Make adjustments 3. Approve work and monitor 4. Send Invoice **5. Receive, process, validate Invoice** 6. Make payment	1. Verify claimant policy 2. Make adjustments 3. Approve work and monitor 4. Send Invoice 5. Receive, process, validate Invoice **6. Make payment**

Fig 7. Illustrates the process of processing supplier claims and the participants and systems involved.

3. Problem

3.1. Problem Overview

For reliable process operations, all teams and systems involved should operate the same end to end Process,

They often don't!

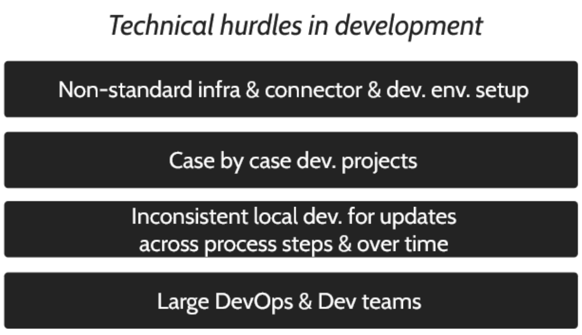
Enterprises face operational problems due to the complexity of their processes. Processes are operated across teams and systems resulting in disjointed and inefficient operations. Also, operations are non-standardized and inflexible leading to high costs, errors, and delays.

The Insurer's supplier order fulfillment process operates across separate large teams and software systems, making it expensive, inefficient and error-prone due to the need for manual intervention. This leads to operational problems such as duplicated requests due to slow time-scales, decreased customer satisfaction, and poor regulatory compliance.

3.2. Enterprise Process Operations Problems

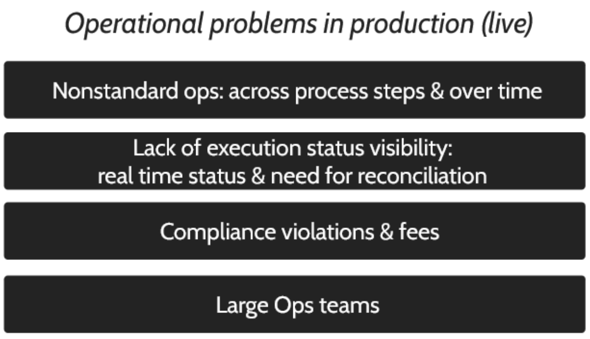
Enterprises are complex organizations operating many processes. Enterprises operate Processes across fragmented and siloed teams and software systems resulting in disjointed, inconsistent and inefficient end to end operations, leading to high costs, delays and errors. Specifically, operating processes across fragmented and siloed teams and software systems affect process operations both technically during the development phase & operationally once they go live in production.

Due to different infrastructure between participants and non-standard developer environments, large developer teams must be used to develop many bespoke pieces of software. These applications must then be strung together, requiring more developers, time and resources. The use of multiple development pipelines and environments leads to inconsistent and bloated code.


Technical hurdles in development
Non-standard infra & connector & dev. env. setup
Case by case dev. projects
Inconsistent local dev. for updates across process steps & over time
Large DevOps & Dev teams

Once the solution is live, the fragmented and separated systems require large teams to be able to operate each participant separately.

Nonstandard operations across the process lead to prolonged timescales which can lead to compliance violations. The separated systems involved in process operations continue to require dedicated development teams as they all require separate updates, which are often out of sync, further hindering the process.


Operational problems in production (live)
Nonstandard ops: across process steps & over time
Lack of execution status visibility: real time status & need for reconciliation
Compliance violations & fees
Large Ops teams

3.3. Supplier Claims Order Fulfillment Process Operations Problems

The Insurer's supplier claims order fulfillment process operates across several teams: Claims Handling team, Supplier Management team, Finance team and Payments team, and involves multiple software systems. These teams and systems are siloed and operate separately, and consequently the Insurer must employ large teams to ensure reliable operations. One of these large teams is Supplier Management. Assessments from suppliers, and clarifications, are received in different formats, and the Supplier Management team processes this information through the Supplier Portal and Email systems. Another large team is Claims Handling. The policies of claimants must be assessed to ensure they are valid against the policy database, and appropriate suppliers must be chosen to match the replacement or repair the claimant requires. Large teams incur high costs, and the large amount of manual intervention makes the process error-prone. Communication between large numbers of individuals and teams makes the process disjointed and inefficient. Additionally, clarification and approval between participants in the process take place in multiple formats, causing long delays. The large number of suppliers and product codes mean payments are often attempted but rejected as employees cannot find the correct code. Existing validation protocols leave room for error. Frustrated by slow payment timescales, suppliers frequently resubmit the payment request, resulting in duplicate payments and additional delays determining. This can lead to penalties and fines for the Insurer. Ultimately, inefficiencies resulting from the manual inputs in the current system increase wait times for claim completion, reducing customer and supplier satisfaction and increasing operational costs.

4. Traditional approaches to process operations and automation solutions don't work

4.1. Approach to Process Operations today

To set up a process, operations enterprises generally tend to utilize multiple large teams to set up and maintain bespoke, individual systems with associated software, and then write bespoke software (connectors) to connect and coordinate these systems. Because these systems are independent, they require independent maintenance and updates. Another consequence of independent systems is inconsistent operations. The operations script is fragmented across systems and participants because of a lack of end-to-end participant connectivity, resulting in inconsistent and non-transparent processes. Repeated setup, development and maintenance cycles require a considerable amount of resources on behalf of the enterprise, reflected directly in the large budgets required to set up, operate and maintain processes across the entire enterprise. Specifically, enterprises must mobilize large development teams to set up the infrastructure, connectors and developer environments needed to connect fragmented operations. Then large operations teams must be mobilized to maintain these systems. Now, developers must write execution software that will carry out the process on the system, and then software must be written to connect all these systems & to keep them coordinated. Once the process is operational, developer & support teams need to maintain these systems with bespoke updates & fixes.

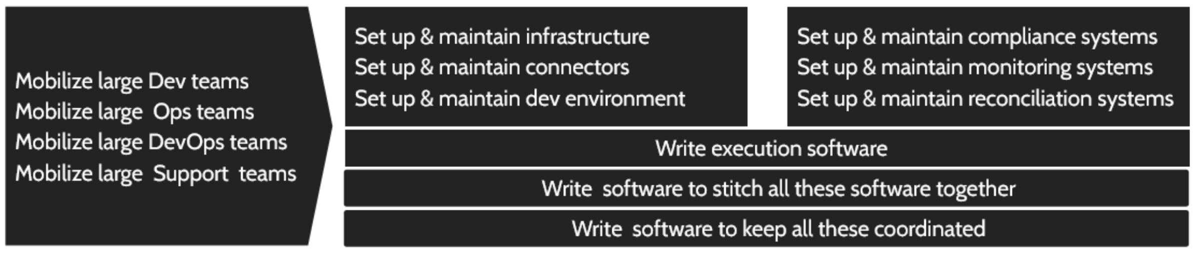

Mobilize large Dev teams
Mobilize large Ops teams
Mobilize large DevOps teams
Mobilize large Support teams
Set up & maintain infrastructure
Set up & maintain connectors
Set up & maintain dev environment
Set up & maintain compliance systems
Set up & maintain monitoring systems
Set up & maintain reconciliation systems
Write execution software
Write software to stitch all these software together
Write software to keep all these coordinated

Fig 8. Enterprises must carry out all of the above to run a process in the modern marketplace.

4.2. Bespoke Connectors & Operations Scripts & why they don't work

Bespoke local connections remove coordination and coupling problems across team boundaries. However, no individual has a coherent view of the end-to-end process and implementers of individual services must coordinate their own execution logic. Because of this, each individual service must be maintained and updated separately, leading to delays and disconnects in the process, as well as higher operational costs.

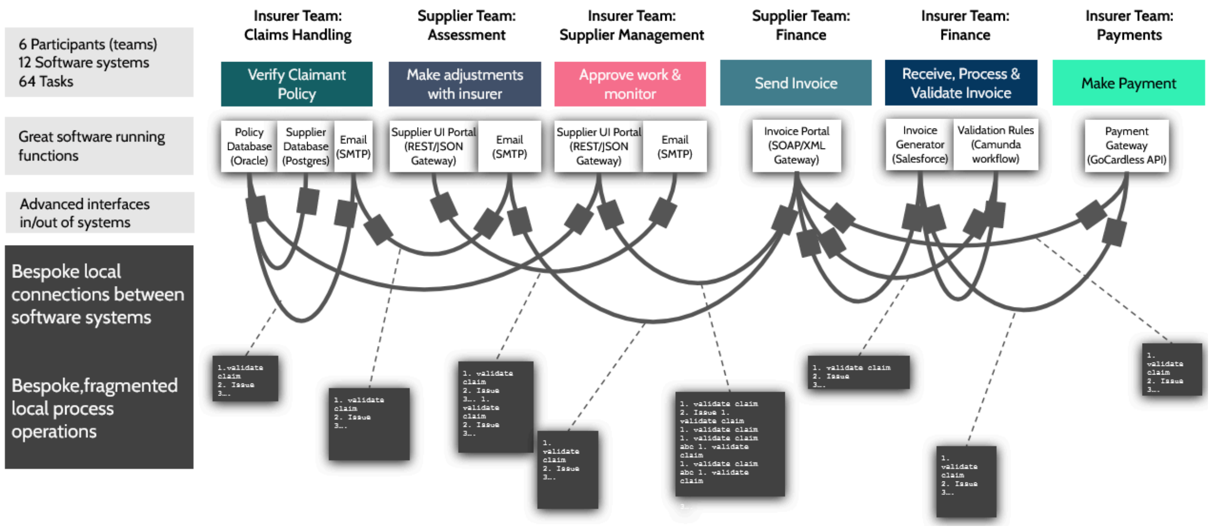
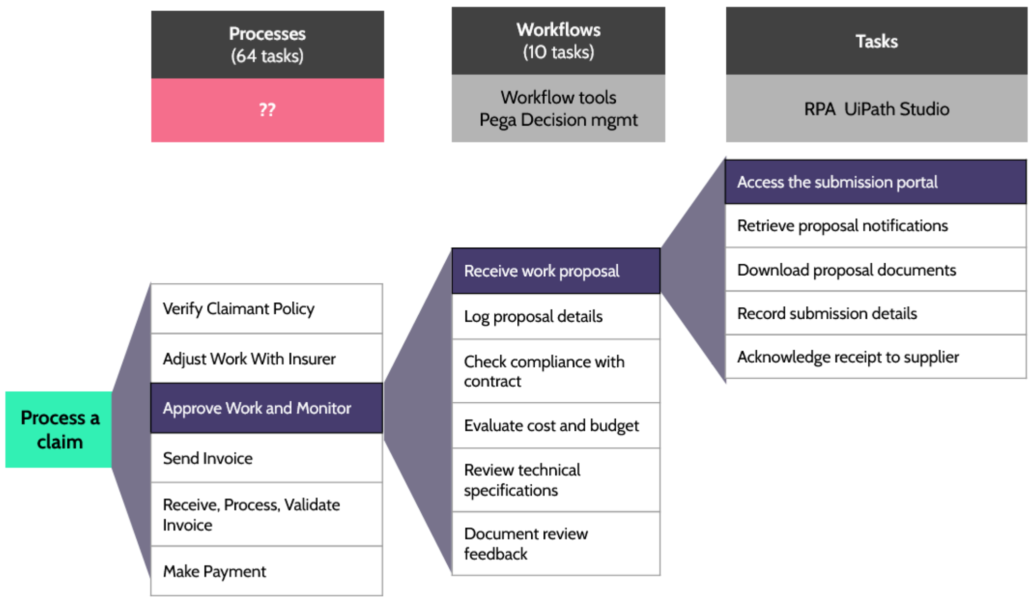

6 Participants (teams)
12 Software systems
64 Tasks
Great software running functions
Advanced interfaces in/out of systems
Bespoke local connections between software systems
Bespoke,fragmented local process operations
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make Payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Portal (SOAP/XML Gateway)
Invoice Generator (Salesforce)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. validate claim 2. Issue 3...

Fig 9. Bespoke local connections across the end-to-end process that are internally developed by the enterprise.

4.3. Local Automation (RPA, Workflow) tools & why stitching them together doesn't work

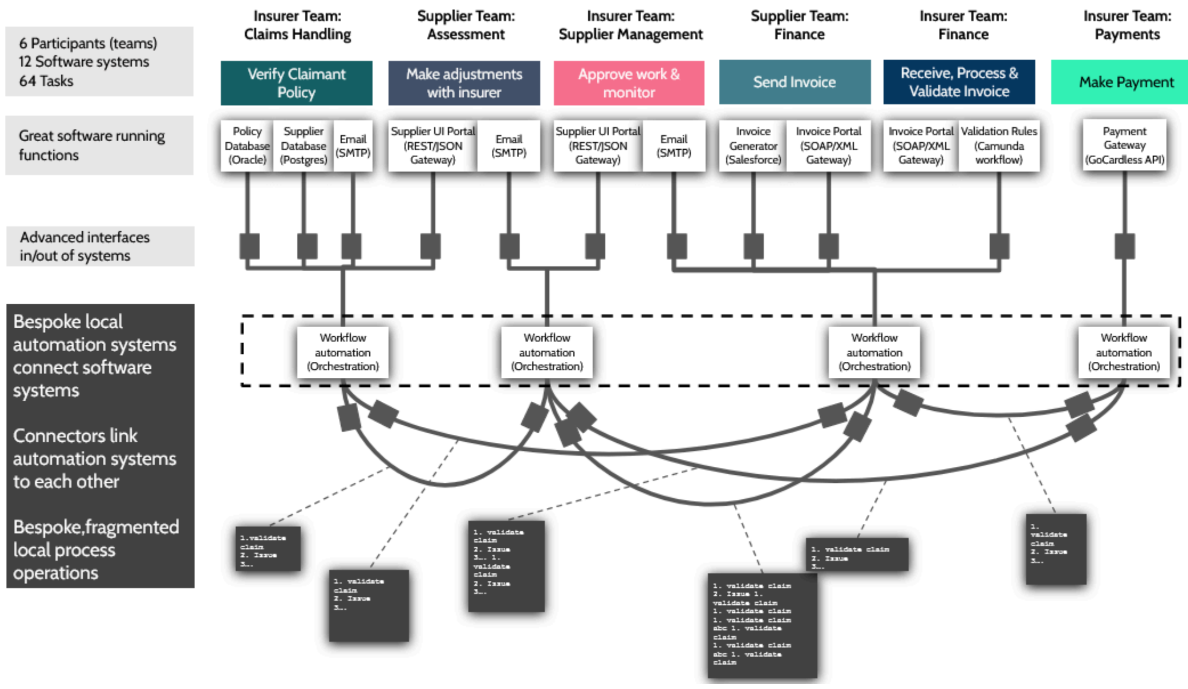
Processes in an enterprise are a series of operations (tasks) that are generally followed in a specific order based on outcomes of the previous operations (tasks). Each process consists of a collection of Workflows (10-20 tasks), each of which is a collection of Tasks.



Processes (64 tasks)
??
Workflows (10 tasks)
Workflow tools Pega Decision mgmt
Tasks
RPA UiPath Studio
Process a claim
Verify Claimant Policy
Adjust Work With Insurer
Approve Work and Monitor
Send Invoice
Receive, Process, Validate Invoice
Make Payment
Receive work proposal
Log proposal details
Check compliance with contract
Evaluate cost and budget
Review technical specifications
Document review feedback
Access the submission portal
Retrieve proposal notifications
Download proposal documents
Record submission details
Acknowledge receipt to supplier

Fig 10. Today, there are no traditional tools which effectively automate processes.

To overcome the traditional approach, enterprises attempt to stitch together software systems, RPA tools and workflow tools across the process. Process orchestration approaches stitch together combinations of RPA and workflow systems through point-to-point message passing techniques. The service implements a batch scheduler or workflow system. This is effective at coordinating tasks within a single team, but is inadequate for the operations of processes spanning multiple teams. Individual teams create bespoke code for their tasks, resulting in "script bloat" - the proliferation of numerous, redundant, poorly documented scripts that complicate maintenance and scalability. There is a lack of transparency between participants and this lack of coordination and integration results in process inefficiencies and errors, which results in delays and operational friction.



6 Participants (teams)
12 Software systems
64 Tasks
Great software running functions
Advanced interfaces in/out of systems
Bespoke local automation systems connect software systems
Connectors link automation systems to each other
Bespoke,fragmented local process operations
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make Payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
Workflow automation (Orchestration)
Workflow automation (Orchestration)
Workflow automation (Orchestration)
Workflow automation (Orchestration)

Fig 11. Stitching together local automation tools through local RPA and workflow tools.

For a full explanation of traditional process operations and Luther's solution, request access to the *["Deep Process Automation Primer"]*

5. Solution

5.1. Solution Overview

Luther's platform was used by the Insurer's development team to build a world-class supplier order and claims order fulfillment process. This required an automated system that could handle the complexity of a process involving many internal teams and external inputs with progressive validation steps. The platform provides:

- standard connectivity across all software systems
- a common operations script that contains the end-to-end process operations

The Luther platform ensures all participants operate through a standardized set of processes operated on and validated by the platform, without interruption and with no loss of data.

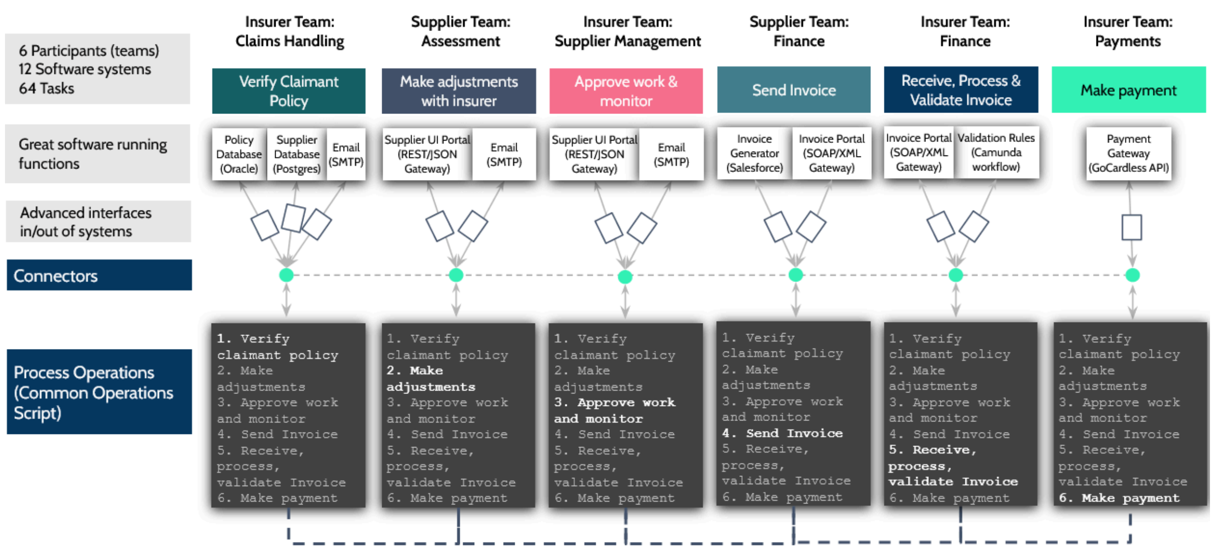

6 Participants (teams)
12 Software systems
64 Tasks
Great software running functions
Advanced interfaces in/out of systems
Connectors
Process Operations (Common Operations Script)
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment

Fig 12. Overview of the Luther Platform automating the claims fulfillment process.

This is very difficult and costly with traditional automation tools and workflows. Luther's Deep Automation Platform allowed the automation of the process of claims fulfillment filing. Project Ford is the result of this work and is an end-to-end claims fulfillment system that standardizes the process with minimal manual intervention while reducing claim fulfillment times and operational costs.

5.2. Demo

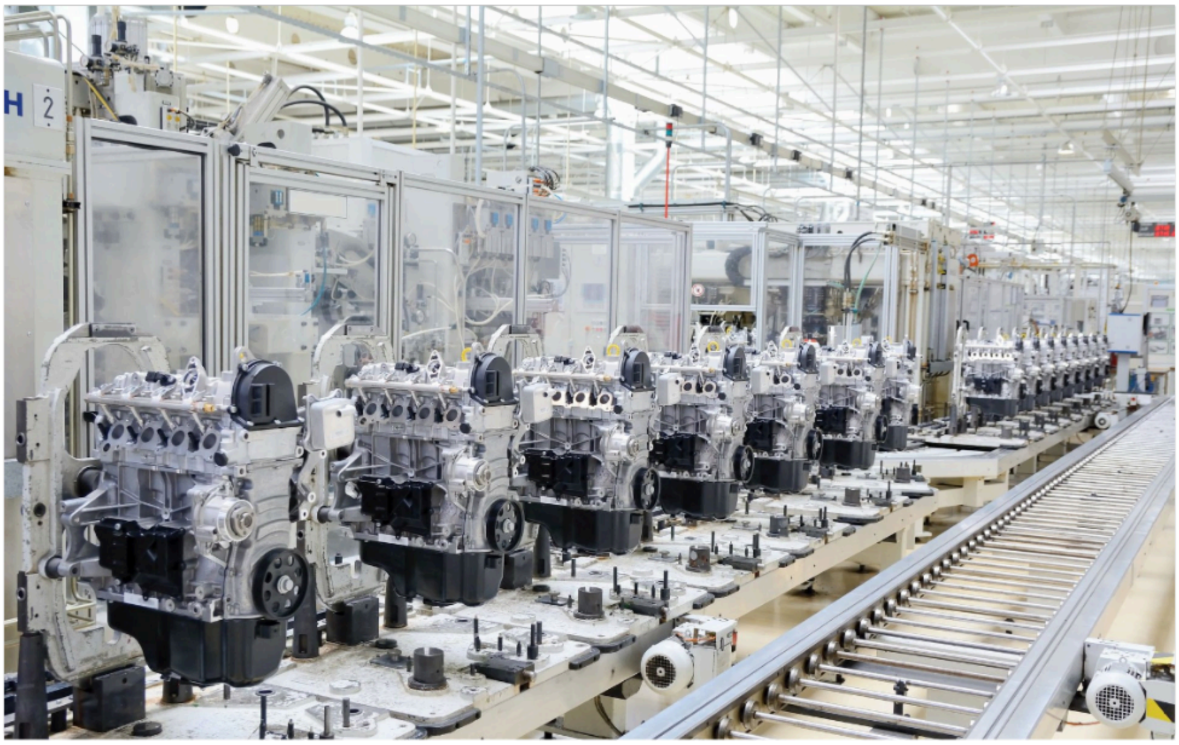
Please find the "Ford Solution Demo" here. Access is available on request.

5.3. How it works on the Luther Platform

The Supplier Claims Order Fulfillment Process:

- A claimant comes to the insurer requesting a repair or replacement item
- Platform acknowledges request & verifies validity of policy for claim & claimant
- The insurer provides the claimant with a list of approved suppliers
- Supplier provides an assessment of damages to the platform
- The insurer responds requesting clarifications, and the supplier responds
- The insurer approves work starting
- Supplier informs the insurer of progress/adjustments to the work via a supplier portal,
- The insurer approves progress/adjustments
- Work is completed and supplier sends invoice to Invoice Portal onto the Luther Platform
- The invoice is processed and validated using the common operations script
- The payment is made to the supplier automatically once the invoice is validated
- The claim is marked as completed and closed

For a more detailed view of one of the steps operating the Luther Platform, please view the appendix.



6. Implementation

The Luther platform ensures all participants operate through a standardized set of steps executed on and validated by the platform. An itemized timeline (Fig 13) was created to implement Project Ford with the Insurer. Request access to an example of a more detailed timeline here: *"Luther Get Started Questionnaire"*.

First, Luther's team identified all teams and all software systems involved in the operations of the process. Luther's team worked with the teams at the Insurer to map the process into a series of objective steps. View a process map at *Platform: How it works - Luther Systems*.

Then Luther's team allocated a node to each participant, deployed the platform on all nodes, and connected the nodes to each of the software systems. Then the Luther team worked with developers to create a robust common operations script for process operations. Request access to the *"Ford Demo Video"* here.

For a full explanation of the implementation process, view the *Full Luther Platform Setup*.

For a walkthrough of the implementation process, view the *Luther Systems Sandbox Setup*.

Customer Team		*Business owner, Application owner, Technical lead*	*Day 1*
Discover	*Phase 1*	*Describe process operations*	*2-4 weeks*
	Phase 2	*Describe systems & technical requirements*	
Process mapping		*Map the process*	*1 week*
Platform set up		*One-time platform set-up*	*1 day*
Build application		*Develop (code) application operations*	*4-8 weeks*

Fig 13. Implementation timeline for Project Ford.

6.1. Process mapping

Luther's team worked with multiple Insurer team members to map the process operations, which can be hosted on the Luther Platform. The process map includes functions, data inputs and outputs at each step, and rules and decisions at each step. Participants are operationally separate teams or entities involved in the process. As part of process mapping, the Luther team identified the exact set of software systems, participants/teams involved in operating the end-to-end process.

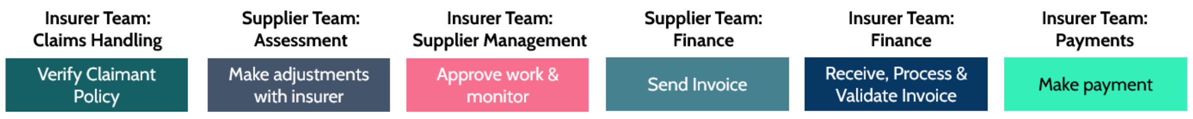

Insurer Team: Claims Handling
Verify Claimant Policy
Supplier Team: Assessment
Make adjustments with insurer
Insurer Team: Supplier Management
Approve work & monitor
Supplier Team: Finance
Send Invoice
Insurer Team: Finance
Receive, Process & Validate Invoice
Insurer Team: Payments
Make payment

Fig 14. Luther's team worked with the insurer and mapped the process into a series of objective, standardized steps associated with 6 participants.

6.2. Identify software systems

Luther's team identified the software systems involved in end-to-end process operations. These systems are: Supplier Database (Postgres), Policy Database (Oracle), Supplier Portal (REST/JSON Gateway), Email (SMTP), Invoice Portal (SOAP/XML Gateway), Invoice Generator (Salesforce, OpenKoda), Invoice Validation Rules (Camunda Workflow), and Payment Gateway (GoCardless Payment Gateway API).

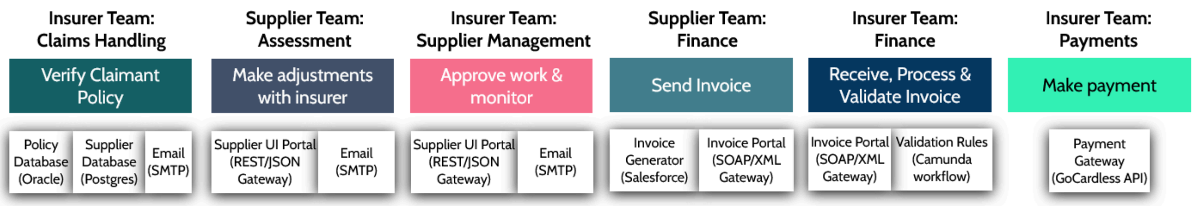
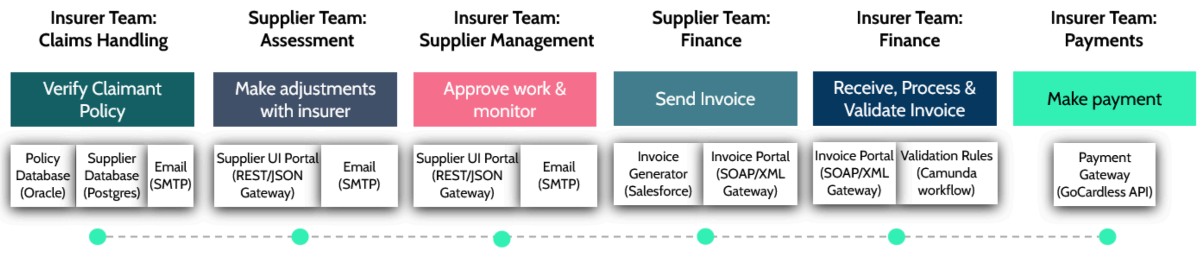

Insurer Team: Claims Handling
Verify Claimant Policy
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier Team: Assessment
Make adjustments with insurer
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Insurer Team: Supplier Management
Approve work & monitor
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier Team: Finance
Send Invoice
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Insurer Team: Finance
Receive, Process & Validate Invoice
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Insurer Team: Payments
Make payment
Payment Gateway (GoCardless API)

Fig 15. Luther's team identified the software systems involved in the end-to-end process operations.

6.3. Nodes and Connectivity through distributed system for end-to-end participant connectivity

Luther's team assigned a dedicated node to each participant involved in the process by allocating servers to their respective teams.

These servers are cloud-native and can be deployed on either public or private clouds, depending on security requirements. All nodes are interconnected through a distributed system, which facilitates the sharing and validation of operational functions and data among all participants.

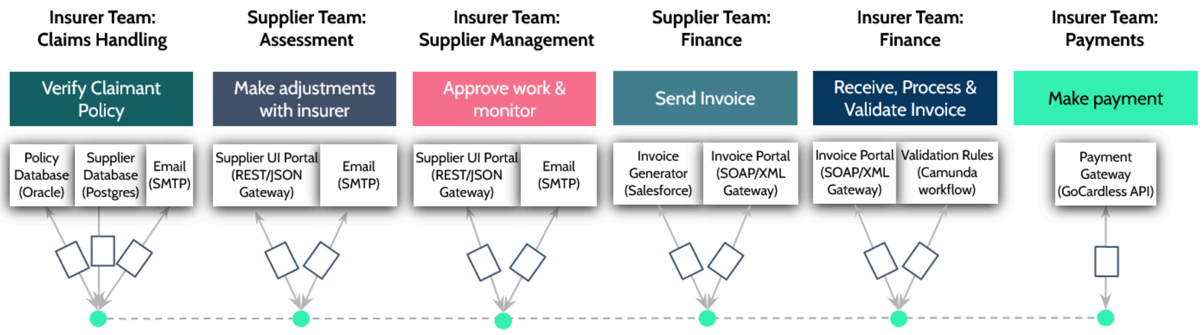


Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)

Fig 16. Nodes are connected via a distributed system on the Luther Platform.

6.4. Connectors to software systems

Each participant/team has a number of software systems involved in its operations, as identified in the process map. For each participant/team, Luther's platform connects its node to all software systems involved in its operations. Luther has a set of standard connectors across a wide range of enterprise systems, which the Luther platform deploys to rapidly connect to a majority of systems in an enterprise. This is done by determining the technology, type and system of the connector to connect to each system in the process.



Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)

Fig 17. Luther's team set up connectors that link the processes together.

For a full list of our connectors, please visit: *"Luther Platform Connectors"*.

6.5. Platform set-up

Luther's team deployed the platform on all nodes.

The Insurer team selected a set of configurations for their platform specifications. This selection depends on the process complexity (number of tasks), amount of data processed (KB), number of participants, reliability, availability and security requirements. For a closer look at the platform configuration specs please visit our website here: *"Luther Platform Connectors"*.

Luther's platform vertically integrates

distributed system technology
optimal resource allocation and management
real time event ordering and streaming (sharing)
deterministic event processing and execution

To make reliable end-to-end process operations possible.

Luther's platform vertically integrates distributed system technology, optimal resource allocation and management, real-time event ordering and streaming (sharing), and deterministic event processing and execution, to provide a modern technology stack to reliably operate an end-to-end process across multiple software systems, at scale.

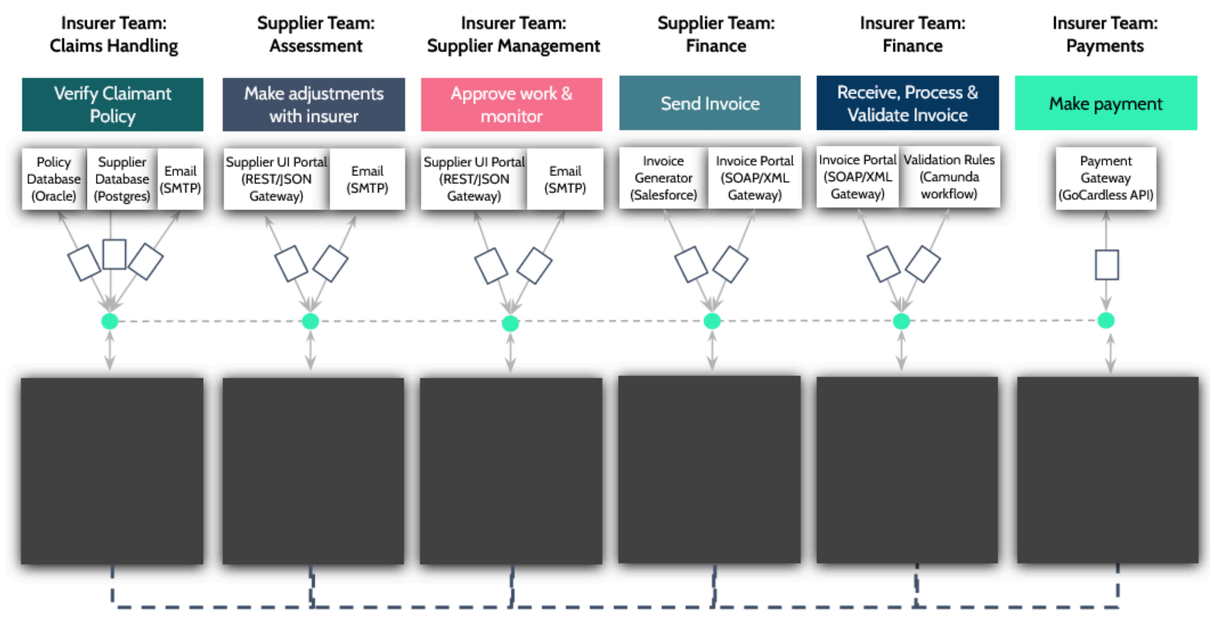

Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)

Fig 18. The platform is set up on each of the nodes, and a common operations script links independent systems into one cohesive process.

6.6. Common Operations Script for process operations

The platform is now fully set up and integrated with all systems involved in the operation. The Insurer's development team, in collaboration with Luther, developed the Common Operations Script to manage the end-to-end process. Connectors translate data from local systems into a common data model utilized by the Common Operations Script. This script encapsulates the business logic, data, rules, and validations for each process step.

The Common Operations Script effectively encodes and operates the process map, executing the following steps across the entire process.

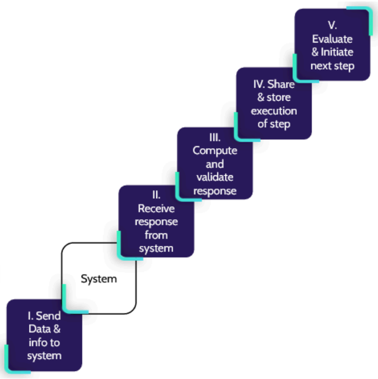
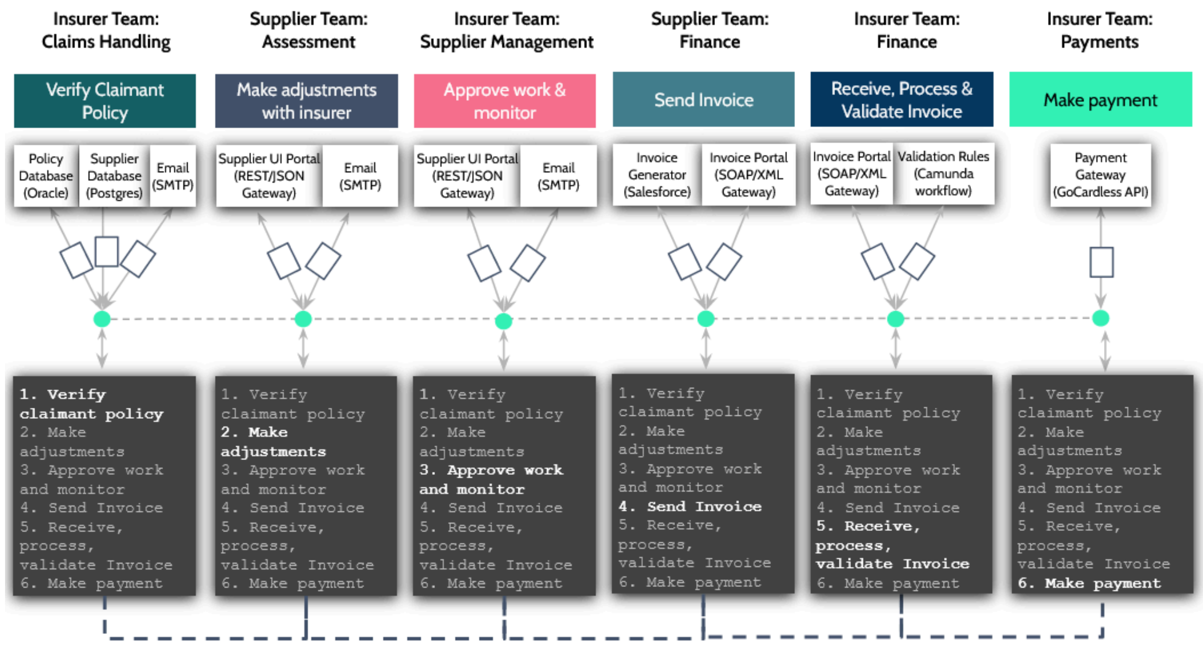

V. Evaluate & Initiate next step
IV. Share & store execution of step
III. Compute and validate response
II. Receive response from system
System
I. Send Data & info to system

Fig 19. These are the requirements that repeat for all functions across the end-to-end Process Operations.

This script is shared by all participants and operates on the Luther Platform. Each participant can change the script through suggesting changes, once the changes to the script are approved by all participants the script is updated for all participants. This gives the enterprise full autonomy over the process operations to modify and change, it also ensures all participants are operating "the same process" at all times. When a team changes their operations, the operations for all participants are updated simultaneously.



Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Verify Claimant Policy
Make adjustments with insurer
Approve work & monitor
Send Invoice
Receive, Process & Validate Invoice
Make payment
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy 2. Make adjustments 3. Approve work and monitor 4. Send Invoice 5. Receive, process, validate Invoice 6. Make payment

Fig 20. Luther and the insurer developer teams work together to write the common operations script, converting tasks into an objective workflow that links every step in the process.

For more information about Luther's platform please consult this video: "Luther Platform Setup".

For a demo of the build process please visit our website: "Luther Build Process Demo".

6.7. Go live (production)

Once the platform is set up and the Common Operations Script has been written, the product is ready to go live. Once live, it automates the claims order fulfillment process by providing end-to-end connectivity between participants, and it receives across the board updates when needed instead of siloed, separate updates to individual software systems. The process is optimally coordinated, scalable, and resilient to change. Furthermore, the process has less downtime, and requires minimal manual intervention.

7. Results

7.1. Commercial results

Using Luther's Deep Process Automation platform, the implementation of Project Ford was able to reduce the cost of the claims order fulfillment process by 70%. This is primarily due to FTE savings in both ops teams involved in process operations as well ops teams involved in reprocessing supplier orders. The percentage of errors is estimated to be reduced to below 2%, due to automating operations and identifying many errors in near real time. The average total time for an invoice to be processed was reduced from 35 days to just 4 days, speeding up the average processing time by 10X. This results in a return on investment of 600%.

ROI: 6X

Cost Reduction: 70%

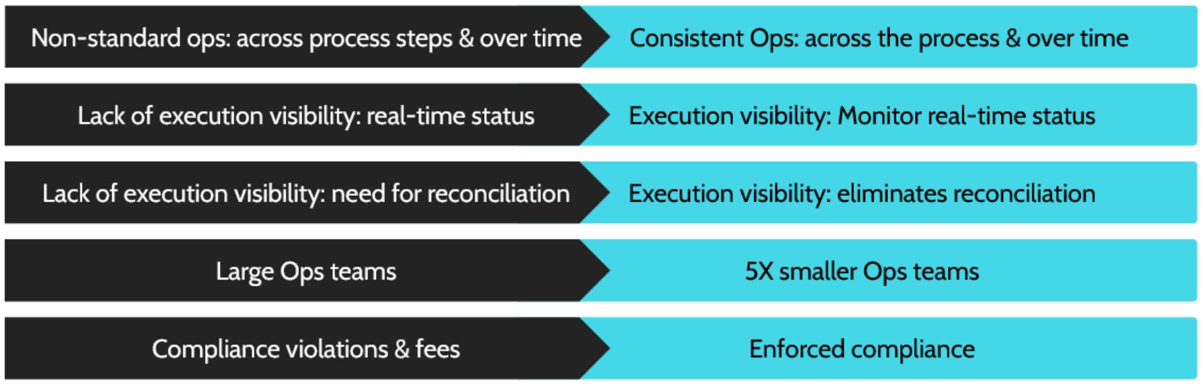
Average Processing Speed Increase: 10X

7.2. Operational benefits

Luther delivered a product that standardizes the supplier claims order fulfillment process operations, and demonstrated the potential for other suppliers to be integrated into the network, while reducing inefficiencies, improving process transparency, reducing the size of operations teams, and improving compliance, which could not have been achieved without Luther's Deep Automation Technology.

General operational advantages

The Luther Platform streamlines operations across enterprise processes, reducing process time and cost while maintaining transparency and flexibility.



Non-standard ops: across process steps & over time	Consistent Ops: across the process & over time
Lack of execution visibility: real-time status	Execution visibility: Monitor real-time status
Lack of execution visibility: need for reconciliation	Execution visibility: eliminates reconciliation
Large Ops teams	5X smaller Ops teams
Compliance violations & fees	Enforced compliance

Fig 21. General results from implementation of the Luther platform

Specific operational advantages

Implementing Project Ford has streamlined the operations of the supplier order fulfillment process, making it more efficient, faster, and standardized all while requiring minimal manual intervention, and without sacrificing transparency or integrity of data. The platform is flexible and scalable to future changes to the process or regulations.

Enhanced invoice processing:

- Provides visibility and transparency for most up-to-date order details to all participants
- Policies and orders can be automatically updated with zero downtime

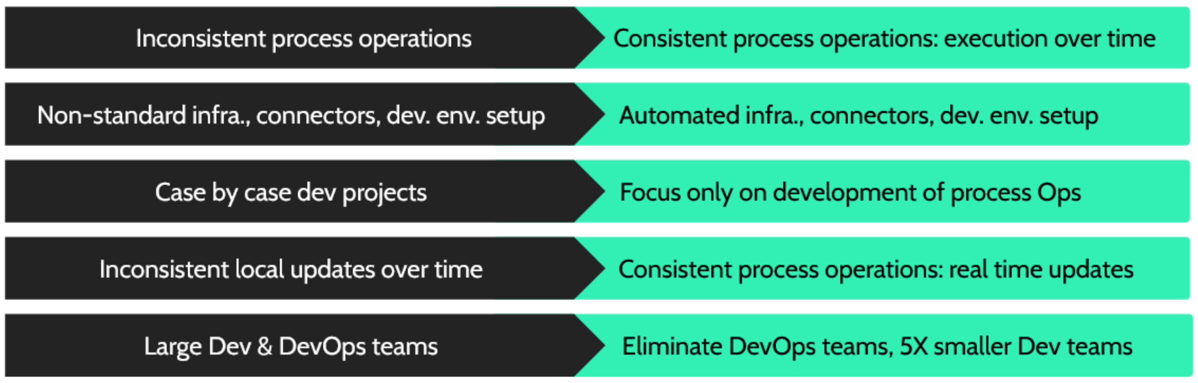
Enhanced operations:

- Flexibility and scalability for policy changes in-built to the platform
- Elimination of manual intervention means smaller operational teams for the insurer
- Standardization of the order fulfillment process, ensuring faster timescales helping to avoid compliance violations
- Increased system reliability and fewer processing errors reduce costs associated with duplicate invoices

7.3. Technical benefits

General technical advantages

The Luther Platform ensures each step in the process is recorded and validated, and enforces common process logic across the entire network, resulting in a standardized and consistent process.



Inconsistent process operations	Consistent process operations: execution over time
Non-standard infra., connectors, dev. env. setup	Automated infra., connectors, dev. env. setup
Case by case dev projects	Focus only on development of process Ops
Inconsistent local updates over time	Consistent process operations: real time updates
Large Dev & DevOps teams	Eliminate DevOps teams, 5X smaller Dev teams

Fig 22. General technical results from implementation of the Luther platform.

Specific technical advantages

Improved operating efficiency:

- Automatically provides verified execution to increase system reliability and reduces processing errors
- Common execution visibility to all participants reduces troubleshooting effort and quickly identifies processing bottlenecks
- Automatically supports Common Operations Script updates including new validation rules

Reduced audit trail and reporting costs:

- Entire order and invoice history is securely maintained by the platform with timestamps for every event

8. Expansion

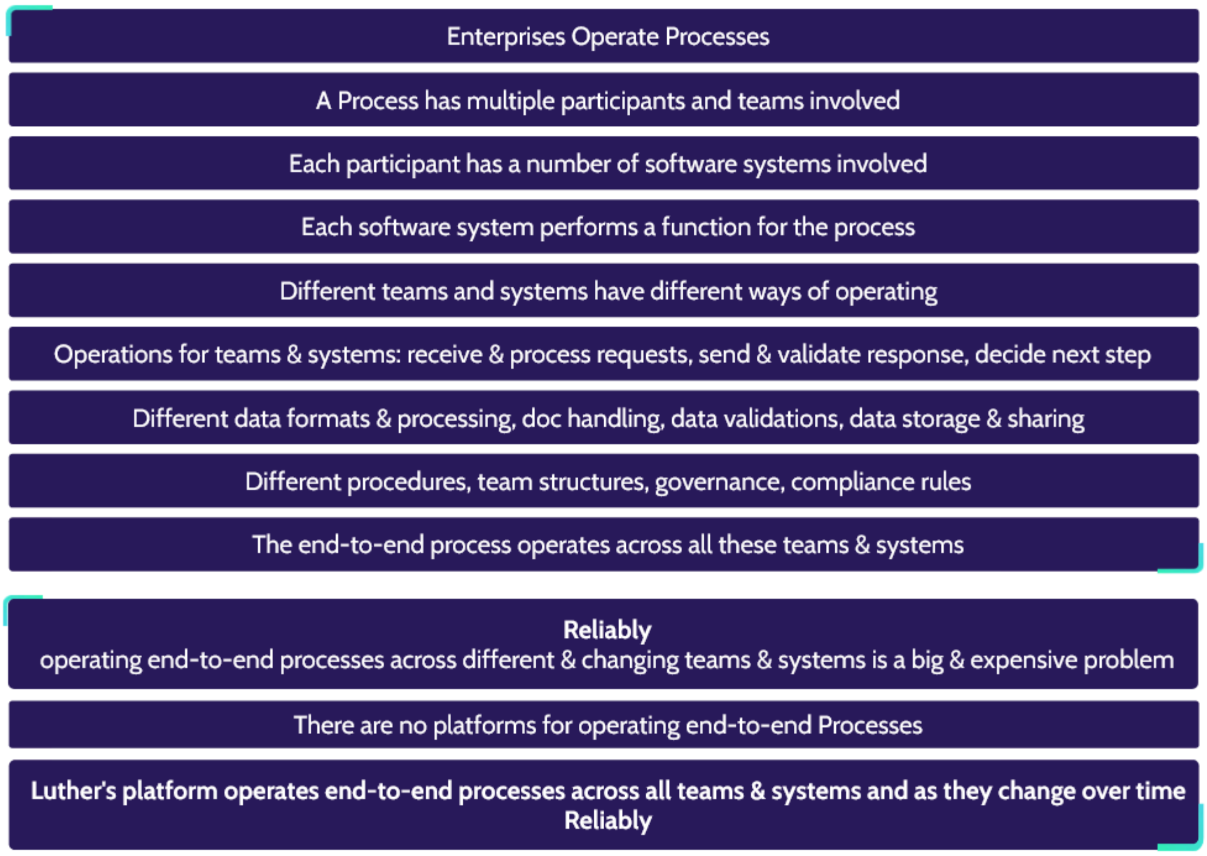
This project demonstrates a sleek, effective system built on the Luther platform to standardize and automate the supplier claims order fulfillment process. The network could be further expanded to encompass other suppliers, but the Luther platform could also be utilized to further streamline other areas of the Insurer's operations by increasing the scope of the network, resulting in further reductions to costs and inefficiencies.

Potential areas of expansion to further integrate the Insurer's operations into a single platform include:

- Expanding the invoice management network to include other suppliers
- Using the Luther platform to implement new technologies to allow customers to purchase insurance policies and open financial accounts more quickly, improving customer satisfaction
- Automated compliance protocols to protect the Insurer from breaches of regulations resulting in fewer penalties and fines
- Expanding the network to encompass other aspects of the Insurer's operations such as financial settlements

9. Luther Company & Offerings

9.1. What Luther does



Enterprises Operate Processes

A Process has multiple participants and teams involved

Each participant has a number of software systems involved

Each software system performs a function for the process

Different teams and systems have different ways of operating

Operations for teams & systems: receive & process requests, send & validate response, decide next step

Different data formats & processing, doc handling, data validations, data storage & sharing

Different procedures, team structures, governance, compliance rules

The end-to-end process operates across all these teams & systems

Reliably
operating end-to-end processes across different & changing teams & systems is a big & expensive problem

There are no platforms for operating end-to-end Processes

Luther's platform operates end-to-end processes across all teams & systems and as they change over time Reliably

Fig 23. Luther's offerings solve the complicated problem of enterprise process operations.

For more information about Luther, please visit luthersystems.com.

9.2. *“In a nutshell” - Luther’s unique value*

Luther’s platform vertically integrates

distributed system technology
optimal resource allocation and management
real time event ordering and streaming (sharing)
deterministic event processing and execution

To make reliable end-to-end process operations possible.

Luther’s unique value for reliable end-to-end Process Operations is providing

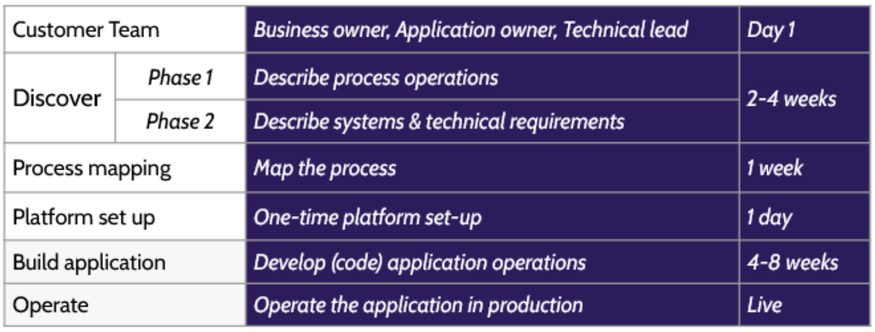
standard connectivity
a common operations script

across all teams and software systems.

Luther automates the operations of the end-to-end process across multiple teams and systems

9.3. Platform implementation

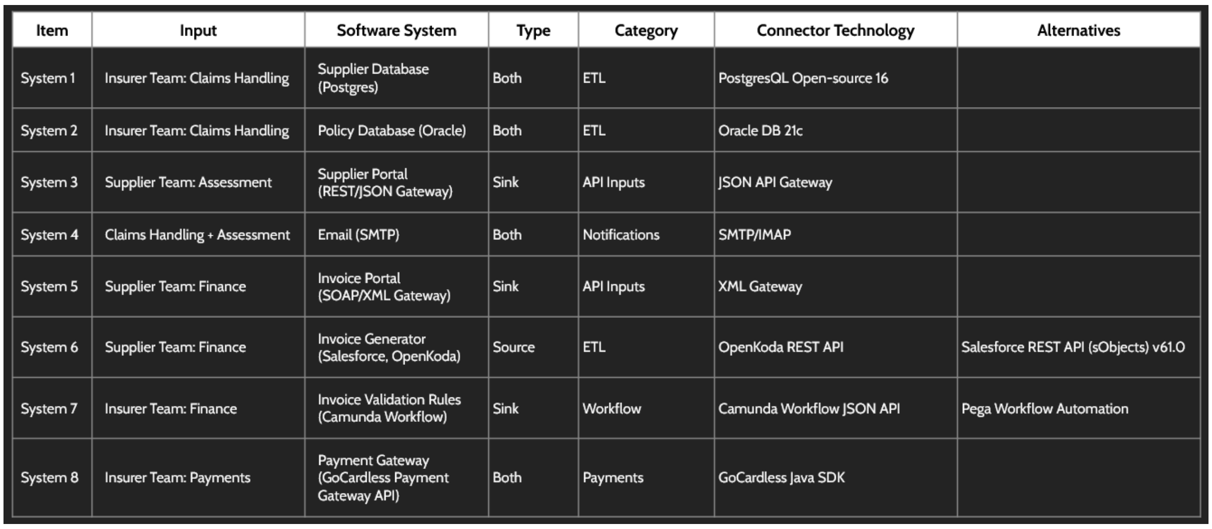
To implement the Luther Platform, organizations work with Luther through an implementation process - laying out objectives and expectations for the project, then mapping the process, setting up the infrastructure, and enterprise developers coding the process.



Customer Team		*Business owner, Application owner, Technical lead*	*Day 1*
Discover	*Phase 1*	*Describe process operations*	*2-4 weeks*
	Phase 2	*Describe systems & technical requirements*	
Process mapping		*Map the process*	*1 week*
Platform set up		*One-time platform set-up*	*1 day*
Build application		*Develop (code) application operations*	*4-8 weeks*
Operate		*Operate the application in production*	*Live*

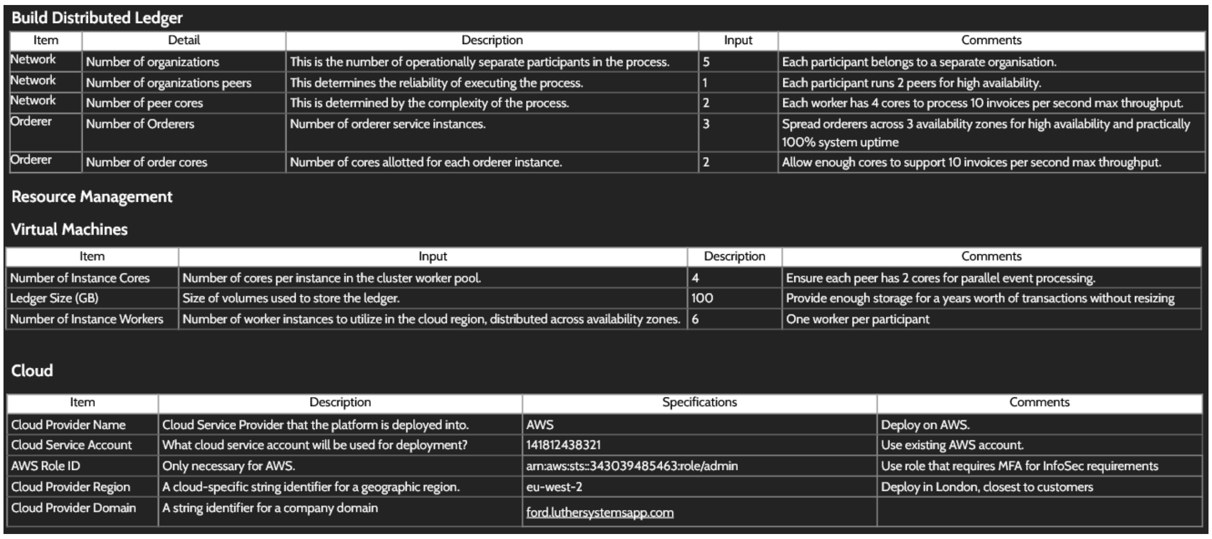
Fig 24. Implementation timeline for an application operated on the Luther Platform..

Enterprises working with Luther fill in the details of all software systems and connectors for their processes. These documents are used to build the process map and subsequently, the application.



Item	Input	Software System	Type	Category	Connector Technology	Alternatives
System 1	Insurer Team: Claims Handling	Supplier Database (Postgres)	Both	ETL	PostgresQL Open-source 16	
System 2	Insurer Team: Claims Handling	Policy Database (Oracle)	Both	ETL	Oracle DB 21c	
System 3	Supplier Team: Assessment	Supplier Portal (REST/JSON Gateway)	Sink	API Inputs	JSON API Gateway	
System 4	Claims Handling + Assessment	Email (SMTP)	Both	Notifications	SMTP/IMAP	
System 5	Supplier Team: Finance	Invoice Portal (SOAP/XML Gateway)	Sink	API Inputs	XML Gateway	
System 6	Supplier Team: Finance	Invoice Generator (Salesforce, OpenKoda)	Source	ETL	OpenKoda REST API	Salesforce REST API (sObjects) v61.0
System 7	Insurer Team: Finance	Invoice Validation Rules (Camunda Workflow)	Sink	Workflow	Camunda Workflow JSON API	Pega Workflow Automation
System 8	Insurer Team: Payments	Payment Gateway (GoCardless Payment Gateway API)	Both	Payments	GoCardless Java SDK	

Fig 25. The list of software systems for Project Ford, similar to one an enterprise building an application on the Luther Platform would fill out.



Build Distributed Ledger

Item	Detail	Description	Input	Comments
Network	Number of organizations	This is the number of operationally separate participants in the process.	5	Each participant belongs to a separate organisation.
Network	Number of organizations peers	This determines the reliability of executing the process.	1	Each participant runs 2 peers for high availability.
Network	Number of peer cores	This is determined by the complexity of the process.	2	Each worker has 4 cores to process 10 invoices per second max throughput.
Orderer	Number of Orderers	Number of orderer service instances.	3	Spread orderers across 3 availability zones for high availability and practically 100% system uptime
Orderer	Number of order cores	Number of cores allotted for each orderer instance.	2	Allow enough cores to support 10 invoices per second max throughput.

Resource Management

Virtual Machines

Item	Input	Description	Comments
Number of Instance Cores	Number of cores per instance in the cluster worker pool.	4	Ensure each peer has 2 cores for parallel event processing.
Ledger Size (GB)	Size of volumes used to store the ledger.	100	Provide enough storage for a years worth of transactions without resizing
Number of Instance Workers	Number of worker instances to utilize in the cloud region, distributed across availability zones.	6	One worker per participant

Cloud

Item	Description	Specifications	Comments
Cloud Provider Name	Cloud Service Provider that the platform is deployed into.	AWS	Deploy on AWS.
Cloud Service Account	What cloud service account will be used for deployment?	141812438321	Use existing AWS account.
AWS Role ID	Only necessary for AWS.	arn:aws:sts::343039485463:role/admin	Use role that requires MFA for InfoSec requirements
Cloud Provider Region	A cloud-specific string identifier for a geographic region.	eu-west-2	Deploy in London, closest to customers
Cloud Provider Domain	A string identifier for a company domain	ford.luthersystemsapp.com	

Fig 26. A sample list of connectors and infrastructure, similar to one an enterprise building an application on the Luther Platform would fill out.

9.4. Results of the Luther platform for Process Operations Automation

At Luther, we recognize that enterprise processes of today are complex and challenging to automate. We provide a platform for successful process automation.

The results are incredible. Enterprises working with Luther see an average of 10 times their investment. Time is saved everywhere, with development of process applications and automation technology sped up by 2.5 times, and processing times 7 times faster.

2.5X faster development

10X less operational costs

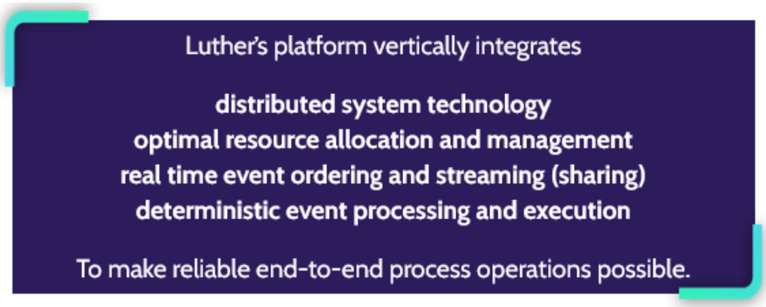
7X faster processing time

10X ROI

1000s of compliance rules automated

Find more information about Luther's Platform Core Features here.

9.5. Luther’s platform architecture


Luther’s platform vertically integrates

distributed system technology
optimal resource allocation and management
real time event ordering and streaming (sharing)
deterministic event processing and execution

To make reliable end-to-end process operations possible.

For a more detailed introduction on the Luther platform please request access to the *"Luther Deep Process Automation Primer"*.

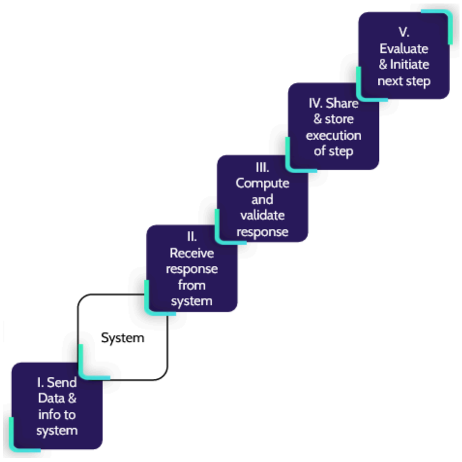
For a detailed introduction and documentation examples please see the Luther Platform site.

For more information about Luther’s platform please visit luthersystems.com.

10. Appendix

10.1 How the platform operates an end-to-end process: Application walkthrough

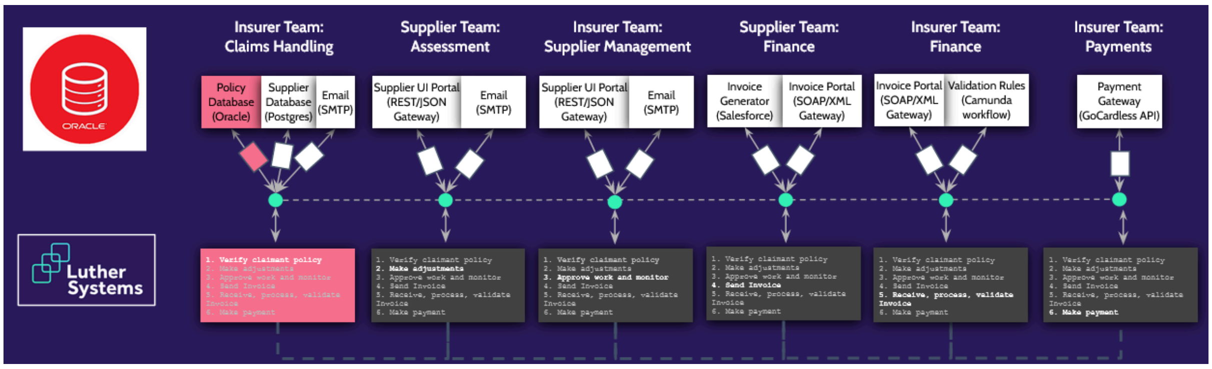
Below is a more detailed walkthrough of the process operations, across the teams and software systems. Each step in the process follows the exact same 5 operational steps which the Platform executes



I. Send Data & info to system
System
II. Receive response from system
III. Compute and validate response
IV. Share & store execution of step
V. Evaluate & Initiate next step

Step 1: Insurer Claims Handling team executes Verify Claimant Policy (retrieve Policy information)

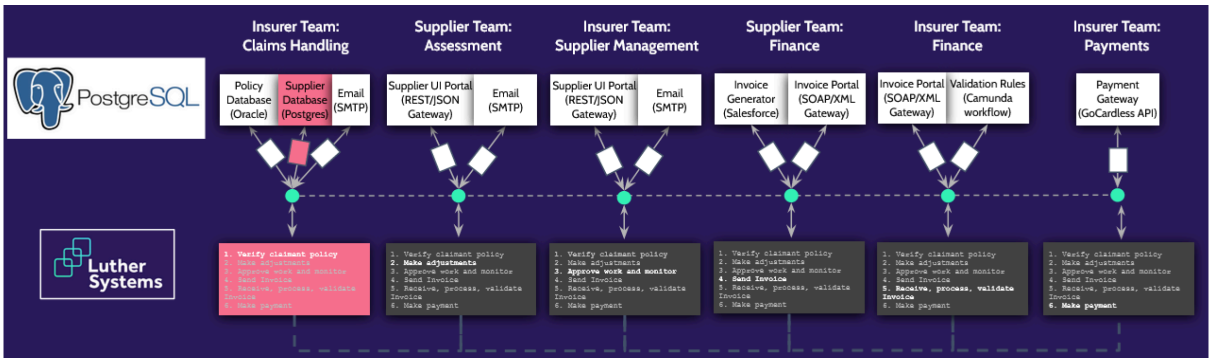
I. Platform sends (request) *policy ID* to Policy Database [Oracle]
II. Platform receives (response) *policy Information* from Policy Database
III. Platform validates policy information based a set of predetermined rules in the Common Operations Script
IV. Platform shares & stores claimant Policy from Policy Database
V. Platform evaluates & initiates next step



ORACLE
Luther Systems
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment

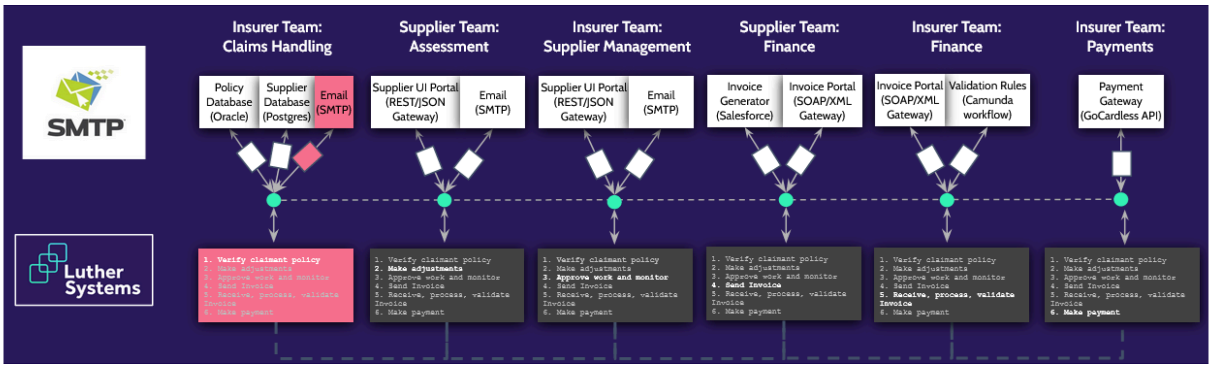
Step 2: Insurer Claims Handling team executes Verify Claimant Policy (retrieve Supplier information)

I. Platform sends (request) *claims ID* to Supplier Database [PostgreSQL DB]
II. Platform receives (response) *supplier information* from Supplier Database
III. Platform validates supplier information based a set of predetermined rules in the Common Operations Script
IV. Platform shares and stores supplier information from Supplier Database
V. Platform evaluates & initiates next step


PostgreSQL
Luther Systems
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment

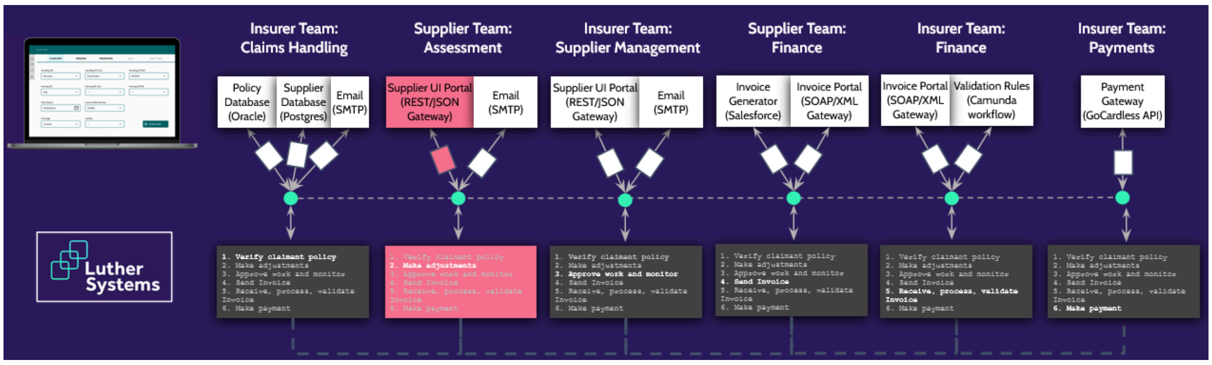
Step 3: Insurer Claims Handling Team executes Verify Claimant Policy (send information about relevant suppliers to the claimant)

I. Platform sends (request) *supplier information* to Email [SMTP]
II. Platform receives (response) *email confirmation* from Email
III. Platform validates email confirmation based a set of predetermined rules in the Common Operations Script
IV. Platform shares and stores email confirmation from Email
V. Platform evaluates & initiates next step


SMTP
Luther Systems
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment

Step 4: Supplier Assessment Team executes Make adjustments with Insurer (send assessment to insurer)

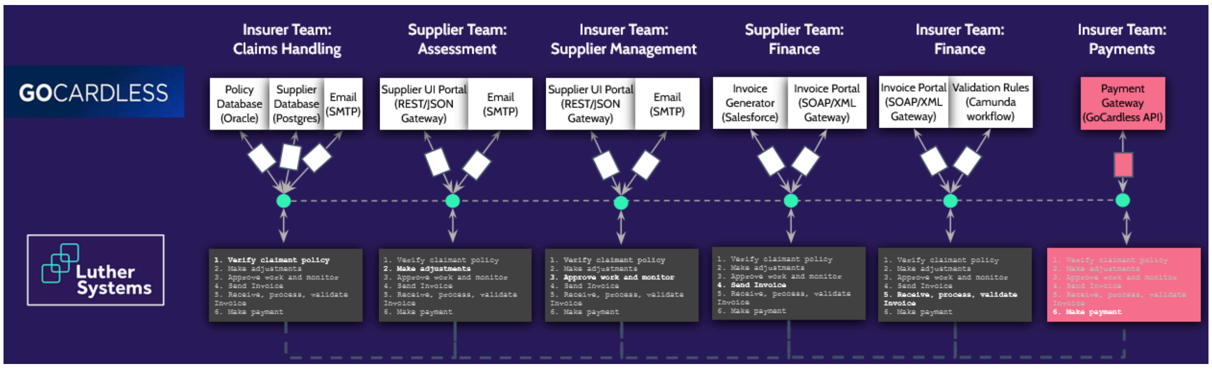
I. Platform sends (request) *supplier assessment information* via Supplier UI Portal [REST/JSON Gateway] to Supplier Team: Assessment
II. Platform receives (response) *supplier assessment information* via Supplier UI Portal from Supplier Team: Assessment
III. Platform validates supplier assessment information based on a predetermined rules in the Common Operations Script
IV. Platform shares and stores verification of assessment from Supplier UI Portal
V. Platform evaluates & Initiate next step


Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment
Luther Systems

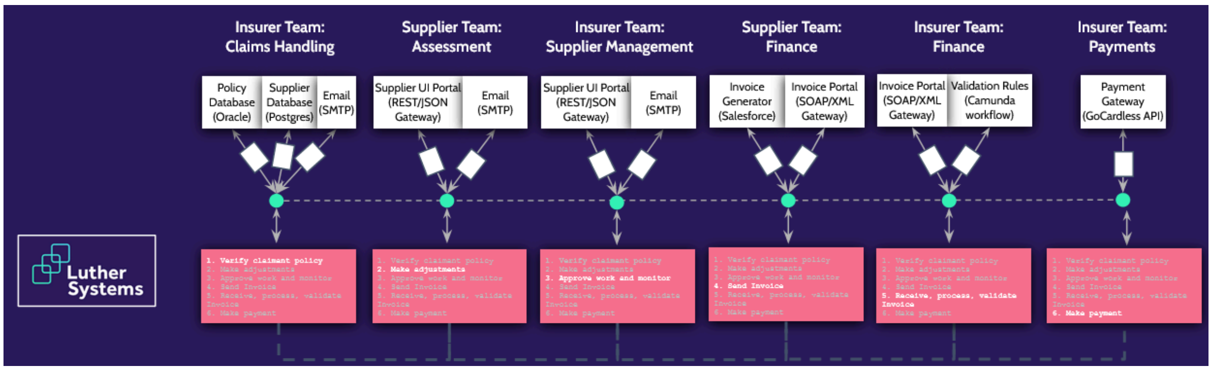
The steps operate in a similar manner until the final step is reached:

Step 12: Insurer Payments Team executes Make Payment (make payment to supplier)

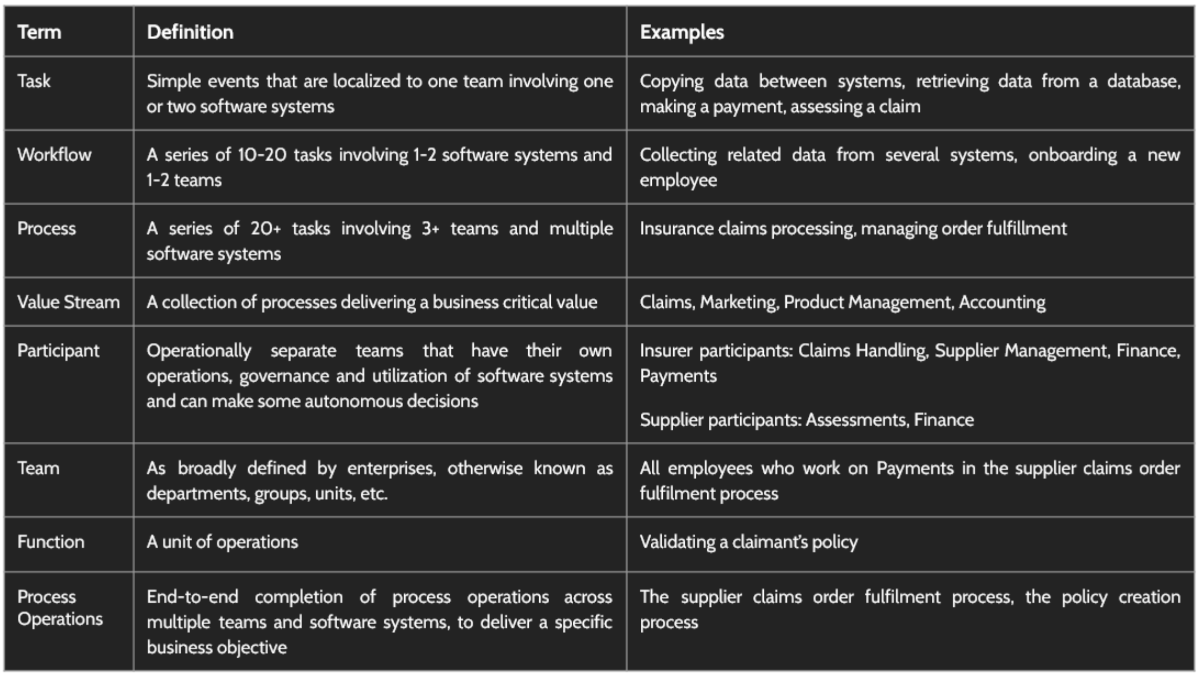
I. Platform sends (request) *payment information* to Payment Gateway [GoCardless API]
II. Platform receives (response) *payment information* from Payment Gateway
III. Platform validates payment information based on a predetermined set of rules in the Common Operations Script
IV. Platform shares and stores payment information from Payment Gateway
V. Platform evaluates & Initiate next step


GOCARDLESS
Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment
Luther Systems

The Platform completes the process:


Insurer Team: Claims Handling
Supplier Team: Assessment
Insurer Team: Supplier Management
Supplier Team: Finance
Insurer Team: Finance
Insurer Team: Payments
Policy Database (Oracle)
Supplier Database (Postgres)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Supplier UI Portal (REST/JSON Gateway)
Email (SMTP)
Invoice Generator (Salesforce)
Invoice Portal (SOAP/XML Gateway)
Invoice Portal (SOAP/XML Gateway)
Validation Rules (Camunda workflow)
Payment Gateway (GoCardless API)
1. Verify claimant policy
2. Make adjustments
3. Approve work and monitor
4. Send Invoice
5. Receive, process, validate Invoice
6. Make payment
Luther Systems

10.2. Definitions



Term	Definition	Examples
Task	Simple events that are localized to one team involving one or two software systems	Copying data between systems, retrieving data from a database, making a payment, assessing a claim
Workflow	A series of 10-20 tasks involving 1-2 software systems and 1-2 teams	Collecting related data from several systems, onboarding a new employee
Process	A series of 20+ tasks involving 3+ teams and multiple software systems	Insurance claims processing, managing order fulfillment
Value Stream	A collection of processes delivering a business critical value	Claims, Marketing, Product Management, Accounting
Participant	Operationally separate teams that have their own operations, governance and utilization of software systems and can make some autonomous decisions	Insurer participants: Claims Handling, Supplier Management, Finance, Payments Supplier participants: Assessments, Finance
Team	As broadly defined by enterprises, otherwise known as departments, groups, units, etc.	All employees who work on Payments in the supplier claims order fulfilment process
Function	A unit of operations	Validating a claimant's policy
Process Operations	End-to-end completion of process operations across multiple teams and software systems, to deliver a specific business objective	The supplier claims order fulfilment process, the policy creation process