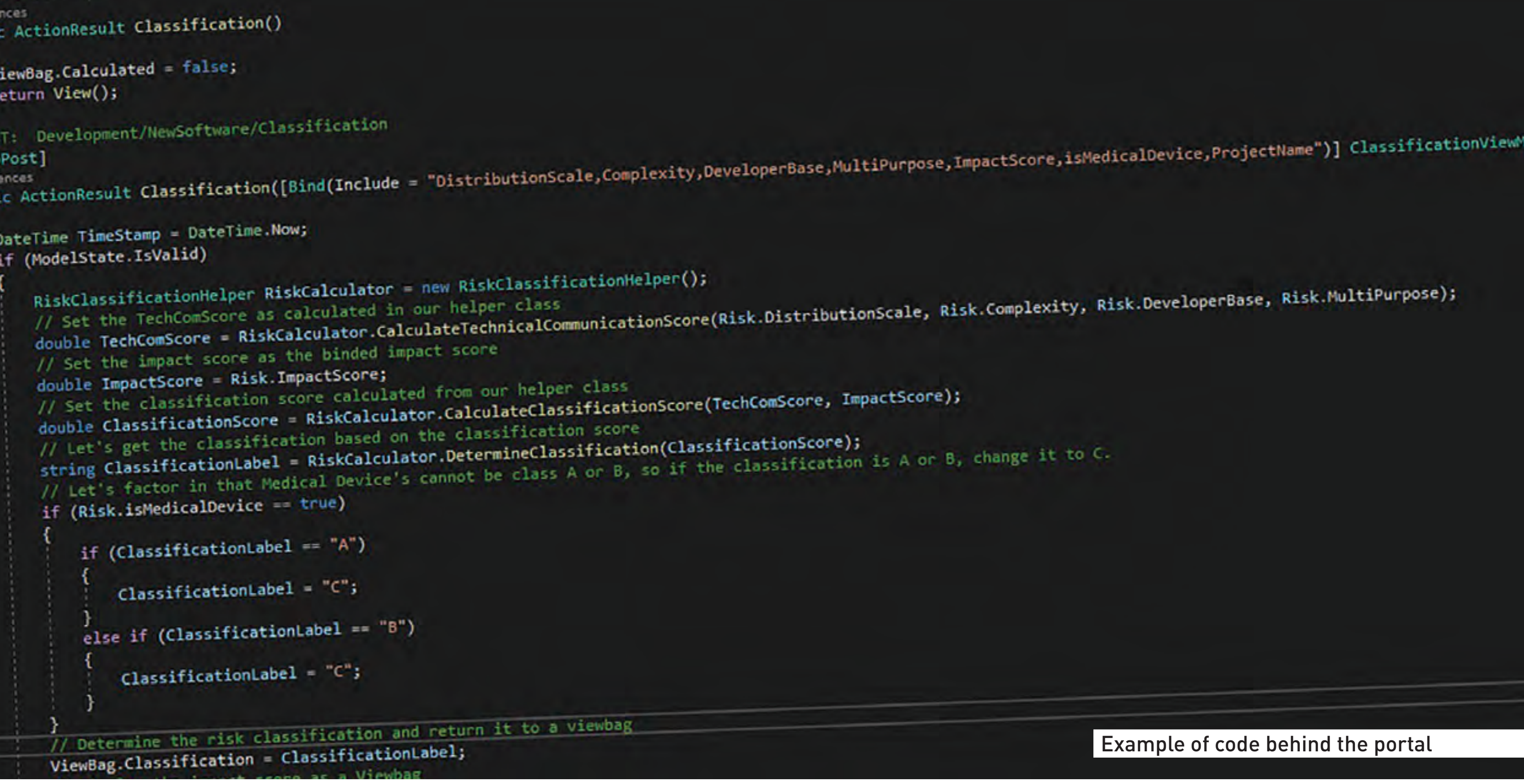


```
nces
c ActionResult Classification()

iewBag.Calculated = false;
eturn View();

T:  Development/NewSoftware/Classification
Post]
ences
c ActionResult Classification([Bind(Include = "DistributionScale,Complexity,DeveloperBase,MultiPurpose,ImpactScore,isMedicalDevice,ProjectName")] ClassificationView

DateTime TimeStamp = DateTime.Now;
if (ModelState.IsValid)
{
    RiskClassificationHelper RiskCalculator = new RiskClassificationHelper();
    // Set the TechComScore as calculated in our helper class
    double TechComScore = RiskCalculator.CalculateTechnicalCommunicationScore(Risk.DistributionScale, Risk.Complexity, Risk.DeveloperBase, Risk.MultiPurpose);
    // Set the impact score as the binded impact score
    double ImpactScore = Risk.ImpactScore;
    // Set the classification score calculated from our helper class
    double ClassificationScore = RiskCalculator.CalculateClassificationScore(TechComScore, ImpactScore);
    // Let's get the classification based on the classification score
    string ClassificationLabel = RiskCalculator.DetermineClassification(ClassificationScore);
    // Let's factor in that Medical Device's cannot be class A or B, so if the classification is A or B, change it to C.
    if (Risk.isMedicalDevice == true)
    {
        if (ClassificationLabel == "A")
        {
            ClassificationLabel = "C";
        }
        else if (ClassificationLabel == "B")
        {
            ClassificationLabel = "C";
        }
    }
    // Determine the risk classification and return it to a viewbag
    ViewBag.Classification = ClassificationLabel;
```

Example of code behind the portal

Winds of change for software regulations

Joe Whitbourn, Ian Boddy, Andrew Simpson, Joshua Kirby, Robert Farley and **Luke Bird,** of James Cook University Hospital, Middlesbrough, examine in what ways medical physics communities will need to prepare for the roll-out of the Medical Device Regulations next year

DEVELOPING AND MAINTAINING in-house software is both important and widespread[1, 2] in medical physics departments.

However, as has been discussed in several previous *Scope* articles,[3, 4] the legislative background to this practice is changing. From May 2020, the Medical Device Regulations (MDR) will be in force which will create important new legal obligations for the in-house development of medical devices including software.

Even in departments where in-house software development doesn't involve medical devices there is now a clear need to ensure that this activity has appropriate governance and assurance arrangements in place.

The process of putting in place design and production controls for higher quality software is a natural fit for a quality management system (QMS). Indeed, the use of a QMS is one of the MDR requirements for the healthcare exemption.[4] The inclusion of software practices in a departmental QMS is therefore both a timely and pressing issue for medical physics departments.

Informal problem

However, a survey[5, 6] of Canadian medical physicists carried out in 2014 found that only a small minority of medical physics departments had formal procedures for the creation and use of in-house software. Our department was an example of a centre that did not have software development and management incorporated into our QMS.

Helpfully, an article in the September 2015 issue of *Scope*[7] provided an approachable overview of what departments should be looking to include in a software QMS. However, we realised that this would be a significant challenge to implement as it would require substantial changes to current working practices.

This article is a summary of how we've met this challenge and bridged across from historic software working practices into improved ones. In particular, we focus on the practical aspects of the implementation.

Audit

Our Head of Medical Physics convened a working group for the project consisting of members from different sections of the medical physics department, to ensure that the outcomes were broadly applicable.

The first active step was to gain a better understanding of the current situation. We therefore audited all software developed (but not necessarily still used) in-house. The software were categorised according to:

- departmental section;
- type of software (e.g. compiled code, databases, functional spreadsheets, etc.);
- user access point, and
- user groups.

Extra detail was also included, such as purpose, programming language and type of outputs. This provided further information for future group discussions in devising a robust process for software practices. The results of the audit demonstrated that our department was:

- inconsistent in how software was accessed and updated;
- overly dependent on spreadsheets for complex applications, and
- using a substantial amount of software whose authors had left the department.

Critically, for the purpose of meeting the requirements of the MDR, this process showed that our department had relatively few pieces of software which would meet the criteria of being medical devices. This encouraged us to ensure that the work we performed wasn't focused solely on medical devices but relevant to all software practices in the department.

As a result of these findings, priorities were set for the next stage of the project; specifically:

- risk management and classification;
- software access and change management, and
- documentation and project management.

Risk management and classification

Different approaches exist to classifying medical device risk status, such as the MDR (Class I, II, etc.) or the IEC 62304 (A, B, C) risk classes. However, these approaches assume that the project in question is a medical device, whereas we wanted an overarching system that would cover all in-house software (except single-use spreadsheets).

We therefore developed an easy-to-use, consistent and effective approach to quantifying the hazard posed by in-house developed software. The first step was to identify relevant risk factors, which were determined to be:

1. User base: the greater the number of users, the greater the risk of an individual using the software in an inappropriate way.
2. Complexity: the proposed continuous development time is used as a universally applicable proxy for complexity. The greater the complexity, the greater the risk of errors occurring, or failing to be detected.
3. Developer base: a large number of developers could infer greater complexity, contributing to the risk identified above. It also increases the potential for failures of communication between individuals and errors occurring as a result.
4. Multi-purpose: software produced for a single, defined purpose is less likely to be used for an inappropriate task.

The risk factors were quantified and the risk score calculated as shown in figure 1 below.

This score is only dependent on the risk factor's hazard as the likelihood is to be assumed as 100 per cent for all potential hazards (in line with the ISO 62304 standard for medical device lifecycle management).

Additionally, the severity of any software malfunction was considered. We chose to adopt the consequence scoring already in use in our NHS Trust. It has five severity levels as in figure 2 below.

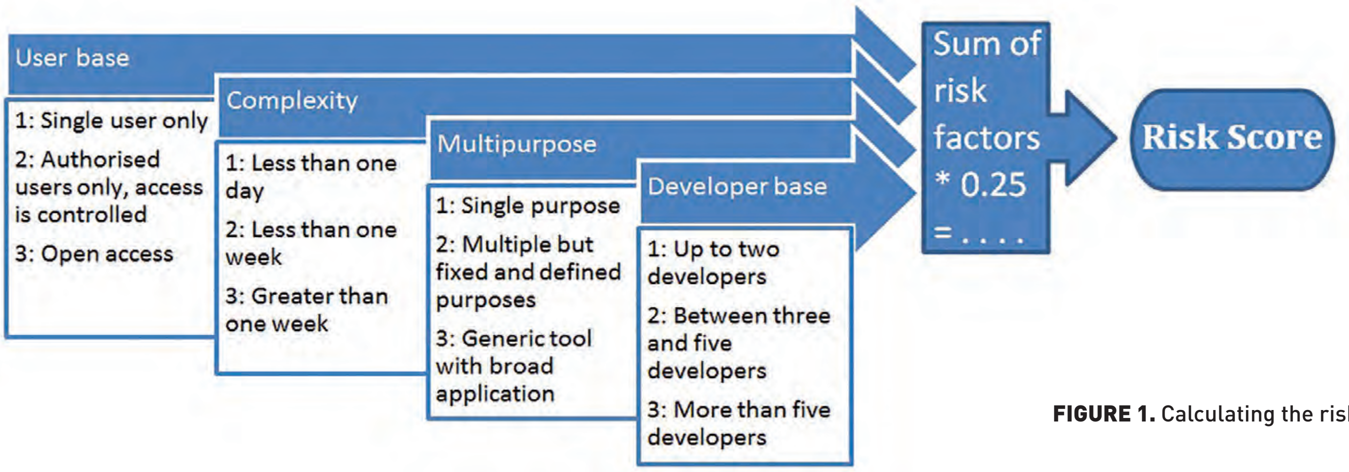

User base
1: Single user only
2: Authorised users only, access is controlled
3: Open access
Complexity
1: Less than one day
2: Less than one week
3: Greater than one week
Multipurpose
1: Single purpose
2: Multiple but fixed and defined purposes
3: Generic tool with broad application
Developer base
1: Up to two developers
2: Between three and five developers
3: More than five developers
Sum of risk factors * 0.25 = ...
Risk Score

FIGURE 1. Calculating the risk score

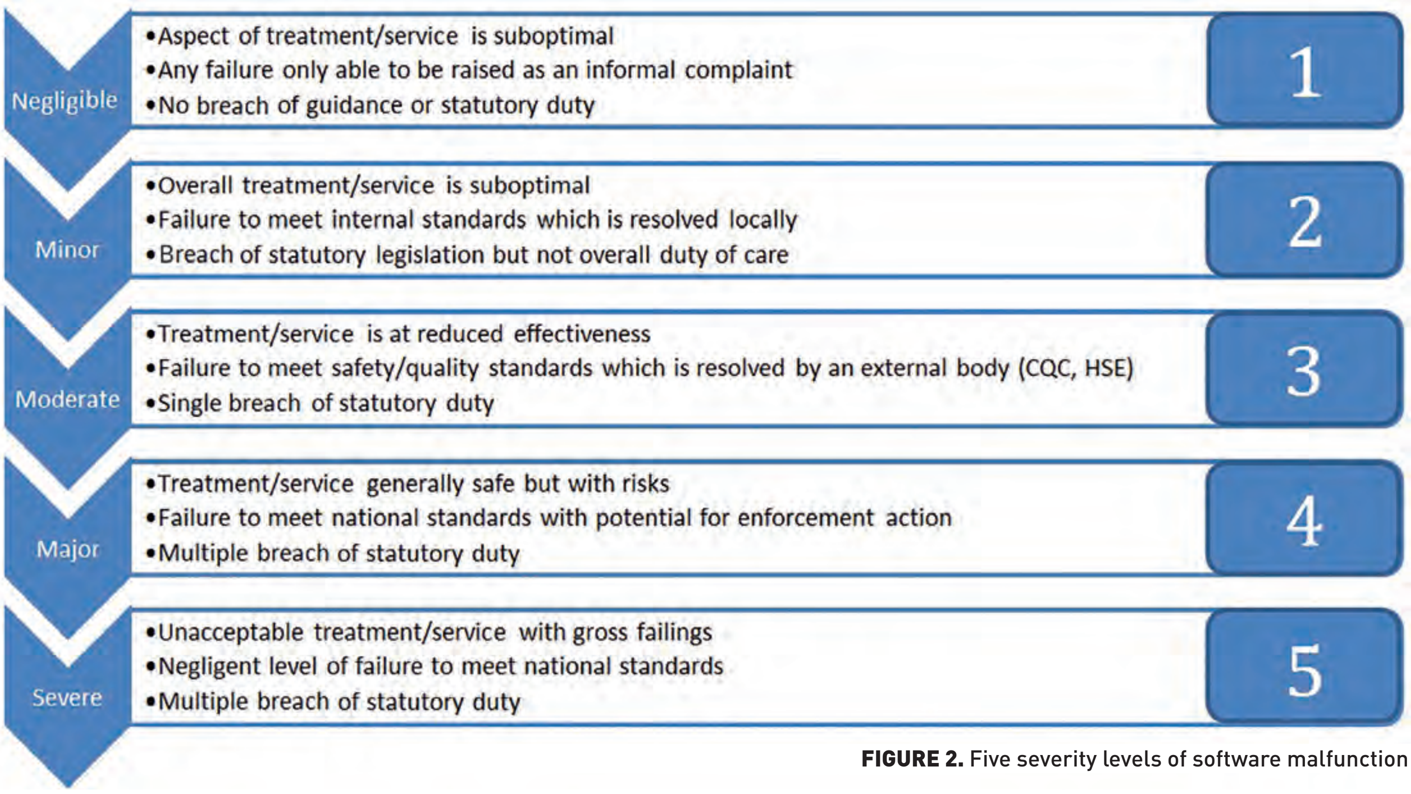
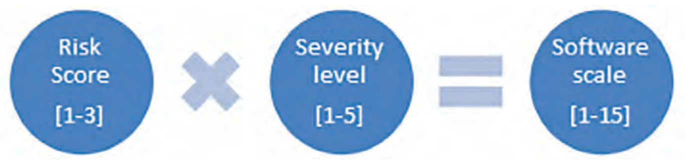
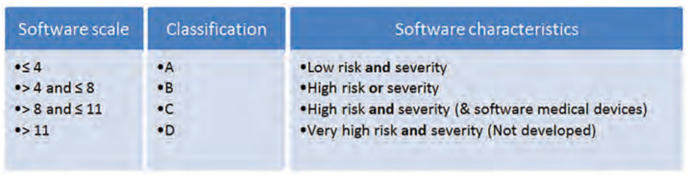

Negligible
•Aspect of treatment/service is suboptimal
•Any failure only able to be raised as an informal complaint
•No breach of guidance or statutory duty
1
Minor
•Overall treatment/service is suboptimal
•Failure to meet internal standards which is resolved locally
•Breach of statutory legislation but not overall duty of care
2
Moderate
•Treatment/service is at reduced effectiveness
•Failure to meet safety/quality standards which is resolved by an external body (CQC, HSE)
•Single breach of statutory duty
3
Major
•Treatment/service generally safe but with risks
•Failure to meet national standards with potential for enforcement action
•Multiple breach of statutory duty
4
Severe
•Unacceptable treatment/service with gross failings
•Negligent level of failure to meet national standards
•Multiple breach of statutory duty
5

FIGURE 2. Five severity levels of software malfunction

The software scale is calculated as follows:



Using this approach, greater weighting is given to severity, acknowledging the potential disruption and harm that a software malfunction could cause. The resulting software scale ranges from 1 to 15 and allows software to be categorised as follows:



Software scale	Classification	Software characteristics
•≤ 4	•A	•Low risk and severity
•> 4 and ≤ 8	•B	•High risk or severity
•> 8 and ≤ 11	•C	•High risk and severity (& software medical devices)
•> 11	•D	•Very high risk and severity (Not developed)

This quantitative approach is broadly applicable to different types of software as well as being quick and straightforward to use. The process was tested on the software identified by the departmental audit and was found to provide appropriate classification of software consistently and rapidly.

The importance of the classification is that it determines the specific activities that are required during each phase of the software lifecycle. Projects with higher software scale scores have more rigorous requirements in terms of software practices. The one exception is in the case of software medical devices, which are automatically assigned a minimum of class C classification.

This all ensures that risk is appropriately managed. One example of the advantages of this approach is that we have decided that the projects involving class D software are beyond the scope of in-house development in our department. Instead, these projects should be directed towards the Trust's innovation team as an unmet need. This ensures that our department's resources are focused on deliverable projects.

Software access and change management

To ensure that there is a consistent process for software access and change management, we created a software portal. The portal features two core elements:

- a software launchpad, and
- a development/project management tool.

The software launchpad quickly allows staff to find approved commercial or in-house developed software, the history of the software and the current approved version of it. The launchpad reduces the use of any incorrect software and thereby reduces the risk of clinical errors caused by using unapproved and outdated software.

The development management tool (figure 3) allows developers to create a new software project, calculate the risk classification, upload documents for approval, assign project members and ultimately obtain project approval from the section heads or head of department. The tool also includes a bug manager, which allows users to submit bugs for resolution. Overall, the key element of this piece of the software is to allow communication and interlink between development and management.

In essence, the goal of the portal is to facilitate the management of software development and deployment in compliance with our in-house procedures, and provide a good user experience whilst doing so.

The portal was developed according to our new software practices and was written in C# (ASP.NET) using the Model-View-Controller (MVC) pattern and Entity Framework. Bootstrap's Cascading Style Sheet (CSS) was used for the look and feel of the portal whilst a Microsoft SQL Server 2008R2 database was used as the backend.

Documentation and project management

One of the priorities of this project was to improve our software development by producing software according to a lifecycle specified by our QMS. The software lifecycle allows us to control and improve our software from the initial stage of proposing an idea to the final implementation and then back round again with any proposed changes to live software. A significant outcome of the project has been a set of documentation, residing within the QMS, to outline the new processes that specify how software is to be developed within the department. The overall process is shown in figure 4. The process also includes the decommissioning of software no longer required for clinical use.

The classification process allows us to develop the software in a manner proportionate to the risk identified. Class A (the lowest risk) software projects have a minimal number of guidelines, such as the need to state the version and intended purpose. Higher classes have additional requirements such as code review and more formal documentation of planning and testing.

Code review is a particular requirement for the more complex projects that we believe will improve the standards of our software by facilitating feedback between developers, such that we are writing more robust, efficient code with styles that become more consistent over time.

Conclusions

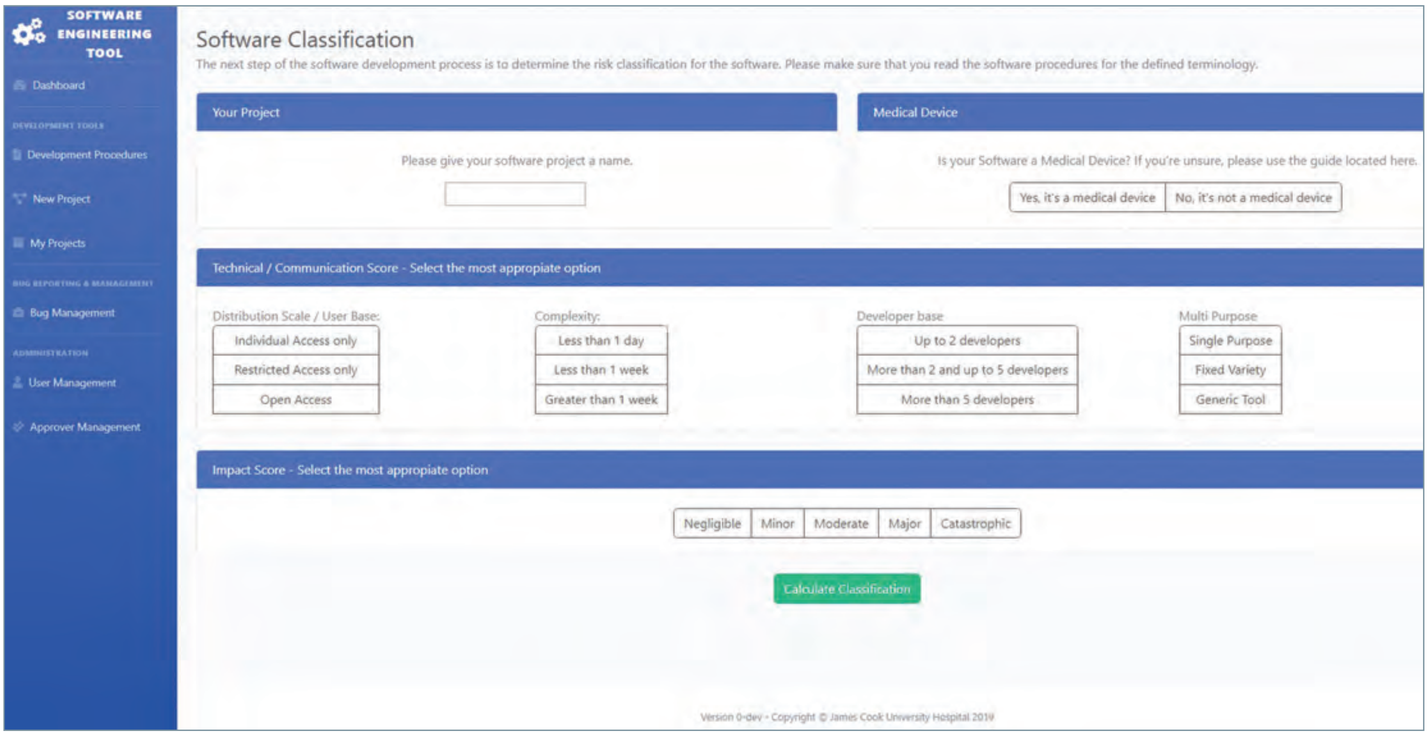
This work has involved people from all areas of the department and took a substantial amount of effort. However, as a result we now have a common framework that has significantly improved software practices.

The system of software classification embeds risk management whilst the graduated classification ensures that best software practices are introduced in a proportionate manner. The creation of a software portal will help us to follow our in-house software development processes. It acts as a 'hub' for all the improved software practices that we initiated whilst making it easier for users to perform their tasks.

The introduction of software project management has created a method for trapping any projects that are likely to take us into areas where significant resources are needed before a large amount of work is undertaken. However, the system has a flexible approach to project implementation details to allow staff to use their professional judgment.

The most important outcome of the work, though, is that we will be compliant with the new MDR which enables our department to provide this same assurance to our NHS Trust.

We hope that this summary is useful to other IPEM members who want to drive up software practice standards in their department. Of course, our system is a version 1.0.0 and will need (managed) change, so we're interested in how other departments are approaching this. Please feel free to join a conversation on the IPEM Informatics and Computing Community of Interest about what else we can do to improve software practices. ■


Software Classification

The next step of the software development process is to determine the risk classification for the software. Please make sure that you read the software procedures for the defined terminology.

Please give your software project a name.

Is your Software a Medical Device? If you're unsure, please use the guide located here.

Version 0-dev - Copyright © James Cook University Hospital 2019

FIGURE 3. Software portal classification

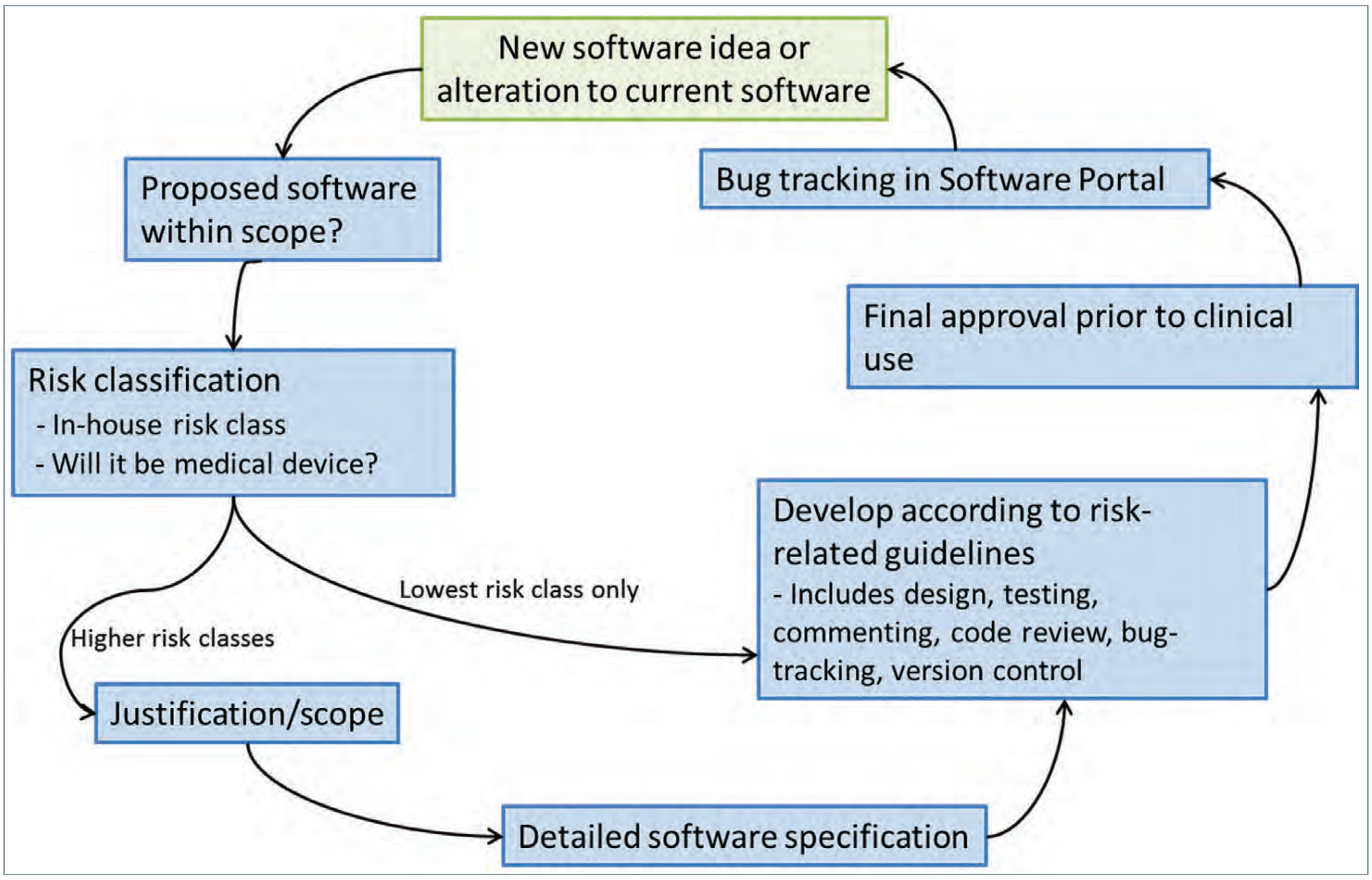

New software idea or alteration to current software
Proposed software within scope?
Risk classification
- In-house risk class
- Will it be medical device?
Higher risk classes
Lowest risk class only
Justification/scope
Detailed software specification
Develop according to risk-related guidelines
- Includes design, testing, commenting, code review, bug-tracking, version control
Final approval prior to clinical use
Bug tracking in Software Portal

FIGURE 4. Overview of the processes created in order to develop software within a software lifecycle defined in the QMS

REFERENCES

1 **Ganney PS.** Dr Ganney's top 10 tips for safer working software. *Scope* 2017; 26(1): 18–20.

2 **IPEM.** *Report 81: Physics Aspects of Quality Control in Radiotherapy, 2nd edition*.

3 **Ganney PS.** MDR: a brief introduction for software. *Scope* 2018; 27(2): 20–23.

4 **McCarthy, J.** MDR: the Health Institution Exemption and MHRA draft guidance. *Scope* 2018; 27(3): 24–7.

5 **Lula UI.** Software quality assurance of in-house software: a survey of Canadian medical physicists. *Scope* 2015; 24(2): 7–9.

6 **Salomons GJ, Kelly D.** A survey of Canadian medical physicists: software quality assurance of in house software. *J Appl Clin Med Phys* 2015; 16: 336–48.

7 **Robert, Ross.** Software quality management: I know how to program!. *Scope* 2015; 24(3): 14-19

CLINICAL SCIENTIST
JOE WHITBOURN
Joe works in radiotherapy having previously gained a PhD at Durham University in astronomy

Email any comments on this article to joe.whitbourn@nhs.net