MEMORANDUM

TO: CATHERINE BENHAM, CHIEF FISCAL OFFICER, JOINT FISCAL OFFICE

FROM: LISA M GAUVIN, IT CONSULTANT FOR THE JOINT FISCAL OFFICE

SUBJECT: THE FEASIBILITY OF CHANGING THE UNEMPLOYMENT INSURANCE MAINFRAME PROGRAM

DATE: NOVEMBER 1, 2021

Executive summary

This analysis provides detailed concerns about changing the mainframe UI program and identifies specific factors that contribute to risks posed by changes. This analysis also outlines unintended consequences to consider when attempting workarounds outside the UI program to enact desired changes.

The key findings presented here include the following:

- Any changes to the program are extremely risky and should be avoided. The reasons include: no way to safely make and test changes, no documentation, and limited access to skilled programmers.
- This is no fault of current staff but a result of using a 40-year-old program.
- It is important to recognize that if this program were written in a modern programming language and conformed to today's development and documentation standards, the expectations for changes within this program would be wholly justified.
- The new modernized program must have ease of use, accessibility, security, and rigorous protection against fraud. It must also provide flexibility to *enable* the policy vision of state leaders.

Explanation of the key findings

In recent weeks, JFO has received multiple requests for the legislative IT consultant to consider the feasibility and risks of changing the Vermont Department of Labor's (VDOL) unemployment insurance (UI) mainframe program to support enacted and proposed changes. Outgoing IT Consultant for the Joint Fiscal Office, Dan Smith, and I assessed these proposed changes and describe them in detail below.

After careful analysis, Dan Smith and I each concluded that changes to the state mainframe UI program pose a high degree of risk and should be avoided for the following reasons:

- The mainframe UI program does not have the traditional environments for development and testing that allow safe changes to be made in modern systems[1]. This is highly unusual, if not unique, for a critical citizen-accessed system in this state.
- The lack of these environments should restrain the state from making changes to code because of the high risk to day-to-day business conducted using the UI program. These risks include, but are not limited to:
    - the inability for citizens, employers, or state staff to access the UI program for an unpredictable amount of time,
    - corrupt data,
    - data loss,
    - incorrect calculations.
- There is no documentation of the UI program to inform code changes. This lack of documentation means that a change in program code could have unpredictable results, including system failure. If the code impacts other data, processes, or calculations, programmers may not realize that a change caused an error in another part of the program until a later date.
- The state does not have staff skilled in F-COBOL, the code used in the UI program. The state relies on contractors to address changes or issues, and there is no documentation to guide them.
- The findings in this analysis are not the result of the negligence of the current staff, but an outcome of letting an essential system operate without an upgrade for 40 years.
- As state staff works on the specification for the new modernized system, their intent is to include flexibility to enable future policy changes. I have already agreed to work with them on this.
- We advise state leaders to be aware that efforts to change the UI program, whether in the existing UI program itself or through workarounds outside it, may have the unintended consequence of diverting staff away from work on the new UI system.

For more information about the assessment of the environments in use by the UI program, please see Addendum 1 of this document.

Factors that contribute to the risk of making changes

After careful analysis of the documents and discussions, Dan Smith and I identified the following factors that are the basis of why changes to the UI program are risky:

- Technical limitations –There are no development or **true** test environments set up to develop and test a revised UI program. *Please see ADDENDUM 1 of this memo for more details about this statement.*
  Regarding changes related to the extra $25 payment, more analysis would be needed to confirm changes did not exceed the character screen limits of the existing programming language. (This refers to the number of characters allowed on the old "green screen" used

[1] The use of the word "systems" versus "program" is intentional. Modern information systems are made up of application, database, reporting and authentication servers and are best describes as "systems" versus the limited components of the existing UI program.

by F-COBOL.) According to Deputy Secretary Nailor, there are finite limits to expanding this part of the program, and it is unclear if this change would reach those limits.

- Documentation limitations – There is no documentation of the UI program. Documentation is required for anyone to make changes in a safe and timely manner. Deputy Secretary Nailor is exploring if old penalty week code in the UI program can be reused. He noted that it was a problem that they didn't know the reason why the code had been removed. It could relate to problems with functionality in the program, or it could have been a policy directive, or still another technical reason. *(See the next section in this document, "How we approached this feasibility and risk analysis," for the context of the penalty week information.)*
- Human resource limitations – VDOL doesn't have the staff skilled in the F-COBOL programming language used to write the UI program interface. The state is dependent on contractors who know this language, but without documentation, even experienced F-COBOL programmers cannot know the impact of the changes they make. It was relayed to me that if a change in the code is required, VDOL contacts ADS, who contacts a contractor, who makes the change, notifies ADS, who then notifies VDOL the change is complete. In addition, this team is also involved with documenting requirements and developing an RFP for the replacement system. Deputy Secretary Nailor also shared that ADS has a staff of 16 assigned to VDOL, but currently has six open positions. Human resource limitations should be carefully considered when asking for workarounds to make desired changes. In many cases, the same staff is charged with developing the RFP for the new UI program.
- Schedule limitations – Even if the above limitations could be resolved, it would not be possible to make changes this fiscal year. This is particularly relevant for the extra $25 benefit.

Dan Smith and I believe **each** of the above findings would be enough to justify a decision not to alter the UI program.

It has been shared with me that VDOL changed the maximum benefit amount in the program successfully in the last year or two. A more recent change caused errors due to staff changes and the lack of documentation of the code that should be adjusted when this change is made. This is a reminder that successful changes in the past do not guarantee future success.

We conclude that the risks associated with making code changes without the best practice use of dedicated development, test, and production environments coupled with the challenges of recovering a critical 40-year-old mainframe program (if a change causes the system to go down) within an acceptable timeframe, are significant. We advise against changes to the UI program.

How we approached this feasibility and risk analysis

To assess the risks that changes to the UI program presented, Dan Smith and I looked at two specific proposed changes.

In one instance, we were asked to assess VDOL's decision not to alter the UI program to distribute the extra $25 benefit authorized in Act 51.

In another instance, I looked at the risk of changing the existing penalty weeks functionality in the program. In a recent meeting with the Unemployment Insurance Study Committee, Deputy Secretary Shawn Naylor committed to ***exploring*** the possibility of a change to the penalty weeks

functionality in the program. This commitment involved assessing inactive code in the program that, on the surface, appeared to make the requested change to penalty weeks functionality.

When examining the risk of changing the mainframe program, Dan and I did the following:

- Dan reviewed Act 51, documents from legislative testimony, and Commissioner Harrington's letter to Sen. Balint of 9/1/2021.
- Dan had discussions with Commissioner Michael Harrington and Cameron Wood, UI Program Director.
- I reviewed Dan's findings and had remaining technical questions, which I discussed with ADS Deputy Secretary Shawn Nailor. This discussion was regarding the decision not to change the program to accommodate the extra $25 payment included in Act 51.
- I reviewed Deputy Secretary Nailor's testimony to the Unemployment Insurance Study Committee on October 19th regarding penalty weeks.
- I met with Deputy Secretary Nailor on October 26th to ask additional technical questions about the program environment.

Next Steps

It is important to recognize that if this program were written in a modern programming language and conformed to today's development and documentation standards, the expectations for changes within this program would be wholly justified.

The only acceptable outcome of this situation will be the implementation of a new UI system that meets the 21st-century expectations of Vermont citizens, which includes ease of use, accessibility, security, and rigorous protection against fraud. It must also provide flexibility to *enable* the policy vision of state leaders.

Deputy Secretary Nailor is very supportive of the idea of working together to ensure the upcoming RFP includes requirements that will provide flexibility to support the type of changes suggested by the UI Study Group.

Please reach out if you have any questions about this analysis or its conclusions.

ADDENDUM 1: IMPACT OF MAINFRAME UI ENVIRONMENTS ON RISK

A modern system includes, at a minimum, three environments and sometimes a fourth pre-production environment. In its most basic terms, an environment consists of a program interface and a database where data is input via the program interface. Each environment of a modern system usually includes an application server, database server, authentication server, and a reporting server. This contrasts with the outdated UI program, which is simplistic in nature.

The three typical environments (with optional fourth listed) are:

- **Development** – Environment where developers write and change code and conduct preliminary testing.
- **Test** –Environment that is a replica of the production environment. Developers move changes from the development environment to the test environment, and rigorous testing is done here. Because the test environment is set up like production, there is relative assurance that once an altered or new portion of code passes testing, it will work when moved to production.
- **Pre-Production (optional)** – **This is usually used in complex systems only** – where you may have numerous complex modules that must work together. In the test environment you would test the module, in pre-production you would test the intersection points between all modules as a final test.
- **Production** – This is where "live" data and current programs are in use.

The current mainframe UI system environment is set up differently:

- **Development** – Does not exist
- **Test** – Does not exist
- **Pre-Production** – The UI program includes an **Edit Environment** that I would classify as a hybrid pre-production environment. I would classify it like this because its purpose is to ensure edited data from the production environment is "checked" for errors before the corrected data is "fixed" in production. This means it is a required production process that is most likely used often if not every day.
- **Production Environment** – Does exist

What is the purpose of the UI program edit environment and why is it needed?

This UI edit environment was described as the environment where "edits" are made to production data because the production interface does not allow edits via the user interface. This means all edits must be made within the data stored in the system. The UI program doesn't store information in discreet fields but long strings of data. To illustrate this, I created this fictional string of data, but for UI, the data string is probably hundreds of characters long.

0206034501M093455 JOHNMDOE123MAINSTREETBARREVT0909334K988888

If incorrect data is entered into the production environment, a programmer must go into the long string of data and correct the inaccurate data. The current process is to move the string to the test environment, change the incorrect data in the string, and then load all updated data overnight via a batch file process. If the corrected data does not trigger errors during the overnight run, a programmer then queues the strings of corrected data into production using the same process.

Why is all of this important?

I am sharing this information to make this important point. The current UI environment doesn't include a development environment and a TRUE test environment. Best practices require testing changes to codes in a dedicated test environment.

The UI edit environment, which I classified as a pre-production environment, should be considered part of a production process because it is the only way to test edits before loading them in production. This process is probably used often if not daily.

If the state uses this environment to test code changes to enact a policy change, it risks making the environment unavailable for the typical daily edits. What happens then? Would programmers risk loading untested changes directly in the production environment? In addition, because there is no documentation, it is possible that a change could cause a problem that might not be evident until calculations or reports are run – and then how would it be possible to back out the changes?

Not having a true development and test environment risks bringing down the entire UI system for an unpredictable amount of time.