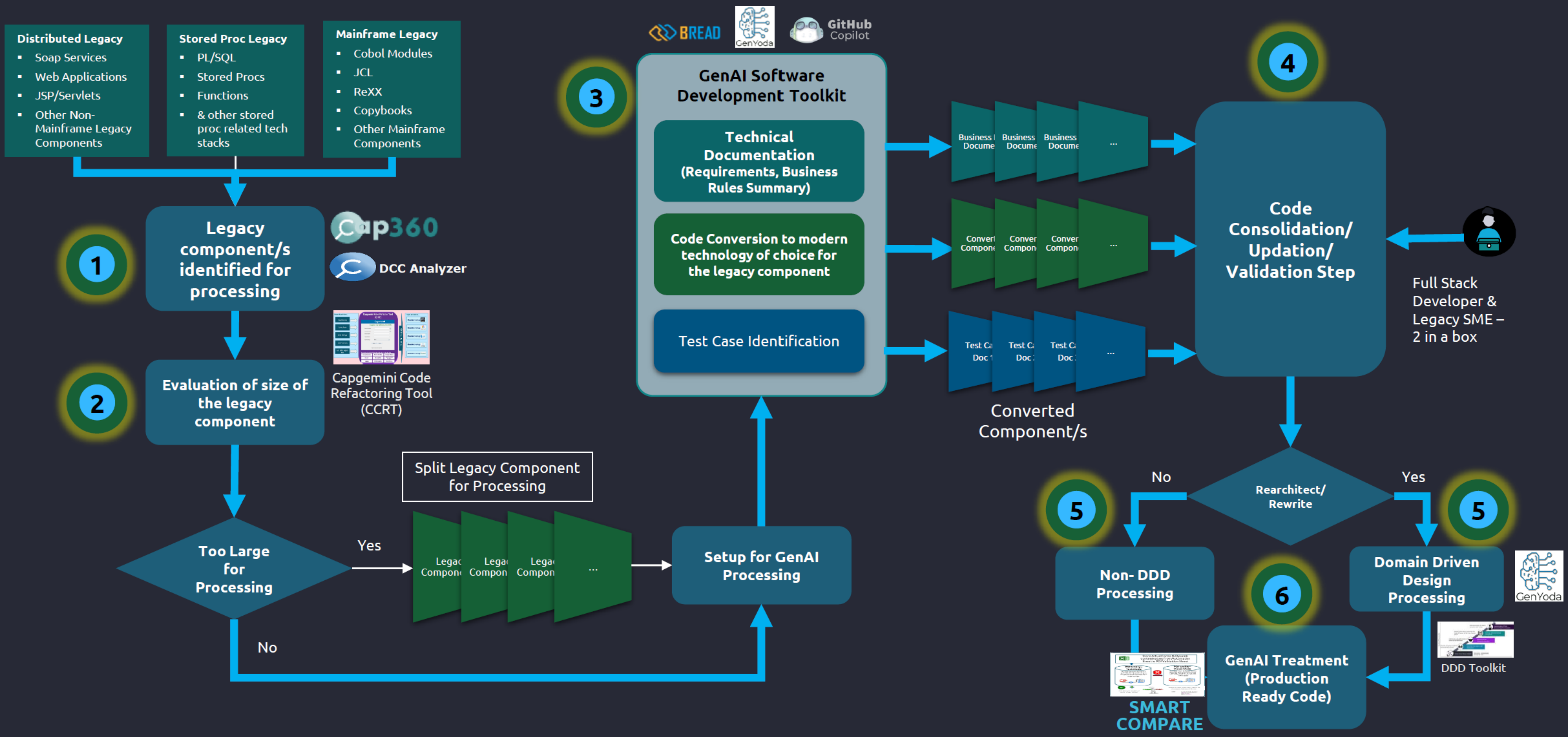
Mainframe Code Conversion Journey

Capgemini

Legacy modernization workbench in action

Legacy distributed, stored proc and mainframe legacy modernization



Distributed Legacy
- Soap Services
- Web Applications
- JSP/Servlets
- Other Non-Mainframe Legacy Components

Stored Proc Legacy
- PL/SQL
- Stored Procs
- Functions
- & other stored proc related tech stacks

Mainframe Legacy
- Cobol Modules
- JCL
- ReXX
- Copybooks
- Other Mainframe Components

1 Legacy component/s identified for processing

Cap360

DCC Analyzer

Capgemini Code Refactoring Tool (CCRT)

2 Evaluation of size of the legacy component

Too Large for Processing

Yes

No

Split Legacy Component for Processing

Legac Compon | Legac Compon | Legac Compon | ...

Setup for GenAI Processing

BREAD

GenYoda

GitHub Copilot

3 GenAI Software Development Toolkit

Technical Documentation (Requirements, Business Rules Summary)

Code Conversion to modern technology of choice for the legacy component

Test Case Identification

Business Docume | Business Docume | Business Docume | ...

Convert Compone | Conver Compon | Conver Compon | ...

Test Ca Doc 1 | Test Ca Doc 2 | Test Ca Doc 3 | ...

Converted Component/s

4 Code Consolidation/ Updation/ Validation Step

Full Stack Developer & Legacy SME – 2 in a box

Rearchitect/ Rewrite

No

Yes

5 Non- DDD Processing

SMART COMPARE

5 Domain Driven Design Processing

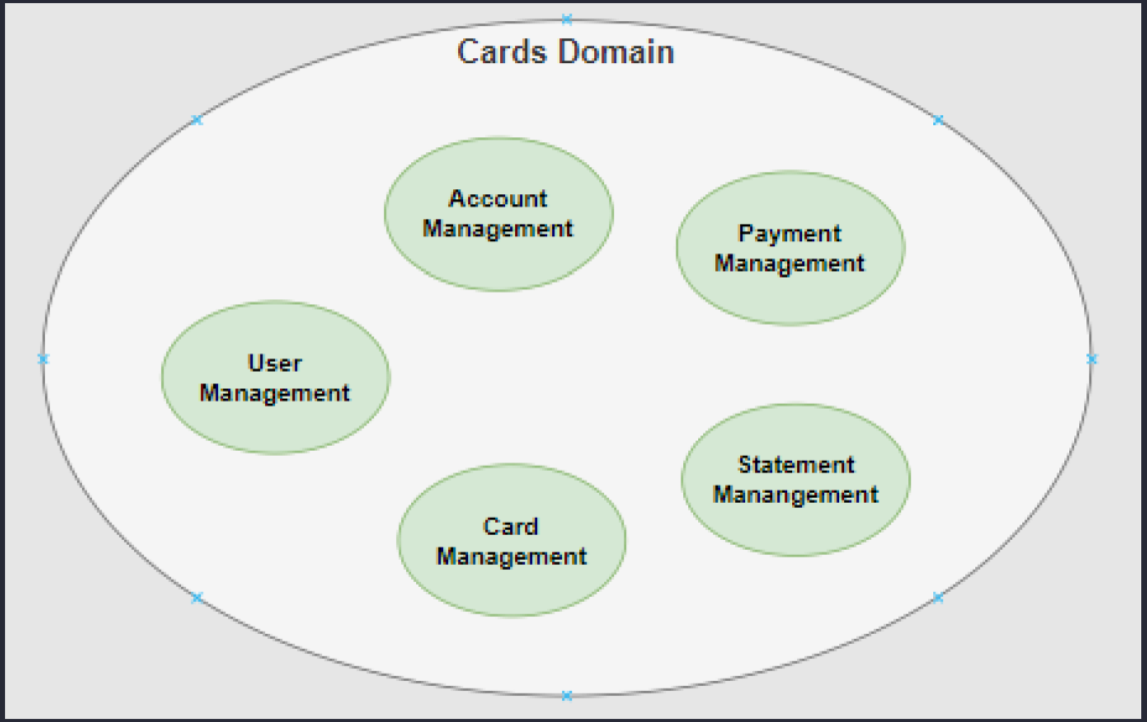
GenYoda

DDD Toolkit

6 GenAI Treatment (Production Ready Code)

DDD – STRATEGIC DESIGN

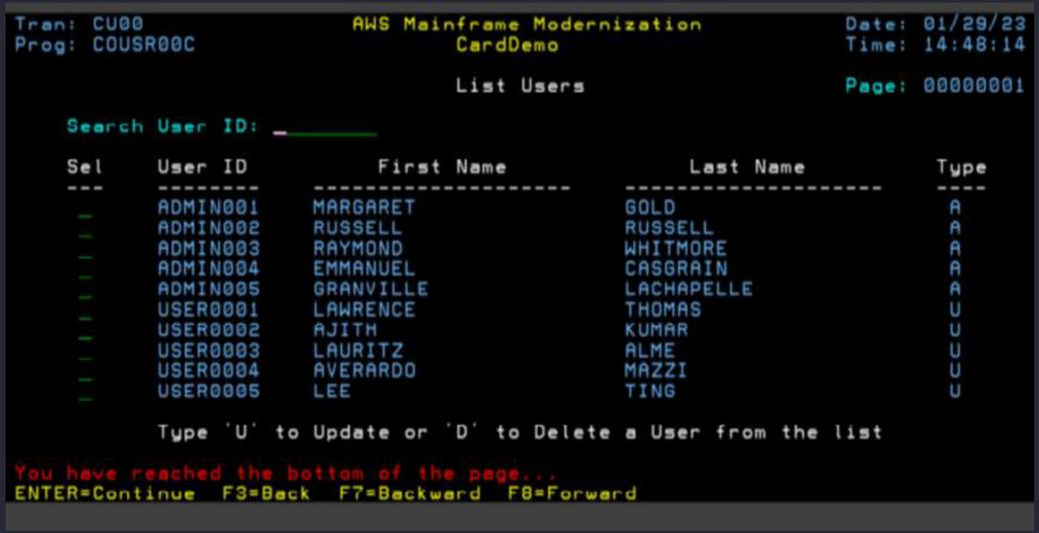
Cards Demo Context Map



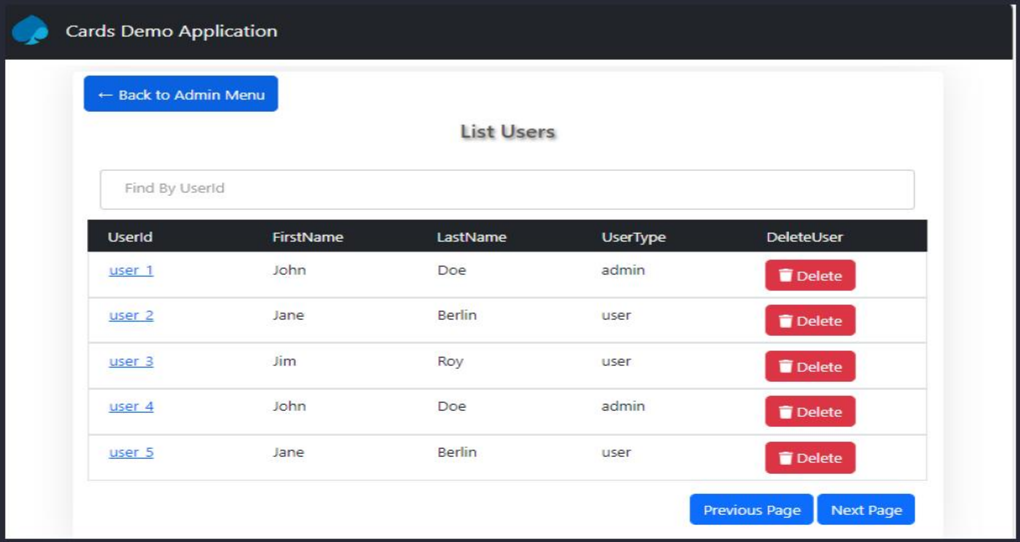
Summary

- Bounded Contexts represent the boundaries within a subdomain where a particular model (model and behaviors) applies.
- 5 bounded contexts (highlighted blue in the context map) are identified & evolve independently
- All components of a bounded context will be implemented together - Online, UI & batch
- Transaction, Reporting data & behavior is included in each bounded context
- Domain entities, API models will evolve as new attributes & value objects are discovered

GENAI GENERATED UI



```
Tran: CU00                AWS Mainframe Modernization                 Date: 01/29/23
Prog: COUSR00C                        CardDemo                        Time: 14:48:14

                                   List Users                         Page: 00000001

    Search User ID: ______

    Sel     User ID          First Name                Last Name              Type
    ---     --------    --------------------    --------------------          ----
     _      ADMIN001    MARGARET                GOLD                           A
     _      ADMIN002    RUSSELL                 RUSSELL                        A
     _      ADMIN003    RAYMOND                 WHITMORE                       A
     _      ADMIN004    EMMANUEL                CASGRAIN                       A
     _      ADMIN005    GRANVILLE               LACHAPELLE                     A
     _      USER0001    LAWRENCE                THOMAS                         U
     _      USER0002    AJITH                   KUMAR                          U
     _      USER0003    LAURITZ                 ALME                           U
     _      USER0004    AVERARDO                MAZZI                          U
     _      USER0005    LEE                     TING                           U

            Type 'U' to Update or 'D' to Delete a User from the list

You have reached the bottom of the page...
ENTER=Continue  F3=Back  F7=Backward  F8=Forward
```



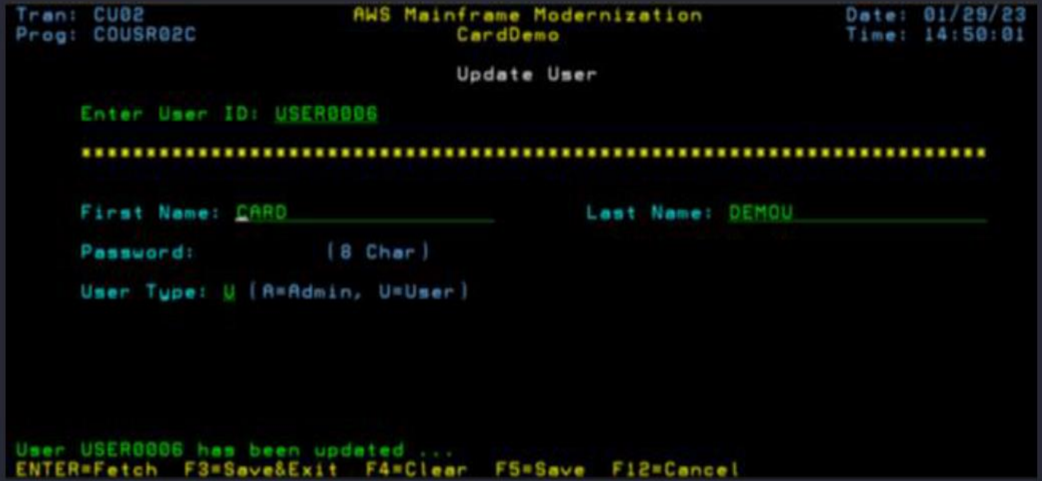
Cards Demo Application

← Back to Admin Menu

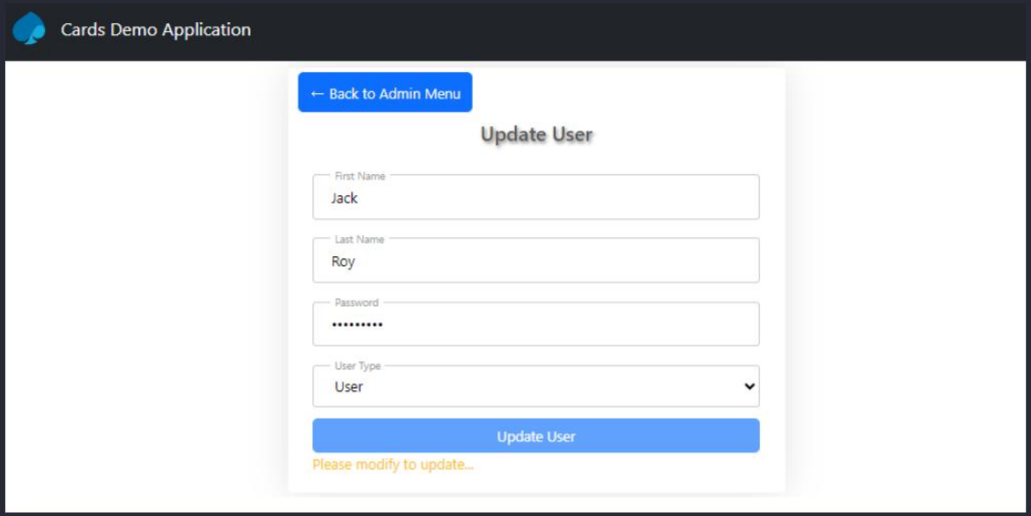
List Users

Find By UserId

UserId	FirstName	LastName	UserType	DeleteUser
user 1	John	Doe	admin	Delete
user 2	Jane	Berlin	user	Delete
user 3	Jim	Roy	user	Delete
user 4	John	Doe	admin	Delete
user 5	Jane	Berlin	user	Delete



```
Tran: CU02                AWS Mainframe Modernization                 Date: 01/29/23
Prog: COUSR02C                        CardDemo                        Time: 14:50:01

                                   Update User

      Enter User ID: USER0006

      ************************************************************************

      First Name: CARD                              Last Name: DEMOU

      Password:              (8 Char)

      User Type: U (A=Admin, U=User)



User USER0006 has been updated ...
ENTER=Fetch  F3=Save&Exit  F4=Clear  F5=Save  F12=Cancel
```



Cards Demo Application

← Back to Admin Menu

Update User

First Name: Jack

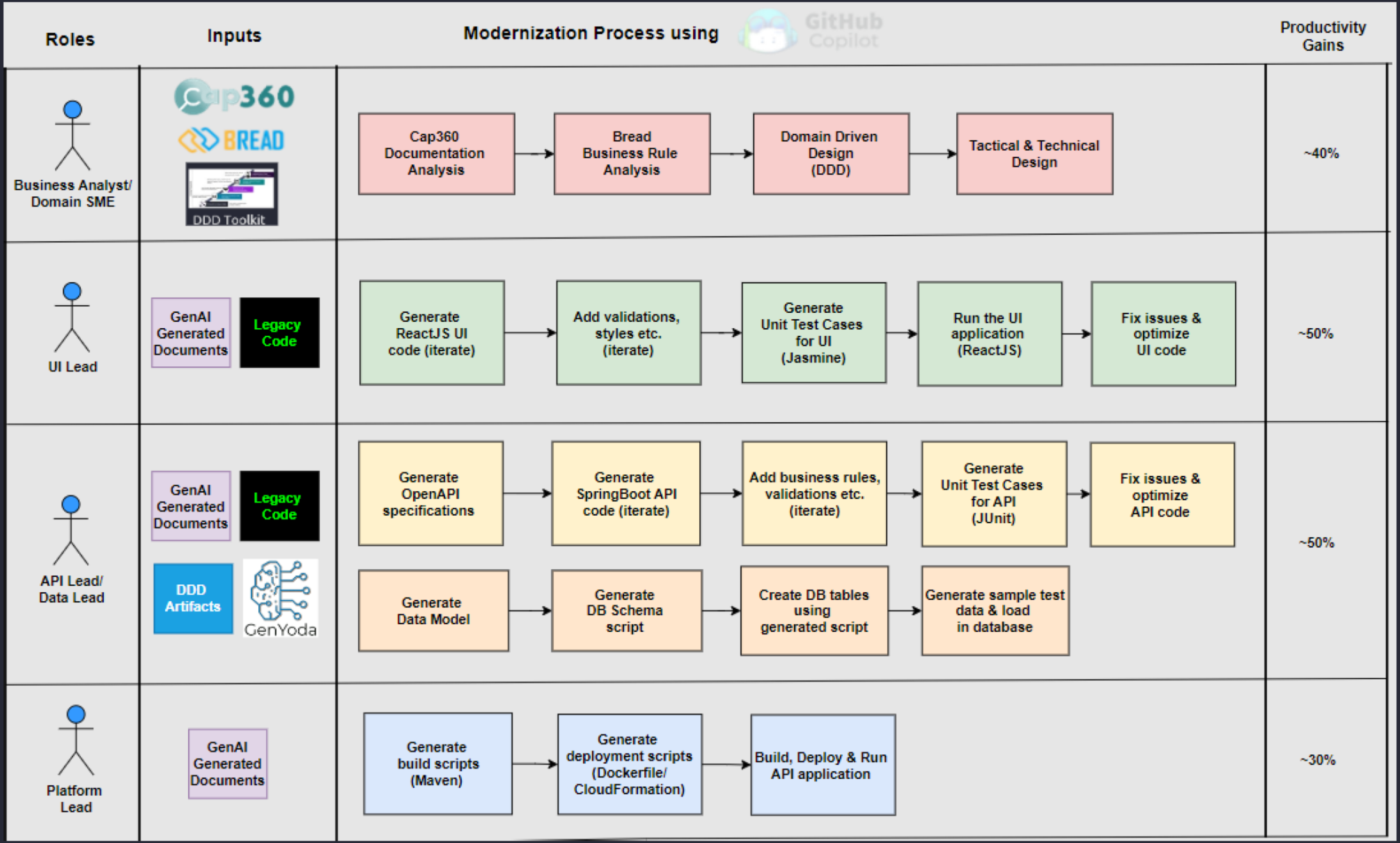
Last Name: Roy

Password: •••••••••

User Type: User

Please modify to update...

MAINFRAME CODE CONVERSION JOURNEY …



Roles	Inputs	Modernization Process using GitHub Copilot	Productivity Gains
Business Analyst/ Domain SME	Cap360 BREAD DDD Toolkit	Cap360 Documentation Analysis → Bread Business Rule Analysis → Domain Driven Design (DDD) → Tactical & Technical Design	~40%
UI Lead	GenAI Generated Documents Legacy Code	Generate ReactJS UI code (iterate) → Add validations, styles etc. (iterate) → Generate Unit Test Cases for UI (Jasmine) → Run the UI application (ReactJS) → Fix issues & optimize UI code	~50%
API Lead/ Data Lead	GenAI Generated Documents Legacy Code DDD Artifacts GenYoda	Generate OpenAPI specifications → Generate SpringBoot API code (iterate) → Add business rules, validations etc. (iterate) → Generate Unit Test Cases for API (JUnit) → Fix issues & optimize API code Generate Data Model → Generate DB Schema script → Create DB tables using generated script → Generate sample test data & load in database	~50%
Platform Lead	GenAI Generated Documents	Generate build scripts (Maven) → Generate deployment scripts (Dockerfile/ CloudFormation) → Build, Deploy & Run API application	~30%

OBSERVATIONS

Business Analyst :

- GitHub Copilot was used to extract business requirements/rules from the legacy mainframe code & produce summary, this was done ~40% faster than traditional approach by reading the code line by line

- Dependency on the legacy SMEs was reduced by ~50%, routine business rules were extracted, detailed & clarified by prompting using GitHub Copilot. SMEs were primarily involved for evaluating the efficacy of the generated content/code by GitHub Copilot

UI/API Developer:

- CICS/Online code required reengineering by building separate UI & API components. GitHub Copilot generated business rules served as prompts to generate the base UI & API code.

- 70% of the new code (ReactJS UI & Spring Boot API) is generated using GitHub Copilot. UI code was developed by backend Java developers without prior ReactJS knowledge by using GitHub Copilot

- GitHub Copilot was used by developers with framework assistance, syntax, fixing issues, refactoring & optimizing without need to spend time researching online, resulted in ~50% productivity improvement

- 60% of the batch job code was generated using GitHub Copilot, breaking legacy code into multiple components if files are large & then combining them. Manual intervention was only needed to make the code compile & functional, without GitHub Copilot entire code should be written from scratch.

OBSERVATIONS CONT…

- ~90% of the Junit test cases for the API code are generated by GitHub Copilot. Test cases for ReactJS UI were also generated using GitHub Copilot

- Base API, data & domain models are generated from Cobol, Copybooks using GitHub Copilot

- Java documentation is generated automatically using GitHub Copilot

DevOps Engineer:

- OpenAPI specifications, build & deployment scripts were generated by GitHub Copilot which otherwise are manually coded

LEGACY & MAINFRAME MODERNIZATION - OUTCOMES

Code Category	Code Conversion from Cobol to Java		Efficacy of the Transformed Code		Expected Efficiency Gains	
	Conversion %	Observations	Efficacy %	Observations	Efficiency Gain %	Observations
Cobol Programs	60%	• Modules with extensive business logic showed high conversion rate of levels 80 to 90% • Modules with CICS interaction and/or linking to other modules showed a lower 30-40% of conversion rate • Modules with repetitive steps (e.g. SQL statements to delete multiple DB2 tables) observed to have the first instance accurately converted and needed manual reverse engineering for remaining statements	70%	• Converted Java code had higher accuracy with certain exceptions(Ex. Declaration of communication areas included as is in Java). • Conversion efficacy was also impacted due to Java specific coding optimization not applied on the converted Java code	45%	• Benefits found in leveraging converted code from Cobol to Java as it provides ~40% acceleration and standardization for developers • Business logic (documentation) pseudo code extracted enables the developer to accelerate the development
Copybooks	100%	• All the variables in the copybooks along with their datatypes & default values gets converted accurately	90%	• Converted all the Copybook entries to Java classes & variables, variable type declarations weren't very accurate	90%	• The converted copybooks can be used directly in the java program or as an external file with minimal changes needed
JCL	40%	• Groovy scripts is the better fit for JCL conversion for step-based processing • JCL with standard processing steps (program execution, sorting/processing files) showed higher conversion 70-80% rate • Groovy was observed to be not suitable in scenarios involving environment setup/ VSAM processing • JCLs used for functions like declaring & copying VSAM file/s and setting up environments was found to have very low conversion 10-15% rate	50%	• Conversion efficacy was impacted due to Groovy specific coding optimization not applied on the converted Groovy code	20%	• Extracted pseudo code documentation enables the developer to accelerate the development • Converted code from JCL to Groovy where standard processing was involved(program execution, sorting/processing files) helps to accelerate the development by 30-40%