ICS

Building Testability into the Software Architecture of Your HMI

Roland Krause Ph.D
VP of Engineering

Medical Device Recalls Q4/2018

Medical device recalls decreased just 1% to 280 – lower than the last three quarters but higher than the three quarters before that. Recalled units increased 449% to just over 161 million – the second highest quarter since at least 2004

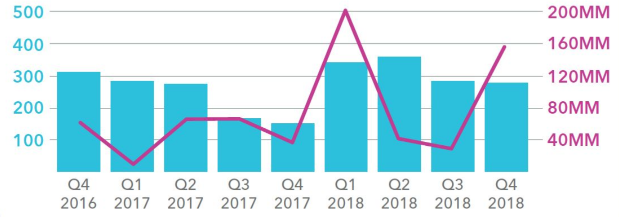
MACHINE MISFORTUNE CONTINUES

At **73.5%**, machine failure was the top cause based on units, mainly due to one large recall. This is the **second consecutive quarter** machine failure was the top cause based on units.

CONNECTING THE DOTS

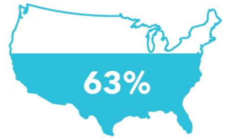
1. The average recall size was 575,449 – the second highest quarter since Q3 2006.
2. At 28.2%, software was the top cause based on recalls for the eleventh consecutive quarter. No other cause was responsible for more than 15.4% of recalls.
3. Five companies reported ten or more recalls in the quarter. This is the highest number since Q3 2016 and the second highest since Q3 2013.

MED DEVICE RECALLS & UNITS



CLASS I UNITS RECALLED PER QUARTER

Q2 2018	599,165
Q3 2018	14,035,195
Q4 2018	1,380,706



OF MEDICAL DEVICE RECALLS WERE NATIONWIDE

TOP MEDICAL DEVICE CAUSES BASED ON RECALLS

Software Issue	Mislabeling Issue	Quality Issue	Sterility
79	43	36	27

More than 160 Million recalled units!

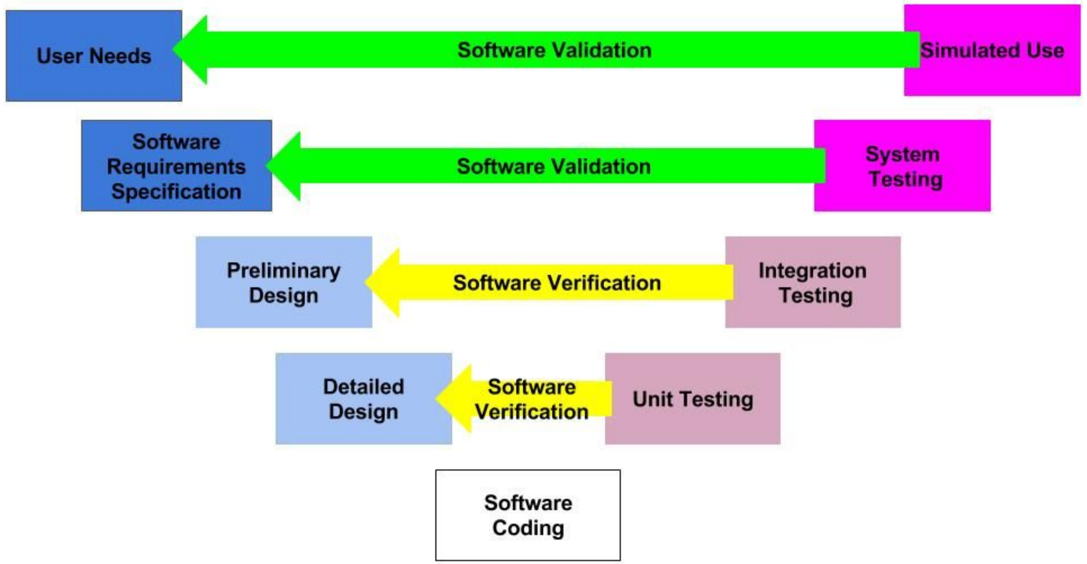
- At 28.2% software was the top cause based on recalls for the eleventh consecutive quarter.
- No other cause was responsible for more than 15.4% of recalls.
- FDA is concerned about software safety since many medical devices now include software.
- Software failure can result in serious injury, or even death to a patient.
  - This places significant liability on the device manufacturer to ensure their software is safe.
- One way to ensure software safety is to perform software verification and validation (V&V).
- But this alone will not ensure software safety.

Safety must be *Incorporated By Design*

Overview

- How to design a robust software architecture taking the requirements of medical device creation into special consideration.
- Validation and Verification requirements addressed by different parts of the process
- Part of these requirements is to introduce testability at the component level, both for system design as well as for individual software components.
- This leads to a layered system architecture where each component fulfills a specific role. The idea is also extended to software.

A quick look at V&V in the SDLC

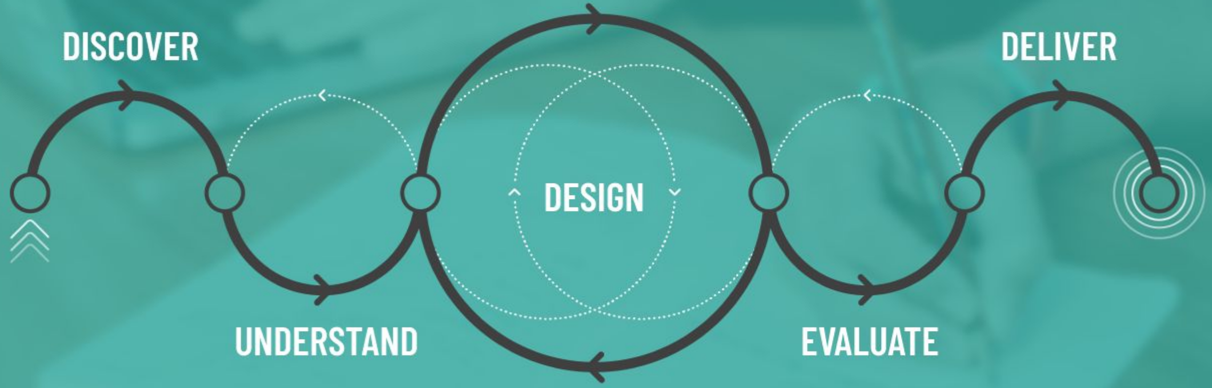


This is actually part of the problem!

PROCESS

Iterative and adaptable design thinking.

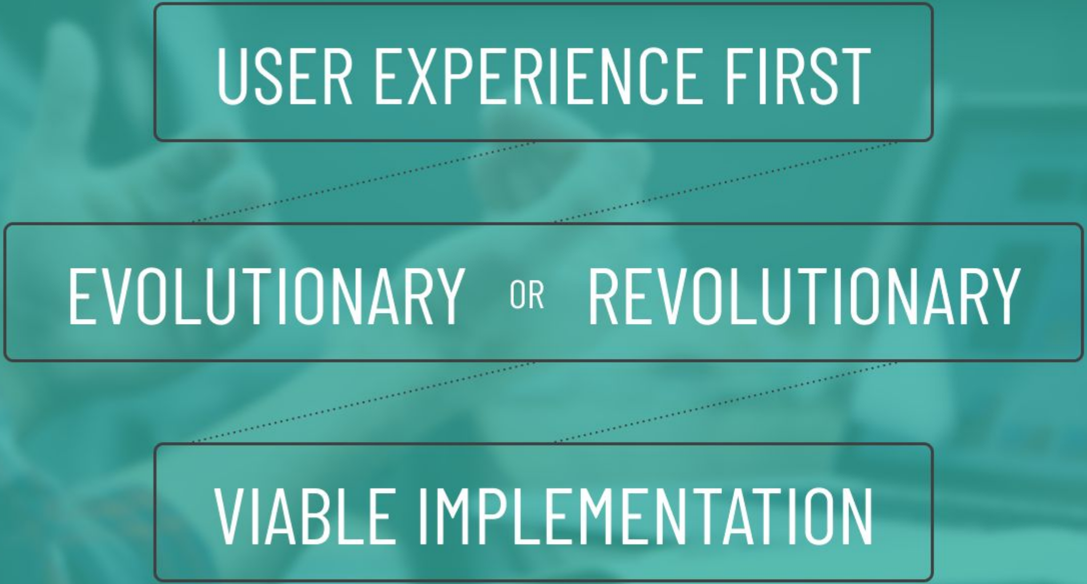
- Improving or creating engaging and powerful experiences on any device
- Activity customization to fit specific project needs
- Keeping clients in the loop every step of the way
- Successfully utilized on a wide range of platforms across multiple industries



PHILOSOPHY

A holistic, user-centered approach combining the practical and the visionary.

- UX requirements are the guiding principles
- Incremental or next generation product development
- The product design must be implemented successfully
- Aligned with client business goals, deadlines, and budget

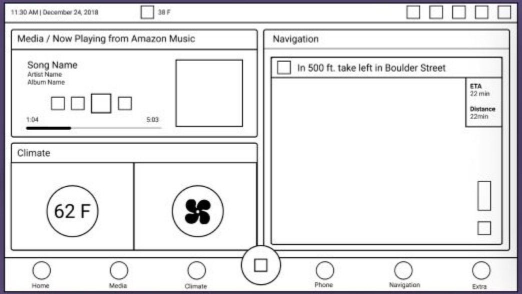


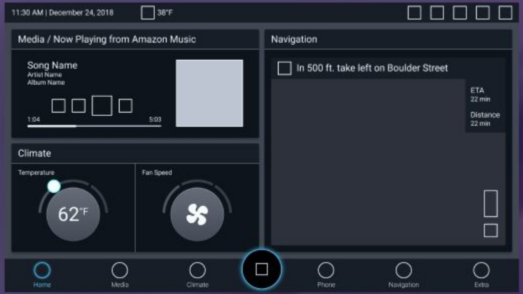
PROCESS

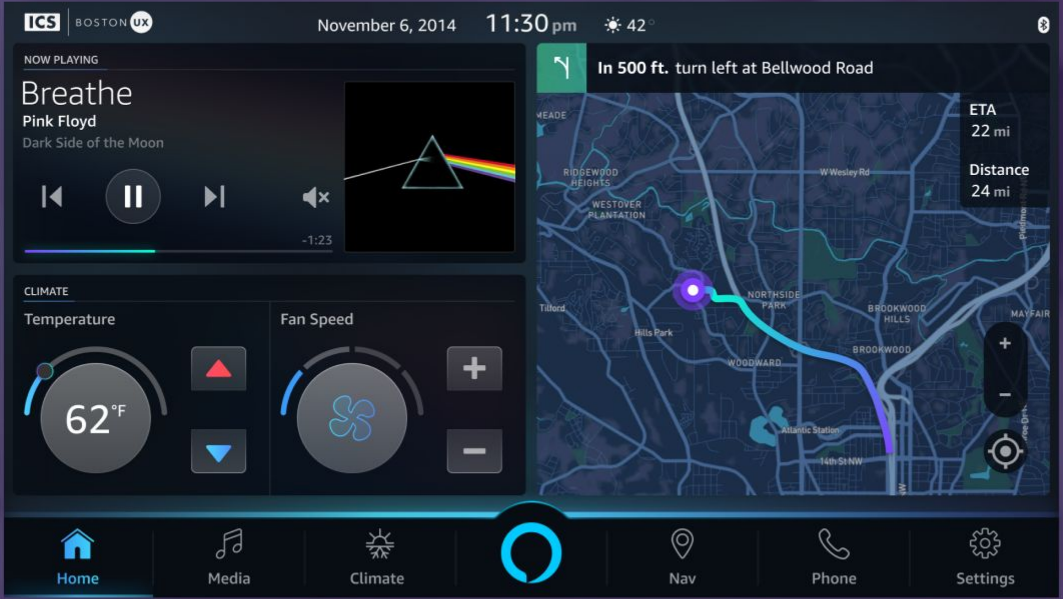
DESIGN

Ideation and generation.

- Sketching
- Whiteboarding
- Map and flows
- Low-fidelity wireframing
- High-fidelity prototyping
- Pixel-perfect visuals







ICS | BOSTON UX

PROCESS

EVALUATE

Testing and examination.

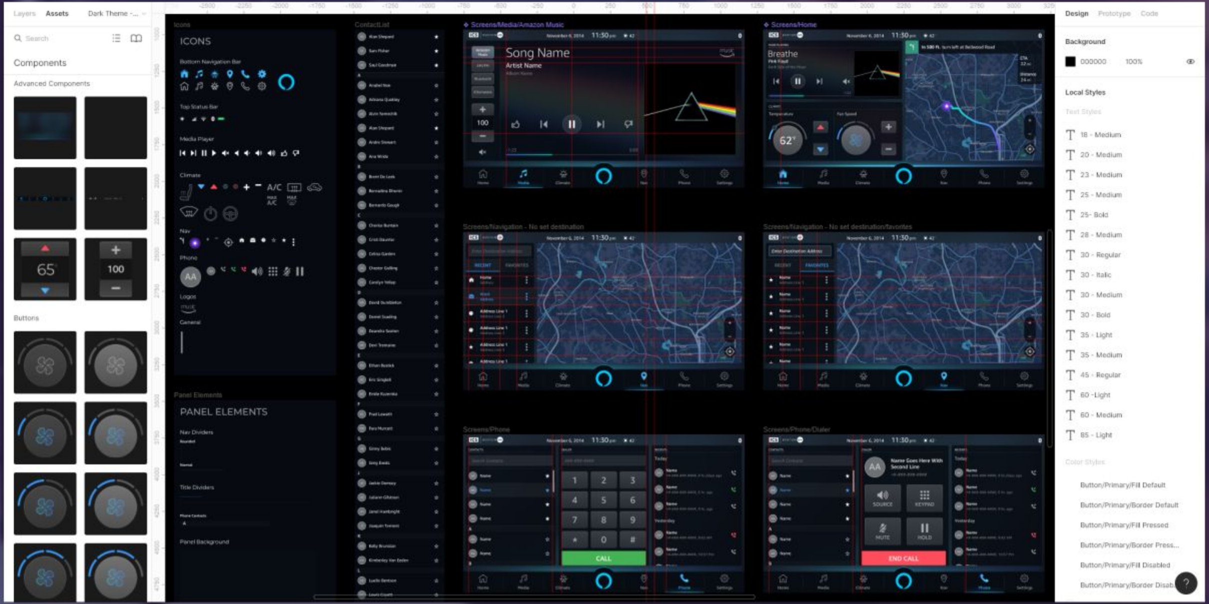
- Usability assessment
- Contextual inquiry
- Heuristic review
- Design recommendations
- Standards and guidelines
- Participatory design

PROCESS

DELIVER

Transfer and actualization.

- Design system
- Visual specifications
- Interaction specifications
- Asset package
- Original artwork
- Implementation support

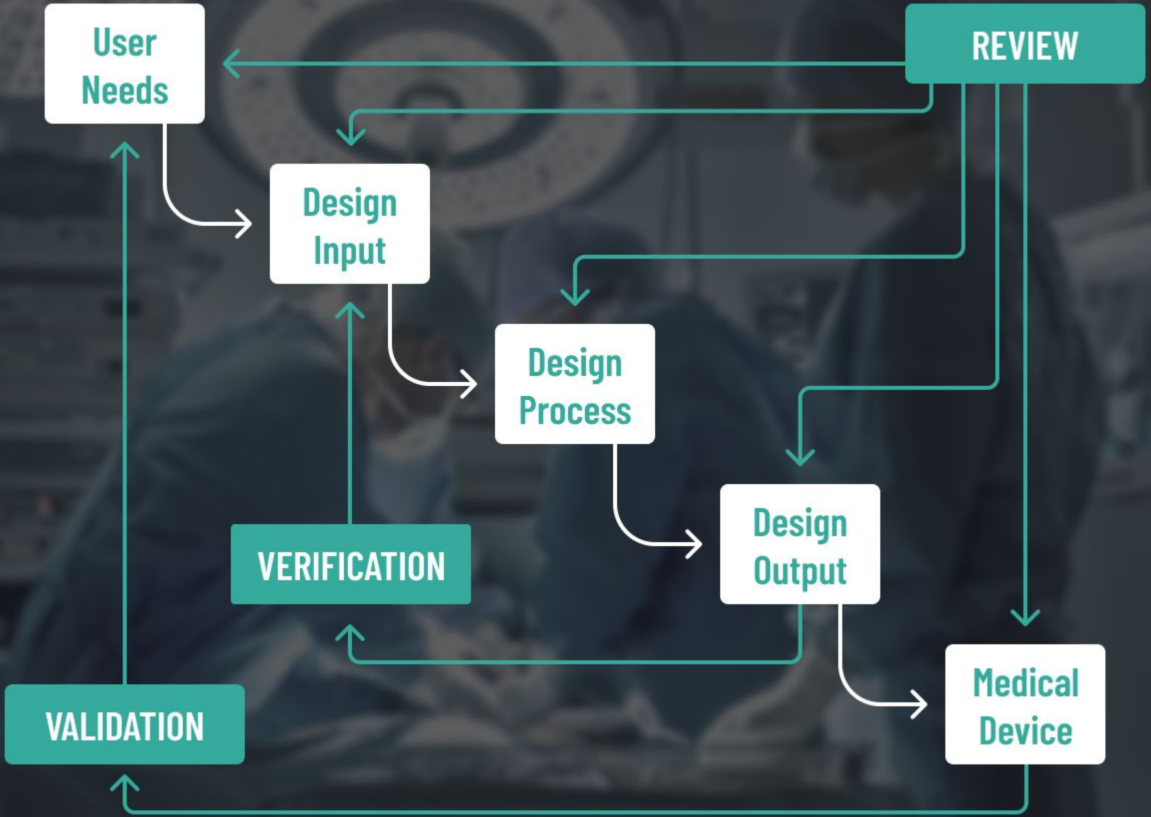


ICS | BOSTON UX

MEDICAL

Supporting medical device development.

- Actionable interpretation of usability objectives
- Blending HE75 guidelines and HCI best practices
- Formative usability testing to uncover potential use errors
- Detailed documentation and design specifications
- Identifying design output to input relationships



ICS | BOSTON UX

Challenge: Create Early Feedback Loops

Let’s take a look at a few definitions:

“Verification means confirmation by examination and provision of objective evidence that specified requirements have been fulfilled.”

“Validation means confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use can be consistently fulfilled.”

In simpler language:

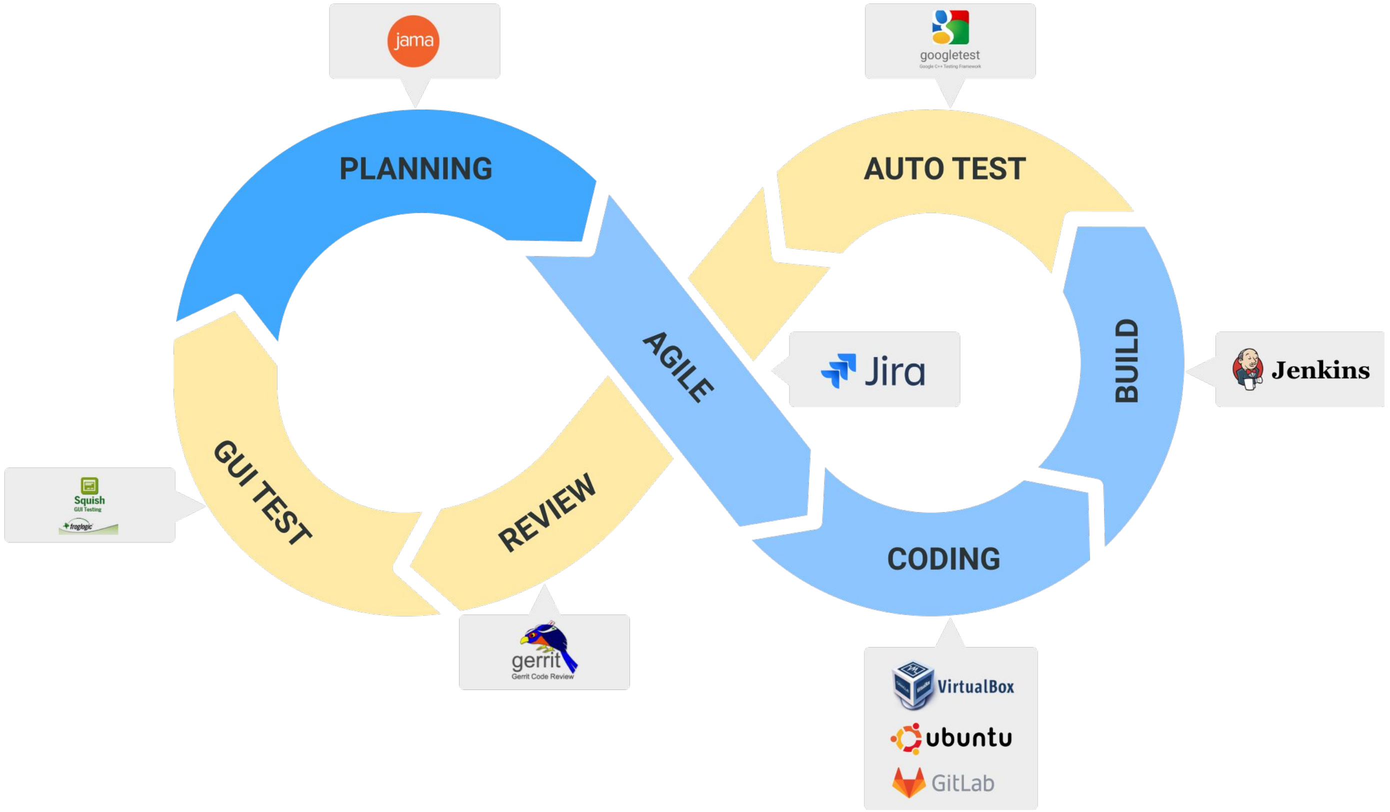
“Validation” ensures user needs are met.

“Verification” ensures that the software is built correctly.

Yet the main challenge here is:

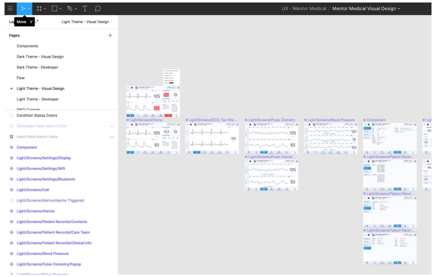
How do you find out early?

ICS DEVELOPMENT OPERATIONS

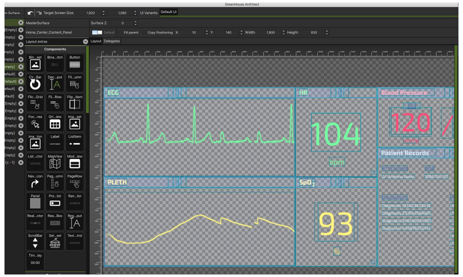


ICS Application Builder: Generating UI Code from UX Design

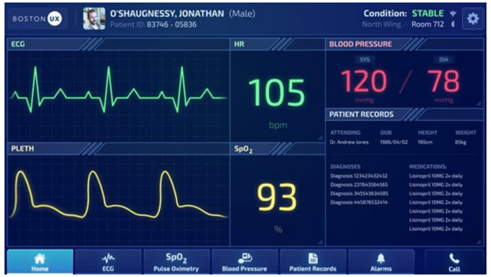
UX Design



IDE and code generation tool



UI of Choice

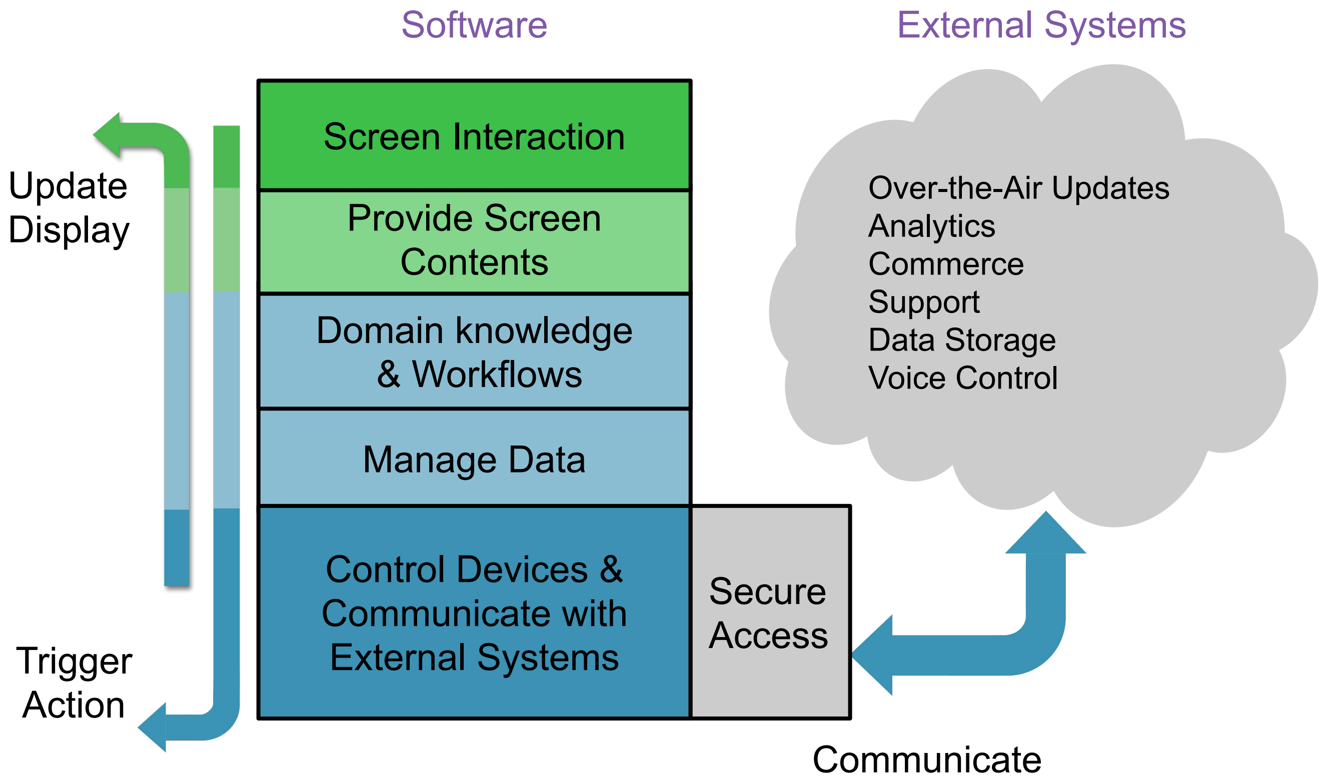


- Reduces time to generate assets
- Easy handoff from designers

- Import assets via design tool
- Layered architecture
- Reusable code

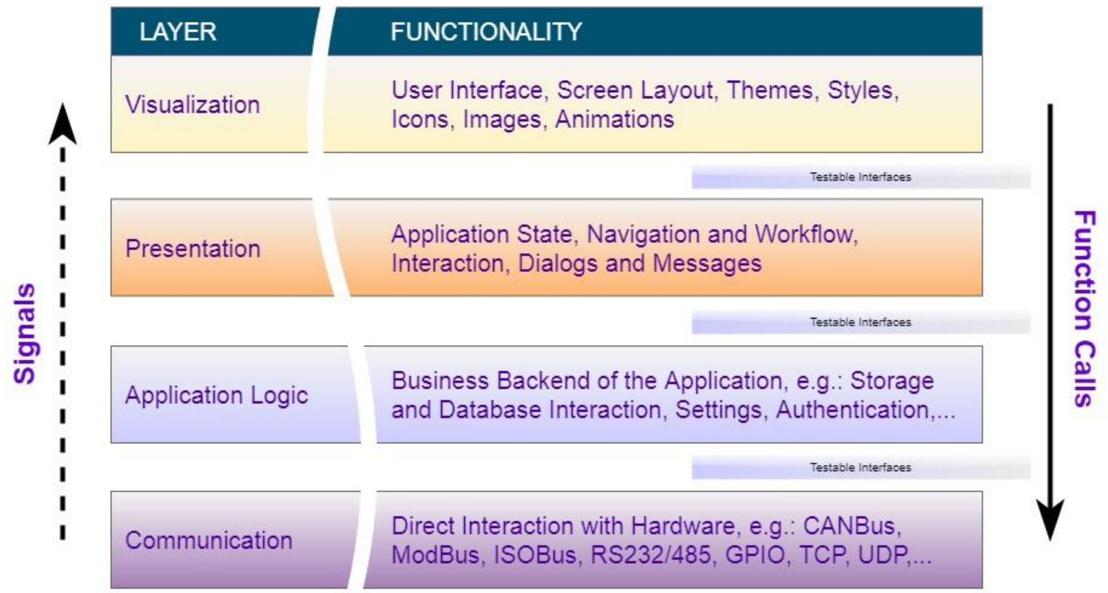
- Generates QML, GL Studio, HTML5
- Agnostic code generation

Technical Approach



Layered Architecture & Reusable Components

- Enable ICS to build better user experiences — faster
- Reusable Components
- Layered framework for building interfaces based on standardized architecture,
- Rapid development tools and reusable code.

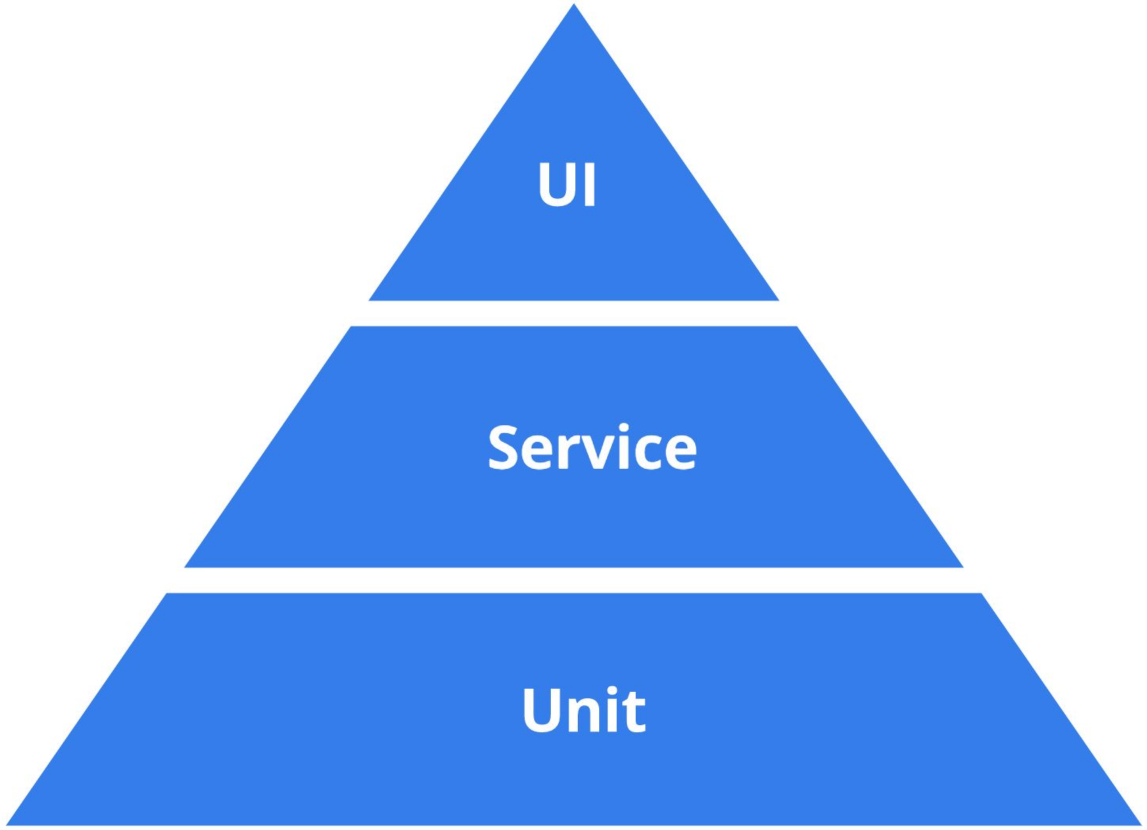


LAYER	FUNCTIONALITY
Visualization	User Interface, Screen Layout, Themes, Styles, Icons, Images, Animations
Presentation	Application State, Navigation and Workflow, Interaction, Dialogs and Messages
Application Logic	Business Backend of the Application, e.g.: Storage and Database Interaction, Settings, Authentication,...
Communication	Direct Interaction with Hardware, e.g.: CANBus, ModBus, ISOBus, RS232/485, GPIO, TCP, UDP,...

GreenHouse

Testing

Testing Pyramid



Test Plan

- Test Driven Development
- Unit Tests (Continuous Integration)
- Integration (multiple Unit)
- Automated UI
- Verification (local or integrated)
  - Test protocols → Executed Test Evidence
- SOUP/COTS Validation
- Other objective evidence
- Static Analysis
  - Coverity, qmllint, etc
- Dynamic Analysis
  - Valgrind, Profiling, Benchmarking

Code and Test Generation in GreenHouse

- The Qt/QML GreenHouse code generator produces a set Google Test based unit tests that aim to ensure that
  - the GreenHouse runtime and generated backend interfaces can be regression tested and have at least 75% test line coverage.
- The generated unit tests include:
  - A set of unit tests for the GreenHouse runtime library (91% line coverage)
  - A set of unit tests for the standard AppCore components such as plugin management etc. (97% line coverage)
  - A set of unit tests for the generated C++ (~100% line coverage for backend interfaces and a 67% coverage for the state machine factory)

Code and Test Generation in GreenHouse

The generated C++ code unit tests cover aspects such as:

- Making sure all of the backend interface property setters and getters produce consistent and expected results

- All methods emit the signals declared using the Tooling interface

- We are still working to improve the test coverage of the generated state machine,
- To get complete coverage one will need a working UI thus would need to use additional tools such us Squish.
- In real life project the formal testing strategy includes running the unit tests as part of the continuous integration process

Conclusion

- Building Testability into the Software Architecture of Your HMI
- When designing software for medical devices it is most important that
  - the overall system can be validated and
  - the implementation can be verified.
- Components of software components must also be individually testable.
  - Layered architecture model allows testing of the software as a whole and at the individual layer/component level.
- Our presentation gives an example on how to include testability into the software architecture from the beginning which ensures stable and safe software, avoiding expensive rework.

Thank You!