Putting a Brain at the Heart of DQMH

Secretary of State vs. Enemy of the State

Norm thinks state is EVERYTHING.

DNatt could take it or leave it.

#MakeQMHGreatAgain

Norm Kirchner, CLA, NI

Chief ~~TSE~~ TLB’ Fanatic

Darren Nattinger, CLA, NI

Principal ~~TSE~~ DQMH Zealot

Presentation Goal

To improve the way we think about and design our QMH-based LabVIEW programs in two ways:

1. Understand Norm’s ever-present co program flow and how it might affec
2. Understand the QMH Prime(’) desig
3. If all else fails…to entertain

Agenda

- Presentation goal
- What is “state”, exactly?
- Problem statement
- Norm’s solution
- DNatt’s counterpoint
- Norm’s counter-counter point
- Conclusion

Before we get started:

‘Norman-clature’ & 'Natt-onyms'

“Clear as Chocolate Milk”

~Scott Menjoulet

State ~= Mode

State != State Data

Actions = Messages ~= Requests

Actions & Message != State

Do = Action & Message-ish

Brain(s) = State Machine

QMH-based LabVIEW program?

~~Producer/Consumer~~

NI QMH

DQMH

Actor Framework

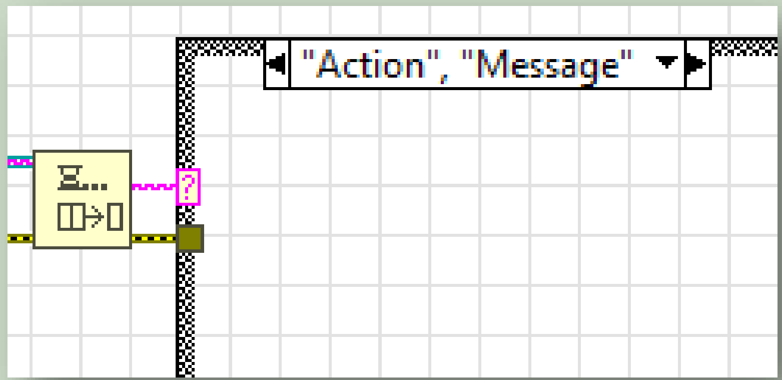
TLB, TLB Prime, TLB` 2.0, TLB Full Tilde

JKI State Machine

…

Pretty much everything!

For the purposes of this presentation, a "QMH-based LabVIEW program" is any pattern or framework that uses a queue or a queue-like mechanism to execute a series of commands from one iteration of a loop to the next.



A quick aside on Template Selection

- This presentation is NOT a “my framework is better than yours” presentation
- Lots of people have been successful with lots of different frameworks
- DQMH comes naturally to Fabiola, Matthias, Olivier, DNatt... and that’s ok
- Actor Framework comes naturally to Stephen, Allen, Derek, Casey… and that’s ok
- TLB’ comes naturally to Norm and … , and that’s ok

What is Top-Level Baseline Prime? (TLB`)

- QMH template (baseline) with a bunch of extra design considerations to aid in 'top level application' creation

- With additional explicit state machine (the prime directive) to functionally contain flow logic and authority over program behavior

- Addresses multitude of common issues by recommending certain design patterns/methods

- “Sophisticated stateful trigger response mechanism with a shitload of white-space” --DNatt

What is DQMH?

- Event-based framework for LabVIEW application development
- Free download on the NI Tools Network
- Maintained by the DQMH Consortium (dqmh.org)
- Built-in productivity tools to decrease development time
- Calling code sends “requests” to DQMH modules
- DQMH modules “broadcast” information to any interested party
- External communication to/from DQMH modules implemented via User Events
- Internal communication within DQMH module implemented via QMH
- “Ridiculously ‘tool rich’ QMH that fails to solve the big issue & thinks you’ll to fit code into **550x380** pixels” ~,~ NJKirchner

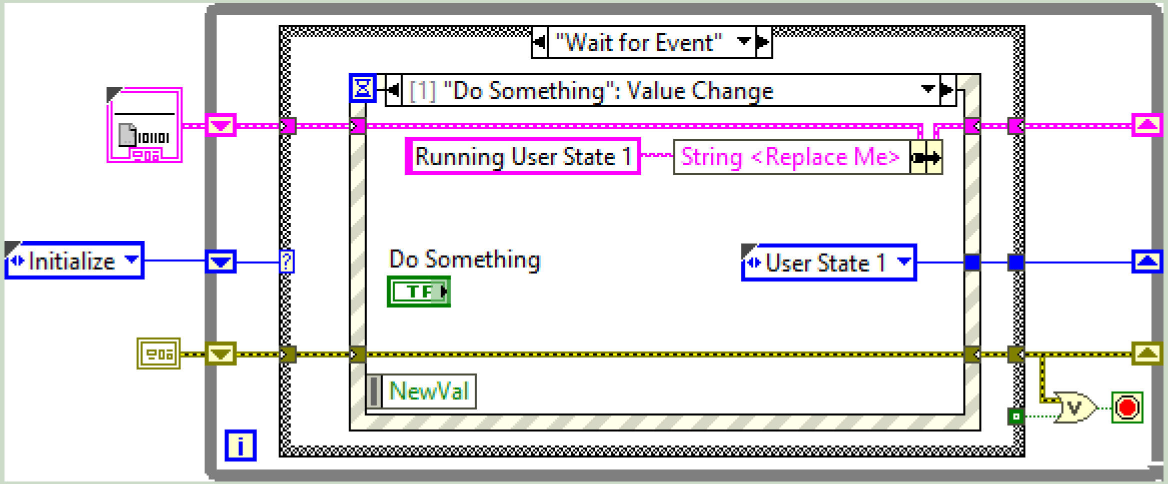
What is “state”?

The iPhone is not a phone.

osxdaily.com

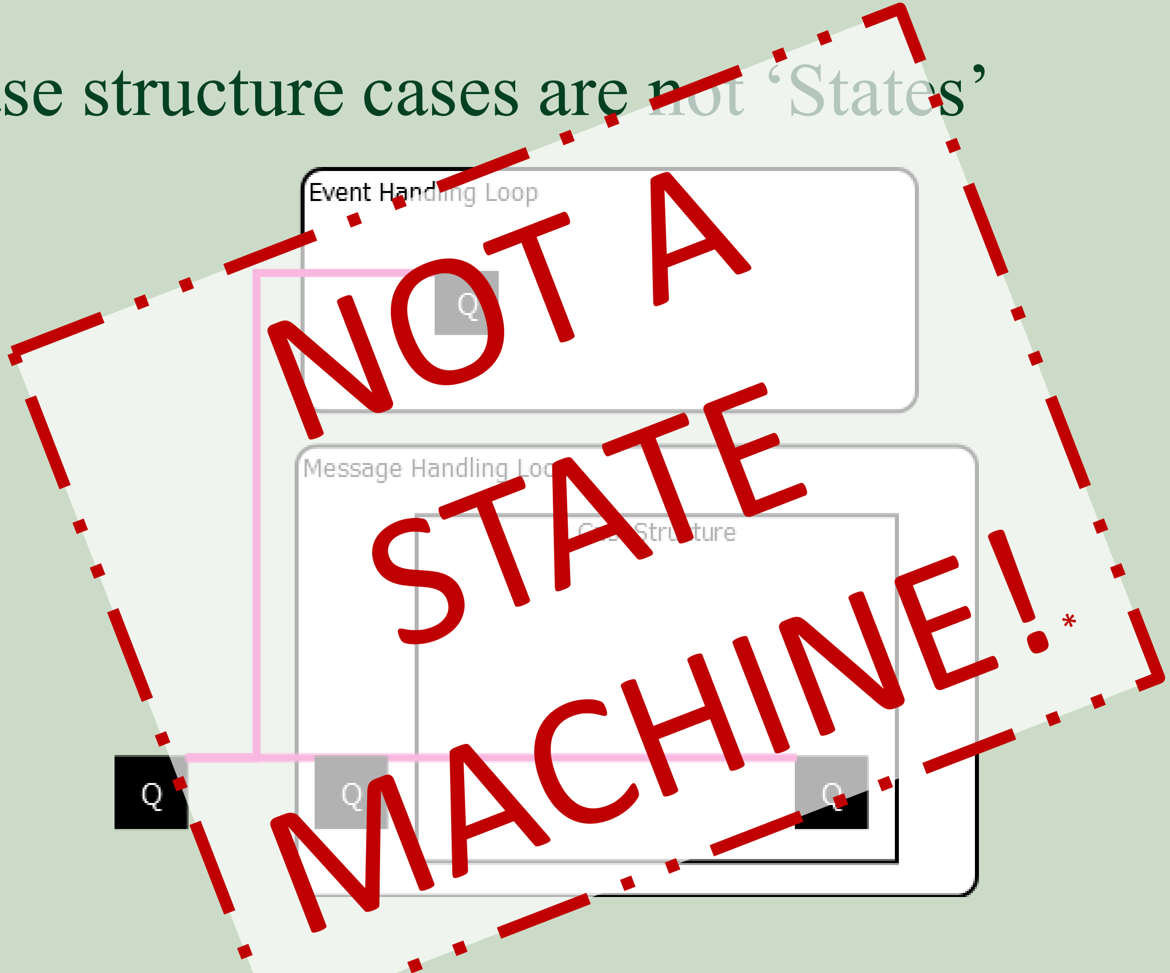
It’s a personal internet connectivity device that is used very occasionally as a phone.

The ‘Simple State Machine’ is not a state machine.

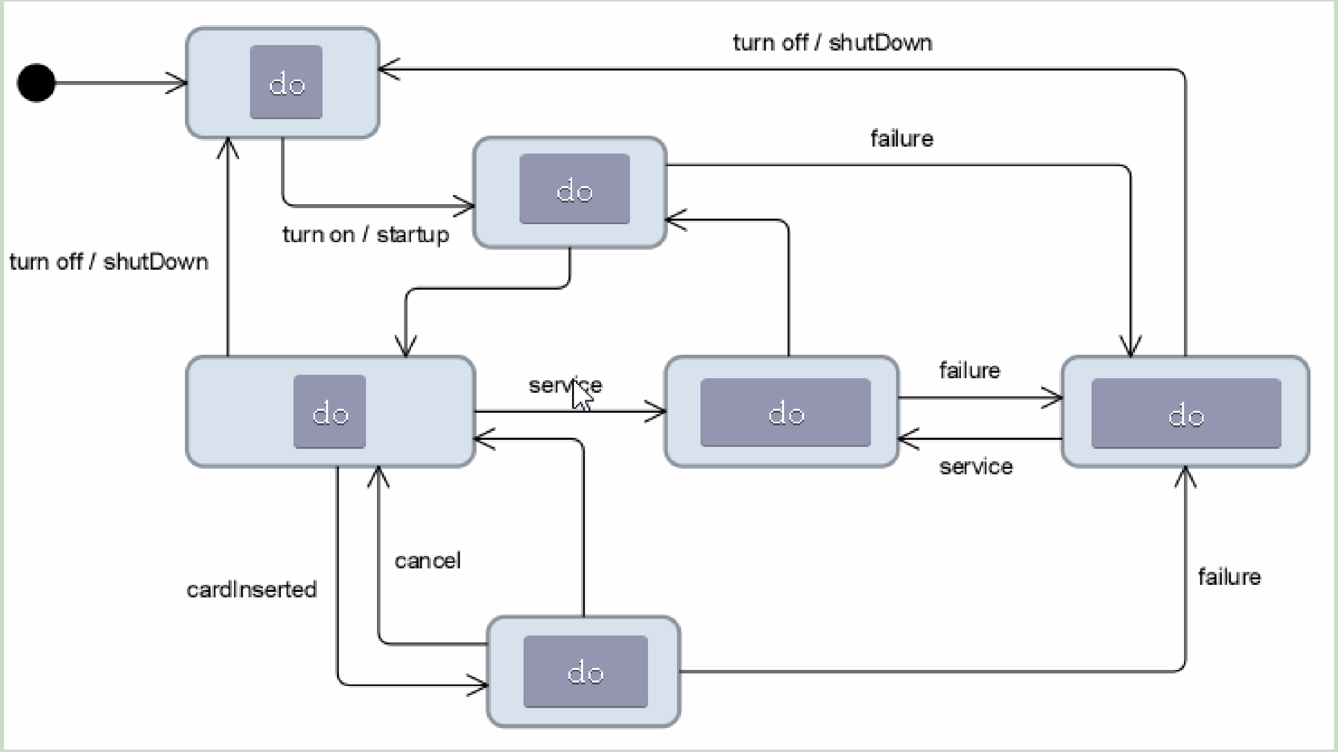


We refer to that enum as the “state” in a simple state machine. But really, it’s more like the “next command to run”. So the “Simple State Machine” is really more of a “Single Message Handler”… it can only handle one message (command) at a time.

QMH case structure cases are not 'States'



"your state machine has Do Do all over it"



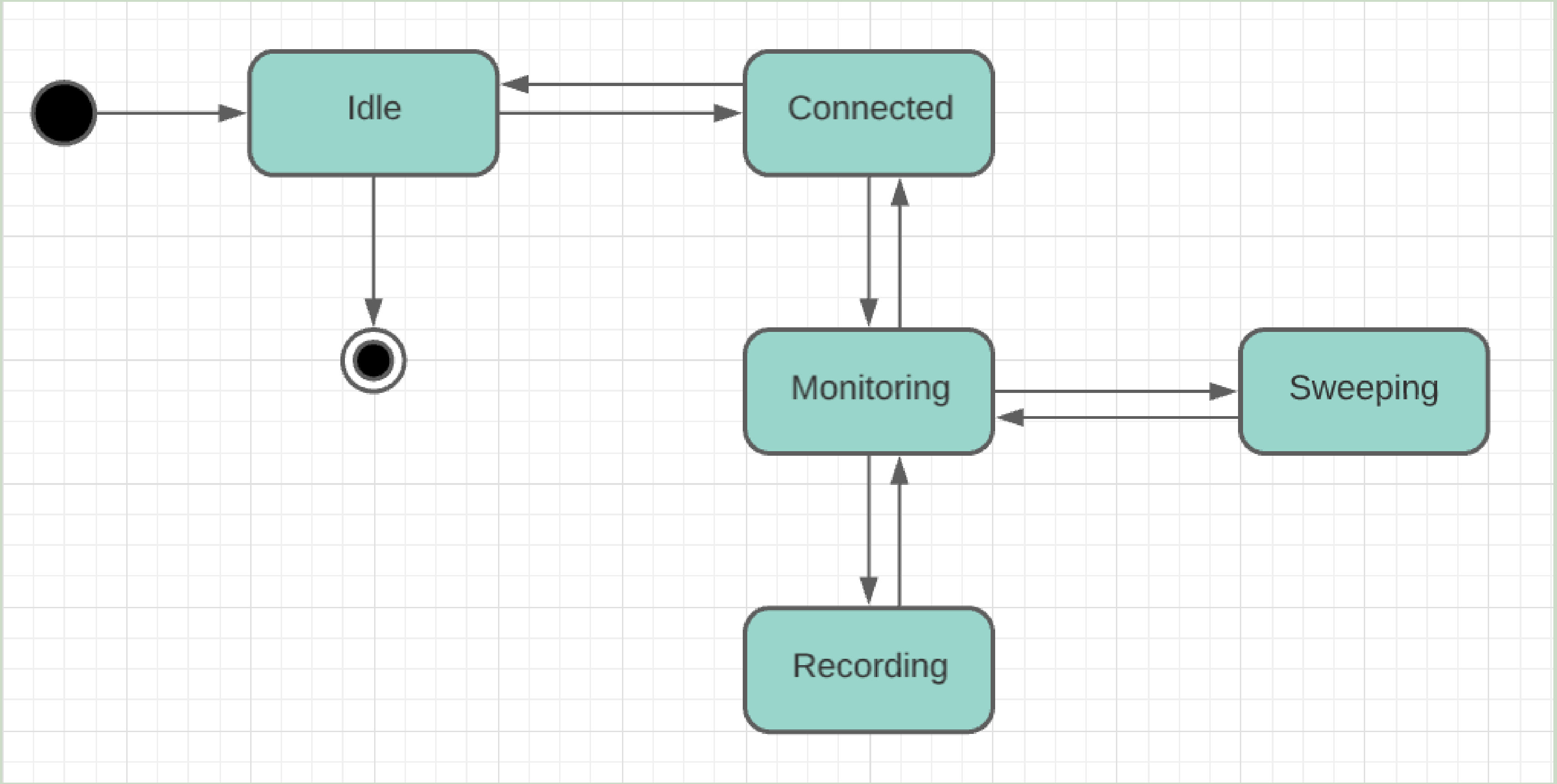
Better definition of state

A discrete mode of a given module which <u>classifies specific behavior</u> to *Stimulus* and *State Data*

- State of ‘Being’
  - All the **–ing**’s
- **Mode** of the program
  - Visual indications
- Functionally contained behaviors/responses
  - Discrete **brains**

<u>Example</u>: You could say “I am…”

- Idle(**ing**)
- Connected
- Monitor**ing**
- Logg**ing**
- Shutt**ing** Down
- Handl**ing** Error


Idle
Connected
Monitoring
Sweeping
Recording

Problem Statement

Problems with traditional QMH program flow

All but the most simplistic of QMH modules

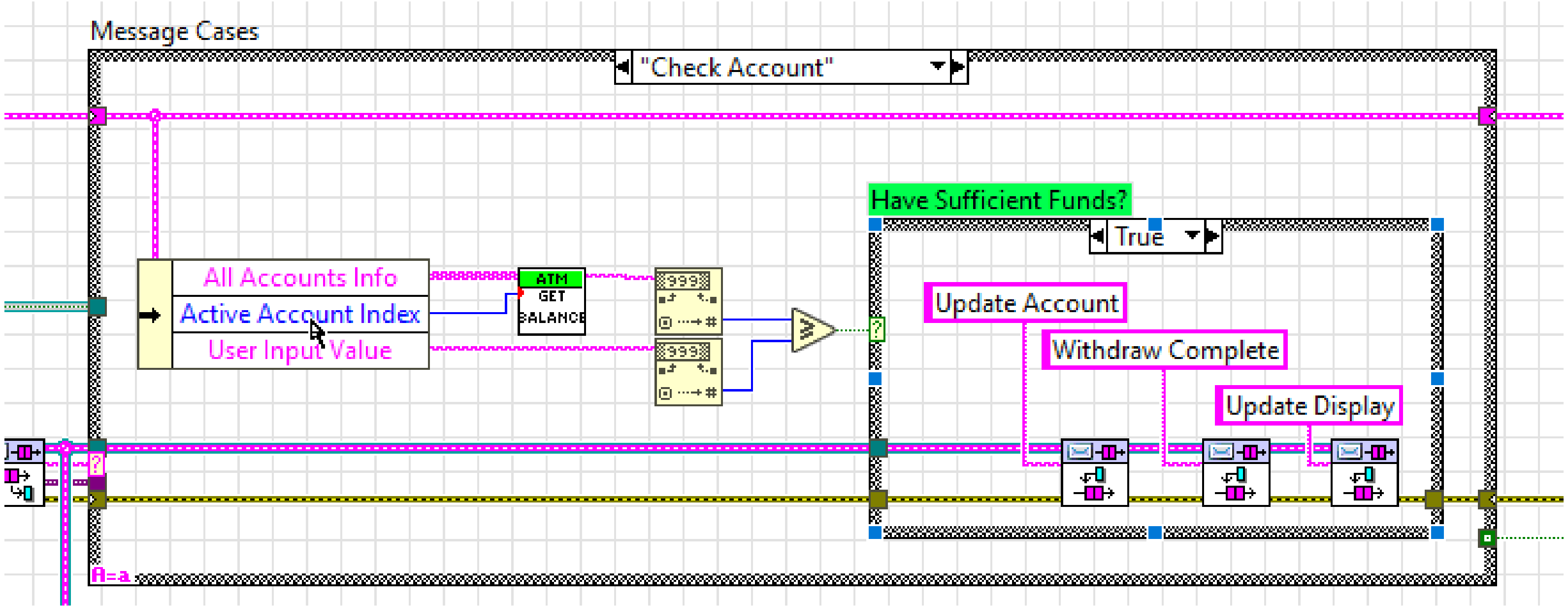
- will suffer bugs
- have difficulty debugging
- implement poor ‘fixes’ or workarounds

Due to no Stateful and Explicit Flow Control

Your Flow Stinks If…

- “Happy Dancing” on a button is bad
- Understanding program operation is not obvious from the code
- Code controlling flow is ‘mushed’ together with ‘do’ code
- Preview Q or Flush Q is used
- You spent no time intentionally developing it

Can you spot the bugs?



Performing a specific action and making choices of what to do next are **categorically different**.

Breaking SOLID & SMoRES

*Single Responsibility Principle

*Modularity (Functional Containment)

Program flow varies based on ‘mode of program’

State nomenclature is a natural self declarative method of defining modal flows

ata’

State: Daddy-ing

Response: Break out the wipes and nose clip

State: Bachelor Party-ing

Response: Push friend out of car near a gas station

How much do you care about state & flow?

- TLB' elevates State and Flow to First Class Citizenship
    - State is a critical component of design
    - Majority of design time is spent thinking about how program should flow

- DQMH minimizes stateful awareness through enabling applications of many simple 'actors'
    - State is (usually) an optional or minimal component of design
    - Majority of design time is spent thinking about the external APIs for your modules

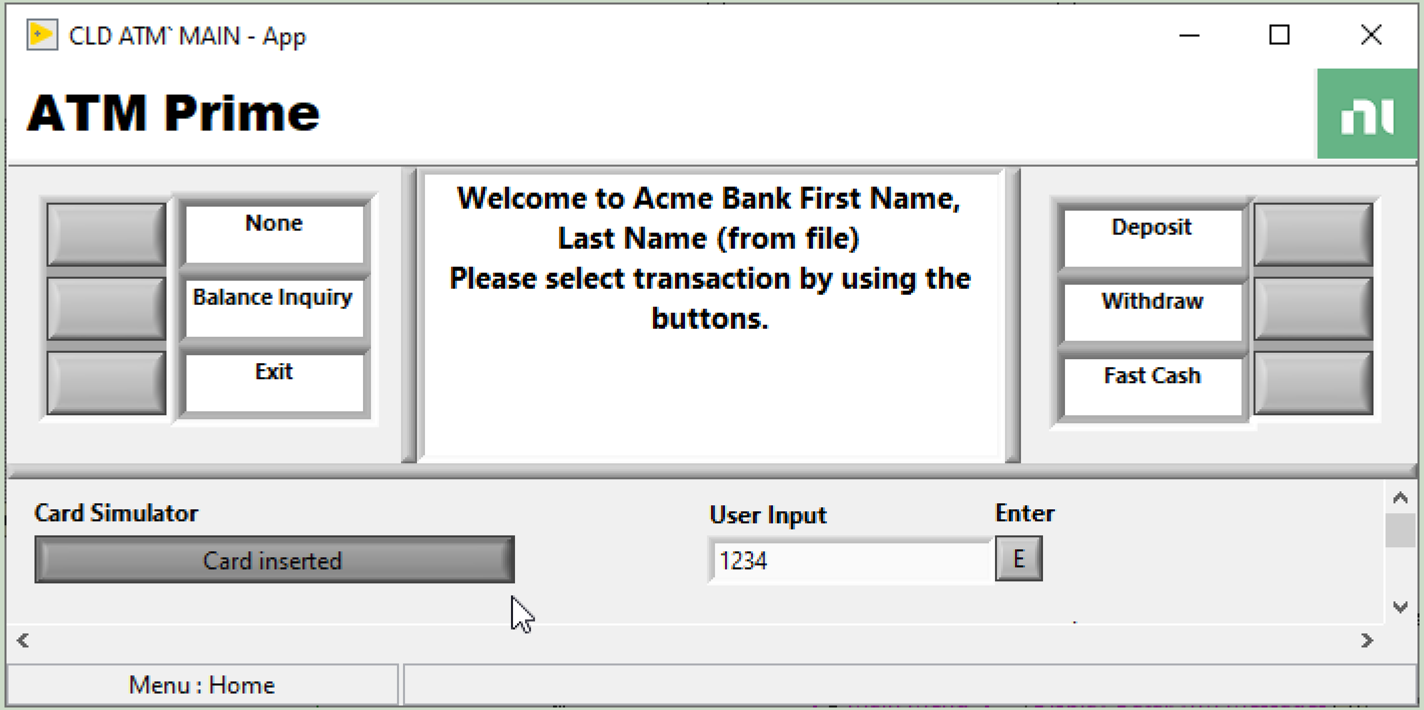
Like it or not, the framework you choose also affects how you think about your application design.

How you naturally decompose a problem will lead to what you like to design with

Norm’s Solution

Demonstrated with a TLB’ implementation of the CLD sample exam – Automated Teller Machine

Norm’s ATM solution

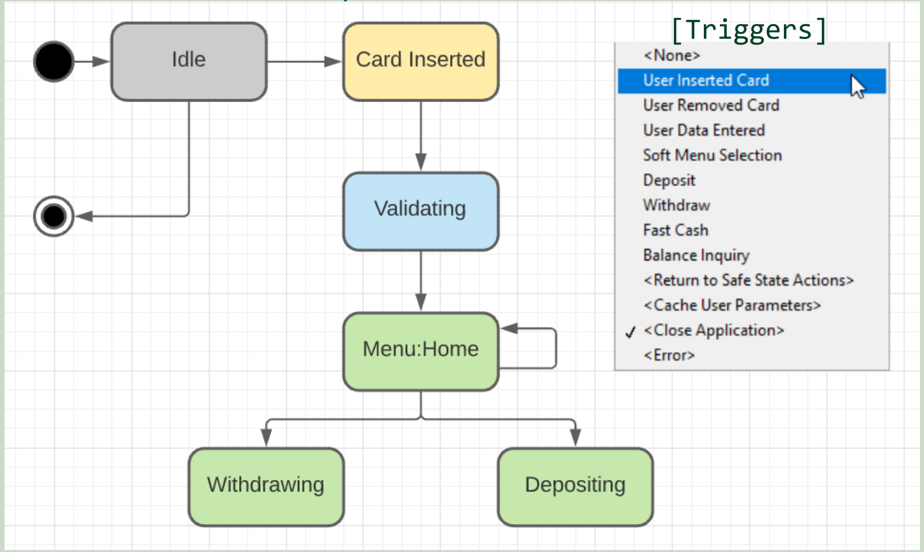

CLD ATM` MAIN - App

ATM Prime

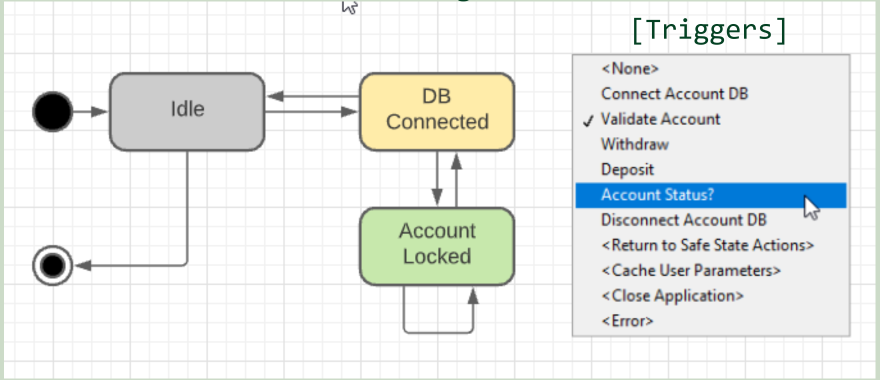
Welcome to Acme Bank First Name, Last Name (from file)
Please select transaction by using the buttons.

Norm’s ATM solution

Top Level ATM Prime



Account Manager Module

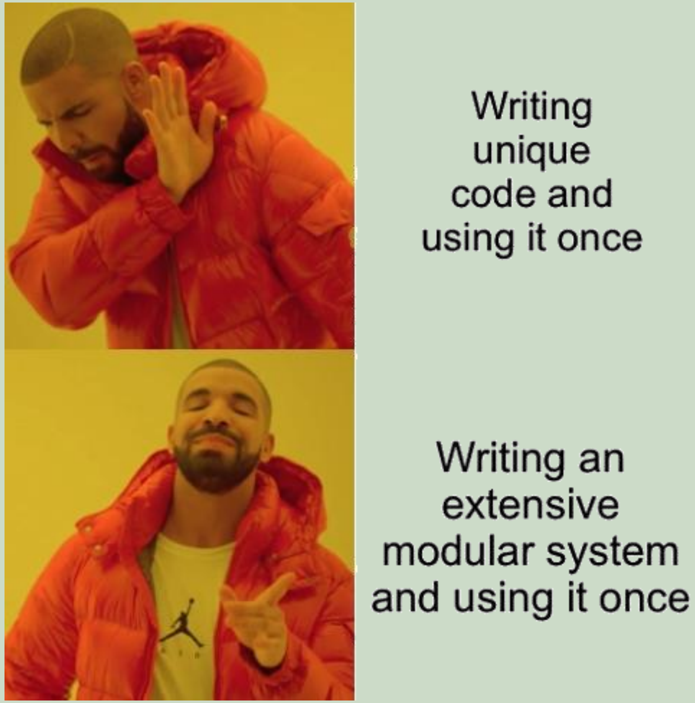


DNatt’s Counterpoint

Demonstrated with a DQMH implementation of the ATM, which has exactly one state variable in the entire codebase

Chewbacca

“What if… ?”


Writing unique code and using it once
Writing an extensive modular system and using it once

DNatt’s ATM solution

Three Modules

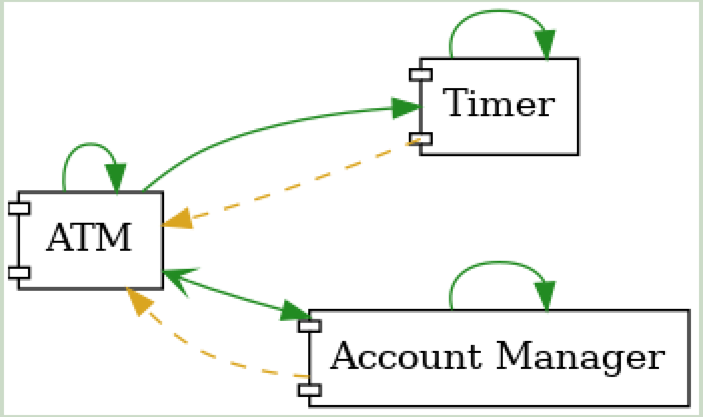


Image created with AntiDoc
vipm.io/package/wovalab_lib_antidoc

State Data

- ATM module
  - Waiting Task (enum)
- Timer module
  - *None*
- Account Manager module
  - *None*

A given DQMH module often has very little (if any) local state.

Norm’s Counter-Counterpoint

Demonstrated with the never-before-seen DQMH’ prime design!

Norm’s DQMH ‘solution’

The Brain at the Heart of DQMH

Implications

- Break the link between request and ‘do’
  - Request Event Caught -> enQ request -> State digests -> Q’s up actions
- Leverage tools to create/mange DQMH Event
  - Such Worth
- How to structure to minimize type casting?
- Maintain QMH familiarity

Conclusion

What have we learned?

Norm’s takeaways

- You might be wasting your time debugging ~Omar Mussa
- If you're catching y
  your QMH doi
  wrong time, yo
  problem
- Recognizing that
  has different mode e
  first step to healing
  - Hi my name is 'Norm and I screwed up my flow

eclaring the mode of your
that you can more smarter
s be gooder

t state awareness and
vulnerable to issues
- ng development (maybe)
- during maintenance/upgrade (definitely)

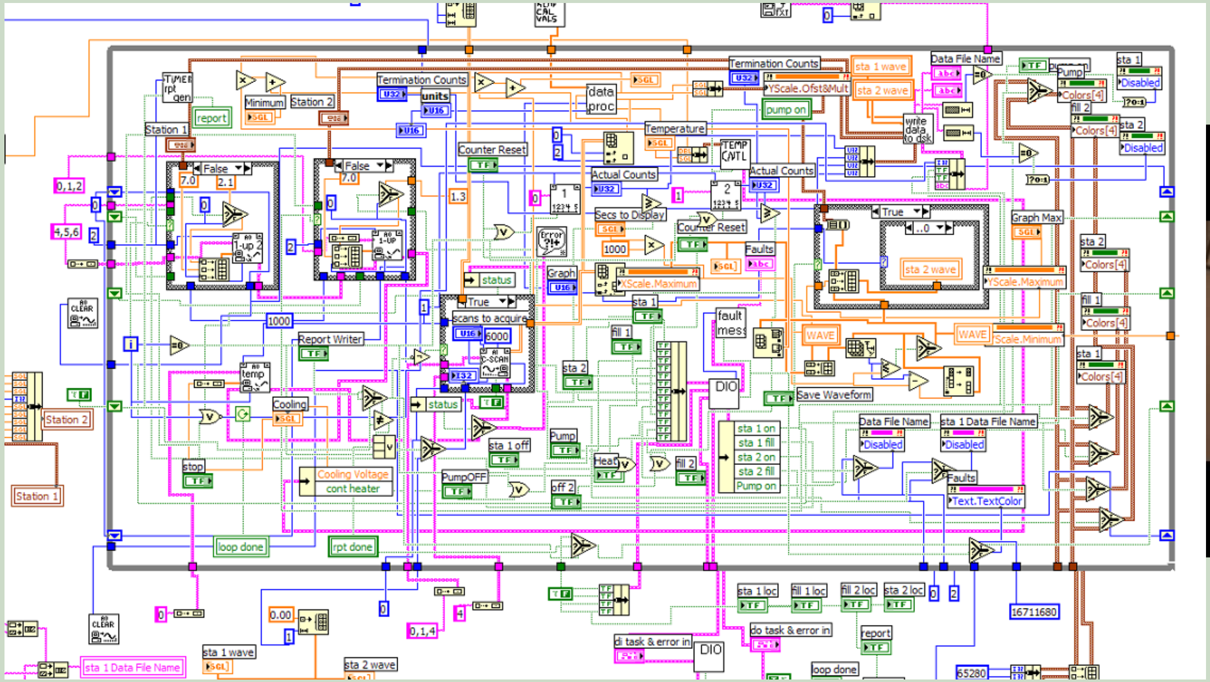
- Even AF is a QMH at the end of the day so it's just as vulnerable
  - And even harder to debug!

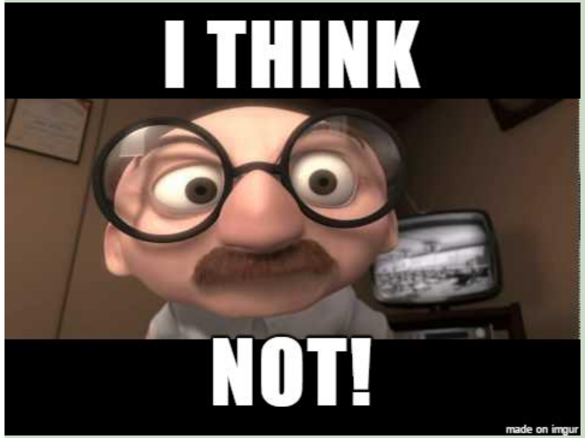
WE NEEDS THIS 4 REALZ

DNatt’s takeaways

- Forget everything Norm just said
- Small modules that can be individually tested make managing trigger response less of a big deal
  - This comes for free with DQMH
- Whitespace is overrated
- Modifying DQMH templates only makes sense if the functionality of the scripting tools is preserved
- In my 7+ years of application development with DQMH, there have been very few times where trigger response/state management was required
  - And when it was, a helper loop with a "simple state machine" tended to do the job

Norm’s counter-takeaways





DNatt Definition of Adequate Whitespace ^^^

Norm’s counter-takeaways

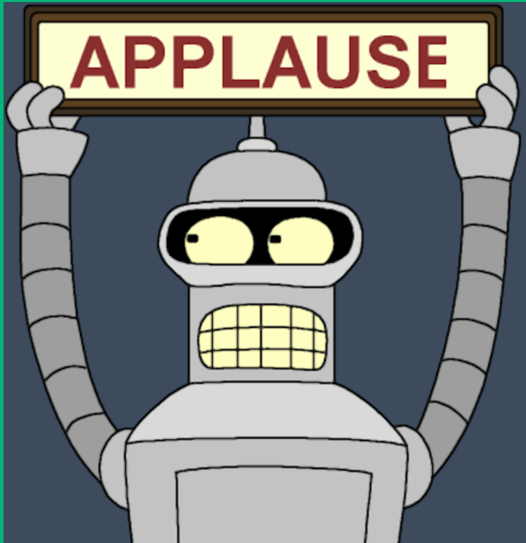
- Why settle for force fitting your flow control just to appease the DQMH gods when you can create a program that gracefully describes and enforces what you want it to do.

- The best of both worlds is possible
  - PrimeTest (MBalla / DPress)
  - HampelSoft (Joerg Hampel)
  - DQMH Prime
    - WIP

DNatt’s counter-counter-takeaways

- History of DQMH
- API Testers
- Events vs. Queues
- Built-in Scripting Tools
- VIPM install
- Make fun of whitespace again
- Number of users
- Enum vs. String
- Inaccurate screenshot
- No UML diagram
- Error generation
- Error propagation
- Versioning
- File I/O
- Politically-oriented hashtag
- “State Machine” naming
- Learning paths
- Certification badge
- VI Analyzer
- Queue API
- Use of LV Classes
- Async vs. Sync calls
- VI Server
- Event lifetimes
- Stopping two loops
- New LabVIEW version
- Get Pizza
- Execution Highlighting
- General Error Handler VI
- Right-click plugins
- LV Speak & Spell
- Virtual Machines
- Quick Drop plugins
- Easter Egg
- Spaghetti Code
- Wire Z-plane Order
- Happy Halloween
- Headless VI Server
- This list is nonsense
- G Web Development
- NI QMH
- Actor Framework
- Messenger
- GDevCon NA
- End of list

Thanks for attending!



Presentation Goal

To improve the way we think about and design our QMH-based LabVIEW programs in two ways:

1. Understand Norm's ever-present complaint about traditional QMH program flow and how it might affect your applications
2. Understand the QMH Prime(') design (even if you don't use it)
3. If all else fails…to entertain