WHITE PAPER | JANUARY 2021

Best Practices for Developing Software for Regulated Industries

The Market Leader in Mechatronics and Detailed Engineering Design Services | simplexitypd.com

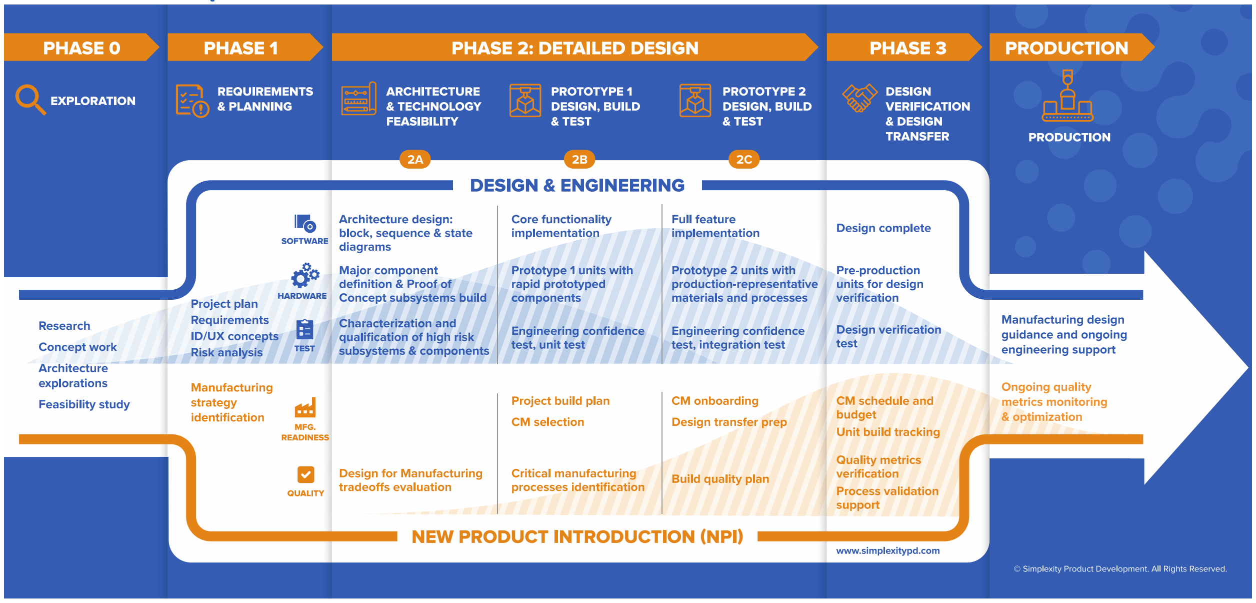
INTRODUCTION

This document reviews best practices for developing software for use in regulated industries. Specific development standards and regulatory requirements exist for different industries. IEC 62304, for instance, is a software development standard specific to medical devices containing software. This paper will approach best practices based on the additional regulatory requirements required by IEC 62304.

Because of the inherent complexity of software systems, software development has become more tightly controlled than hardware in regulatory environments. Thus, standards have emerged to guide this process specifically for software. For medical devises containing software, the software must be developed according to IEC 62304.

The processes put forth by IEC 62304 are established best practices in the software industry, most of which can be followed for both medical and non-medical projects. The standard does not prescribe a particular life cycle model or documentation structure, only that the activities and tasks be completed. The activities described in 62304 fit neatly into many phase-based product development approaches. For the purpose of this white paper, we will reference a phase-based product development approach summarized in the graphics below.

Product Development Process

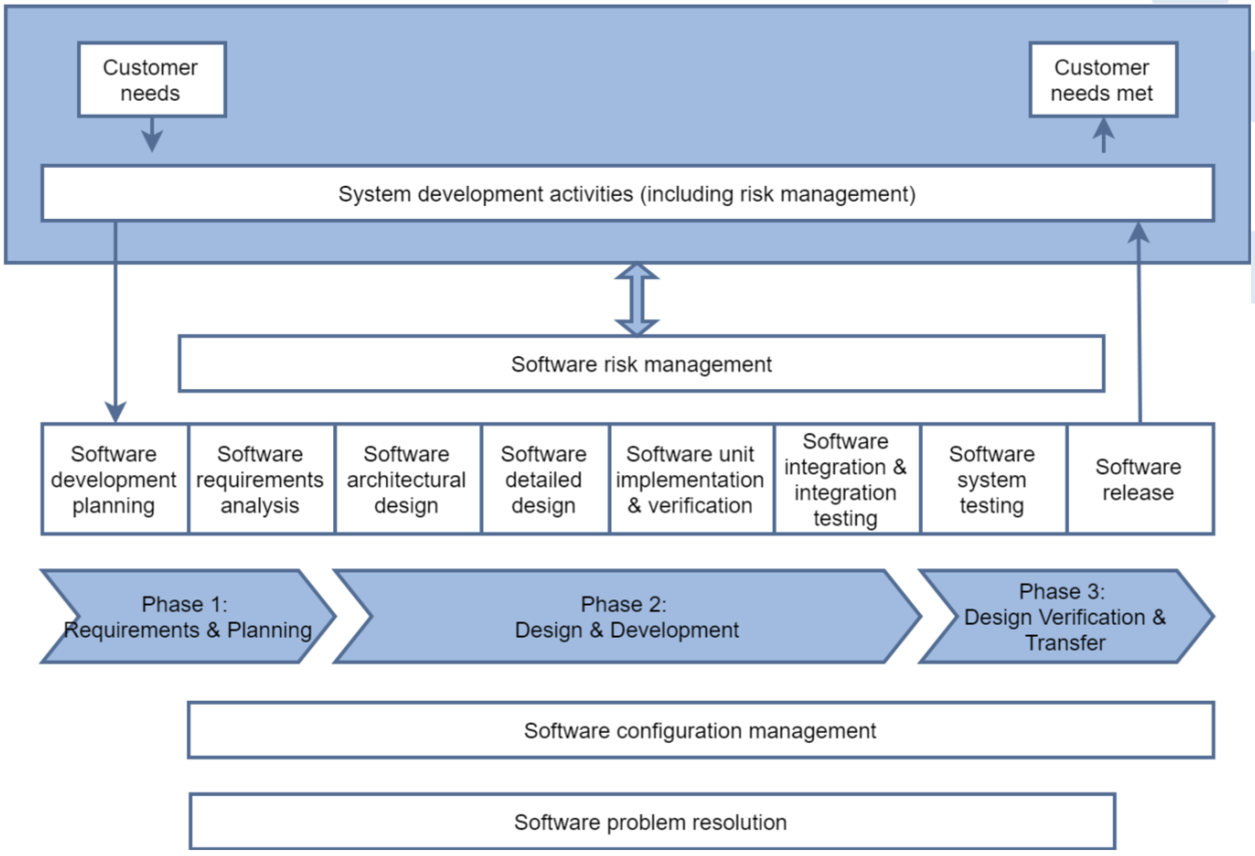

PHASE 0
EXPLORATION
PHASE 1
REQUIREMENTS & PLANNING
PHASE 2: DETAILED DESIGN
ARCHITECTURE & TECHNOLOGY FEASIBILITY
PROTOTYPE 1 DESIGN, BUILD & TEST
PROTOTYPE 2 DESIGN, BUILD & TEST
PHASE 3
DESIGN VERIFICATION & DESIGN TRANSFER
PRODUCTION
PRODUCTION
2A
2B
2C
DESIGN & ENGINEERING
SOFTWARE
HARDWARE
TEST
MFG. READINESS
QUALITY
Research
Concept work
Architecture explorations
Feasibility study
Project plan
Requirements
ID/UX concepts
Risk analysis
Manufacturing strategy identification
Architecture design: block, sequence & state diagrams
Major component definition & Proof of Concept subsystems build
Characterization and qualification of high risk subsystems & components
Design for Manufacturing tradeoffs evaluation
Core functionality implementation
Prototype 1 units with rapid prototyped components
Engineering confidence test, unit test
Project build plan
CM selection
Critical manufacturing processes identification
Full feature implementation
Prototype 2 units with production-representative materials and processes
Engineering confidence test, integration test
CM onboarding
Design transfer prep
Build quality plan
Design complete
Pre-production units for design verification
Design verification test
CM schedule and budget
Unit build tracking
Quality metrics verification
Process validation support
Manufacturing design guidance and ongoing engineering support
Ongoing quality metrics monitoring & optimization
NEW PRODUCT INTRODUCTION (NPI)
www.simplexitypd.com
© Simplexity Product Development. All Rights Reserved.

Software Development Process

Here is a more detailed view of this development approach as relates specifically to software. Software risk management, configuration management, and problem resolution activities span nearly the entire

Simplexity® PRODUCT DEVELOPMENT

development process. Software architecture, detailed design, unit implementation and testing, and integration and testing are software activities that happen during the Phase 2 Detailed Design phase.


Customer needs
Customer needs met
System development activities (including risk management)
Software risk management
Software development planning
Software requirements analysis
Software architectural design
Software detailed design
Software unit implementation & verification
Software integration & integration testing
Software system testing
Software release
Phase 1: Requirements & Planning
Phase 2: Design & Development
Phase 3: Design Verification & Transfer
Software configuration management
Software problem resolution

PHASE 1 - REQUIREMENTS AND PLANNING

The software development plan is a living document that guides the development process for the entirety of the project. It outlines the software tasks to be performed and which roles are responsible for those tasks. In consulting, where the responsibilities could fall to the client or to the client's engineering design partner, it is especially important to clarify ownership of tasks. The software life cycle model must be decided at this point. Will the project follow Agile? Waterfall? Or a middle ground, incremental approach? Some engineering design firms offer processes that are more flexible than others so establishing this early is key to ensure the design partner's processes are flexible enough to respond to client needs.

The plan identifies software milestones and outlines which features will be delivered by which milestone. One common milestone is to ensure there is enough software available to test the hardware when the first prototype is built. The software development plan will be updated in every phase of development, and by the end of the project it must match the reality of what was actually done.

The software requirements specification is derived from the product requirements, which are also developed during Phase 1. The software requirements include the functional and capability requirements, software system inputs and outputs, interfaces, security, and usability requirements. In addition, any risk control measures determined from the risk analysis must be added as requirements.

PHASE 2A - ARCHITECTURE AND FEASIBILITY

The architecture activity in software is where the software requirements are mapped into an architectural explanation of the software's structure. Typically, this includes a high-level block diagram showing how the software interacts with the hardware, as well as system-level state and sequence diagrams to describe the behavior of the software system.

Relating back to the medical device standard, in the language of IEC 62304, software is described with three terms:

- **Software System** – the top level, composed of one or more software items
- **Software Item** – any identifiable part of a computer program. It is composed of one or more software "units".
- **Software Unit** – the lowest level that is not further decomposed (the level of decomposition is determined by the project team. It is NOT required to have each class or function be a "unit")

It is important for the architecture to identify software items, as they will be analyzed during the risk management process.

The architecture must also describe interfaces between the software and internal or external components such as the interface between two software items, two processors, or the processor and a mobile application.

During this phase, an initial software risk analysis would be run and a preliminary software safety classification determined.

PHASE 2B/2C - DETAILED DESIGN

Detailed design typically has multiple iterative sub-phases (2B, 2C, etc.) in which three software activities must be completed:

1. ***Detailed design*** fleshes out the details outlined in the architecture. Each of the items identified in the architecture is divided into software units and described in enough detail to allow implementation to proceed.
2. ***Unit implementation and unit testing*** is the part that software engineers love – the actual coding! Unit implementation converts the abstract designs into concrete source code and then runnable binaries available for test. The implementation of a unit is not complete until it has been unit tested and the code has been reviewed by the team.
3. ***Software integration and integration testing*** is where developers integrate software units into progressively larger subsystems and test them.

During the detailed design phases, it can be challenging to align software and hardware development. Project phases typically follow the hardware build schedule, which has a longer design-build-test cycle

Simplexity® PRODUCT DEVELOPMENT

than software. Generally, most features are complete in the first hardware prototype build. In contrast, each software feature goes through a design-build-test cycle separately. Software development, therefore, has no need for an artificial boundary between the hardware builds. At the same time, the hardware must be tested using functional software or firmware. One effective strategy is to determine what software features are necessary to verify the hardware. Completion of those features can serve as milestones (outlined in the Software Development Plan) for early design phases. By the end of the final design-build-test phase the software should be feature complete.

In addition, it is not necessary to complete detailed design of the entire system before implementing particular modules, so an incremental model is an acceptable approach. Design reviews, another best practice required under ISO 13485 and IEC 62304, would be completed at appropriate increments or milestones as portions of the design are completed.

As much as possible, it is best to automate unit testing for projects. This facilitates a streamlined weekly release process that can run static analysis, create the output binaries, and execute unit tests.

PHASE 3 - DESIGN VERIFICATION AND TRANSFER TO MANUFACTURING

In Phase 3, the team's focus turns away from development and towards verification. The software system verification test protocols were released in Phase 2, typically written by the quality engineer with input from the software engineers. The team executes dry runs and fixes final issues with the software or the protocols themselves.

Before formal verification, the software team prepares a *candidate* release build, including release notes which contain lists of known issues, added features, and defects fixed in this version. This software build is the version that will be used during formal software verification testing (SVT). Though this version is often casually referred to as "released software", it is not officially released until it has passed SVT.

During formal verification, it is important that the tester NOT be any of the engineers involved in writing the software. Any issues or anomalies found during testing during this phase will be documented in the issue tracking system to be evaluated by the project team.

Formal verification of the software happens along with the overall design verification testing for the product as a whole.

CONCLUSION

The formality of documentation may differ, but the best practices of planning, documenting requirements, performing design and code reviews and different levels of testing are best practices that should apply to all projects.

There are a few critical aspects of software development that we haven't discussed in detail. Risk management, software safety classification, configuration management, issue tracking and problem resolution are also important pieces of the software development process, but those will be saved for another publication.

ABOUT THE AUTHOR

Katie Elliott is an Expert Firmware Engineer at Simplexity Product Development. She holds a BS and MS in Electrical Engineering from the University of Washington and is a US patent holder with over 20 years of new product and embedded systems development experience. Her expertise includes embedded software, digital signal processing, real-time systems, wearable devices, and developing firmware for medical devices. She also helped lead our Quality team through ISO 13485 certification.