INSIGHTS

into **Robotics Software Development**

MEDACUITY

MANAGING LARGE ROBOTICS SOFTWARE TEAMS

6 Critical Lessons Learned (the hard way)

Developing a robot is more complicated than a standard embedded software implementation. There are unique challenges to overcome to ensure that a robot will meet its requirements, function as intended, and be delivered on time and on budget.

The teams developing robotics software must coordinate well within their own team while also working in parallel with development teams from many other engineering disciplines. Building a robot requires integrated efforts of software engineers, system engineers, electrical engineers, control engineers, and mechanical engineers. During development cycles, test robots are a limited resource because of their cost and complexity. All engineering disciplines have to share the few robots that are available. Most importantly, robots are fragile and dangerous, so safety is the top priority.

Robots consist of many types of processors, actuators, and sensors. Therefore, robotic software engineers must constantly learn new APIs, tools, and techniques to work with a wide-ranging set of components.

If the robot is for a medical application, you will also need to address the added layer of regulatory considerations. For example, all software for a medical application must be tested to a level that eliminates risk to the patients and staff working with the robot. Emerging and evolving safety and performance standards coupled with regulatory requirements can create a huge burden on software engineering teams developing a medical robot.

Here, we share six critical lessons learned to help directors of software engineers and their managers effectively manage the myriad of challenges inherent in robotic software development projects while keeping software engineers safe and avoid blowing up their robots.

1. Communication & Coordination are Vital

When collaborating with a large multidisciplinary team of engineers, effective communication is critical, especially if the team is distributed.

The daily scrum meeting is a best practice for sharing information across engineering disciplines. Software engineering teams expect this standard practice, but scrum meetings may be new for the system, electrical, and mechanical engineers on the project. As a software manager, consider inviting engineers from other disciplines. They do not need to come every day, but if you extend the invitation, you provide team members from the extended team a forum where they can go when they need something from the software team.

Here's a closer look at the various engineering disciplines, their areas of responsibility, and how they interact:

System engineering is responsible for defining the system requirements and distributing them to the other engineering disciplines. They typically represent the product owner for end-of-sprint demos. The System Test Lead is responsible for verifying the robot works and for distinguishing software bugs from hardware bugs. We have found that when the System Engineering Lead and the Lead System Test Engineer regularly attend stand-ups, there is synergy in sharing the system test burden with the software test team. The System Engineering Lead typically owns the robot hardware and is key to ensuring that the robot is configured for testing the software.

Control engineering can be tightly coupled with the software team if you are using a framework, such as Robot Operating System (ROS). If you are using code generation from Simulink or other high-level design tools to create your controls software, then you will need software engineers to integrate the control system code with the other components in the system.

Mechanical engineering typically owns the product lifecycle management (PLM) tool and works with the software team to plan part numbers for software images stored on the robot. Software managers should work with mechanical engineering early on to establish a process for loading finished software to the system. Otherwise, bringing up your robot product line at the factory could become chaotic with staff trying to find the correct software version to install, especially with last minute bug fixes.

Electrical engineering owns the boards that your software runs on. To verify that the hardware and firmware are working on a new board and to ensure a smooth board bring-up, assign some of your software engineers to start working with the electrical engineers early in the development cycle. The software engineer can influence the schematic by recommending features that will assist in software tests, for example, including LEDs and exposing test points that can be probed with a scope or logic analyzer. Boards that are easy to debug and test can improve your ability to meet your time-to-market window and avoid costly rework.

2. Robots are Scarce Resources

The days of everyone having a robot on their desk are few and far between. With many teams operating in remote work environments, some smaller robots can go home with people, but this is still rare. Surgical robot engineering units typically cost about $500k and require a forklift and lift truck to move.

Large software teams often only have a few robots available for development work. Even with a large distributed development team, it is possible to successfully accommodate everyone with few complaints. Here's how we ensured that large software teams have access to the robots when needed.

The Lead Software Test Engineer has a dedicated robot during core business hours. Anyone who wants access to the robot coordinates directly with them. The lead software test engineer works with software engineers to test their builds between his tests, which is helpful for distributed teams because all communication can be done over email and instant message.

The System Test Lead also has a dedicated robot and coordinates with the software team. They focus on addressing customer-facing bug fixes and dedicate a portion of their time and robot to work on issues discussed at the morning stand-up meetings.

The Software team shares two full robots that have most capabilities intact and two robot "road-kills." A road-kill consists of the same processors, sensors, and actuators of the full robot wired together, but the actuators or motors are not physically connected to the arms or chassis. When the motor spins, nothing physically moves, which helps to keep the operator safe.

The team has a group calendar where they reserve time slots during the week. Typically, this approach covers 80% of the scheduling conflicts. The other 20% of conflicts are handled at the daily stand-up. Effective scheduling coordination is key to ensuring that "not having access to a robot" is never a blocker for completing the assigned work.

3. Robots are Fragile

All electronic systems can be fragile, including robots. For example, an expensive engineering board that is not properly installed can blow up if you insert it backwards. Early robotics prototypes can be especially fragile since all safeguards may not be implemented in the prototype. Prototypes are critical to testing hardware and software, and solving technical challenges in the development cycle. Engineers need to work with these prototypes carefully. A momentary act of carelessness not only destroys the hardware, but also impedes the whole software team who need to use that hardware.

Robots can be mechanically fragile too. Joints have limits and can be overextended and break. Additionally, unless they are programmed, robots are not aware of the relative position of appendages and can collide with themselves. It is very easy to exceed a joint limit or forget to remove add-on items from a previous test that are not in the current collision model before testing the automated movement of the robot.

Software engineers should "know their robot" before starting work with the physical robot. This is done through different types of training. Training begins with a software engineer reading the robot operator's manual, if you are fortunate enough to work with a team that created one. If you are not that fortunate, it's worth the time to create an operator's manual with the systems engineer so that your team will understand the capabilities and limitations of the robot. Consider adding a requirement or training event to the backlog for the software engineer to read the operator's manual before any hands-on work with the robot.

The next training step is having a seasoned member of the team, typically the Lead Software Test Engineer, introduce the new engineer to the robot and demonstrate its typical operation. This includes installing the battery, powering the robot up, and connecting to the robot from the controller. The lead engineer should show the new engineer where the emergency stop is, discuss which operations on the controller are safe to use, and demonstrate how to shut down the robot, and how to disconnect the battery.

The emergency stop is key. The new engineer must be ready to press that button when testing new code or performing any operation that causes robot movement. Accidents do happen. However, with proper training, any negative impact on the robot and the people in the test lab can be greatly reduced or prevented altogether.

4. Robots are Dangerous

A robot with untested software can move in unpredictable ways. If velocity limits are not controlled, robots can easily accelerate and hurt or kill someone. A robot that can operate safely next to a person is known as a collaborative robot (cobot). A surgical robot needs to operate around surgical staff and must be able to collide with people safely. Software engineers might work with engineering units that do not have functional collision sensors. In this state, robots do not detect people around them, which can be extremely dangerous.

Standoff distance is key to safety. Robots should be considered dangerous until proven otherwise. Plan to have two people in the lab for early tests of robots: one person with their finger on the emergency stop and the other triggering the test operation.

Software managers should also make sure robots are isolated from the corporate network. Robots can easily be connected to the network by accident or for convenience. Once on the network, an engineer who is testing controller software could accidentally connect to the real robot instead of their emulator. Seeing a 500-pound robot suddenly come to life and try to drive off its lift is scary and dangerous. You should isolate all local testing behind containers that cannot connect to corporate networks.

5. Robot Software Development Requires Many Skills

Because software engineers develop the entire robotic software stack they need to be jacks-of-all-trades. The software engineer must also be highly proficient in their primary programming language, whether it's C++, Python, or Simulink and MATLAB. This includes firmware, board support packages (BSPs), embedded OS, device drivers, and application code. Plus, if you use a specialized framework like ROS or ROS2, your team will need to be highly proficient with that software framework and tools.

The key to effectively managing the complexity of a large robot software development project is a software architecture that promotes consistency across the software stack. You must always ask yourself,

> *"Can your software architecture span the processors, sensors, and actuators in your robot?"*

6. Medical Robot Development Requires Even More Consideration

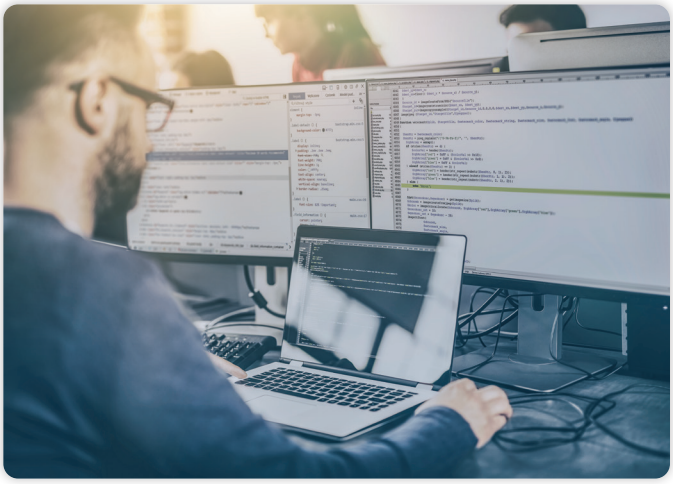
The unique regulatory requirements of medical devices impact the entire robot software development lifecycle. All tools used in the lifecycle need to be validated. The software, depending on the medical device class of the robot, will need to be documented in a rigorously managed quality management system (QMS). For example, ROS and ROS2 are considered software of unknown provenance (SOUP), requiring their own validation for the robot's intended use. Medical device standards exist, as well as standards related to surgical robots. Software engineering managers must be fully apprised of these standards when planning their project.

Many engineers working with medical robotics have robotics expertise but lack the knowledge for how to navigate the regulatory development process. In this case, it makes sense to team up with experts early to help the team adapt their development process to efficiently address the regulatory requirements of medical robot development. This is much better than trying to bolt on the regulatory requirements while rushing a product to market.

Lessons Learned

Robot software development projects require different considerations compared to non-robot software development. Software managers have to contend with the unique challenges of coordinating multidisciplinary teams, sharing scarce resources, handling fragile and dangerous robots, complicated software architectures, and regulatory requirements. Knowledge gaps, shortcuts, and inexperience can compound these challenges.

Robot development still seems like the Wild West since many of these efforts are new and unique. When developing medical robots, like any emerging technology, the biggest risk is you don't know what you don't know. Regardless of whether your robotic software development project is medical or not, follow the six best practices we outline here to avoid making mistakes that could be dangerous, expensive, and delay your project schedule.



ABOUT THE AUTHOR

Tom Amlicke | *Software Architect & Robotics Systems Engineer*

With over twenty years of embedded and application-level development experience, Tom designs and deploys enterprise, embedded, and mobile solutions on Linux/UNIX, Mac, iOS, and Windows platforms using a variety of languages including C++, C#, Python, Java, and JavaScript. Additionally, his expertise includes simulation and model-based design using MATLAB and Simulink to better understand the operating principles of robots and robotic systems under development. As lead software architect for robotics projects, Tom oversees end-to-end development of ROS-based mobile robots and surgical robots.

Let us know where you need help with an ongoing or upcoming robotic software development project.

Contact MedAcuity - info@medacuitysoftware.com

ABOUT MEDACUITY

MedAcuity, a specialized engineering firm, develops custom software solutions to address the most critical product development challenges facing MedTech and Robotics companies and innovators, large and small. With over a decade of experience in software design and development methodologies for heavily regulated compliance-driven industries, our technical capabilities span all levels of software from embedded systems to mobile devices, the cloud, and enterprise technologies.

RBINS01-03.2021

MEDACUITY