Key findings and recommendations
IT

Software Development

Summary of findings

- STUDIO is a proprietary DRAGON software solution created and maintained internally by the DRAGON software development team of c.7 FTE. STUDIO is used as a core business workflow tool, to support business processes for prospect management, integrated CRM, online quotation tool and a lifecycle contract management suite. In addition, STUDIO supports customer identification, management of client contact and appointments, price quotation building based on required office phone hardware and services required, scheduling and management of customer installations and post installation support and service provision
- STUDIO is not subject to any long term product roadmap or development plan, instead Management address development requests as they are raised, either as new feature or bug requests. Requests are considered and prioritised on a resource required basis, with activity of typically less that one week being planned as BAU, and those of more than a week of resource requiring business justification and Senior Management approval. There are currently two long term initiatives being considered by Management:
  - Transition of the STUDIO tools from a legacy Delphi Windows Application architecture to PHP web based tools
  - The adoption, and integration with, PowerBI for reporting needs
- STUDIO was originally built as a Windows application(s) developed in Delphi, but Management have now established a plan to transition into a web-based application with the coding developed in PHP. The backend architecture is based on a Microsoft SQLServer database and this is used to manage key aspects of DRAGON business processes. STUDIO is used only by DRAGON staff
- Delphi is no longer widely supported and availability of developers and resources are becoming limited. The transition of the STUDIO tools from Windows applications to PHP web based tools therefore appears appropriate, allowing migration on a module by module basis when redevelopment of tools is required, and supporting the move to more common and readily available skill sets of PHP from the less common Delphi technology stack. Management appear to have established effective planning workflow to enable development decisions taken as either part of BAU work or requiring business justification for larger resource needs in excess of a week of effort
- We have completed a full review of the technology, code and development of STUDIO with summary data provided against each area. Overall, the development methodology and approach being used by DRAGON appears reasonable and appropriate for the development of internal applications, with some potential to improve in the areas of automated testing and documentation to be considered

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Overview		• STUDIO is a proprietary DRAGON software solution created and maintained internally by the DRAGON software development team • STUDIO was originally built as a Windows application developed in Delphi, but Management have now established a plan to transition into a web-based application with the coding developed in PHP. The backend architecture is SQLServer database based, and it is used to manage key aspects of the DRAGON business processes • STUDIO is used as a core business workflow tool, to support business processes for prospect management and a lifecycle contract management, management of client contact and appointments, price quotation building based on required office phone hardware and services required, scheduling and management of customer installations and post installation support and service provision • We understand that the STUDIO system is hosted at TNS and serves c.600 internal users. There is no external customer access or use of the STUDIO suite	• DRAGON have elected to develop a full end to end workflow tool to support their full range of business operations • We have completed a full review of the technology, code and development of STUDIO (see Appendix T and U for details of our methodology and assessment findings) with summary data provided in the following sections	• See further in this section

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
STUDIO Management		• STUDIO is not subject to any long term product roadmap or development plan, instead Management address development requests as they are raised, either as new feature or bug requests. Requests are considered and prioritised on a resource required basis, with activity of typically less that one week being planned as BAU, and those of more than a week of resource requiring business justification and Senior Management approval • There are two long term initiatives considered by Management for the lifecycle of development: – Transition of the STUDIO tools from Delphi Windows Applications to PHP web based tools – The adoption, and integration with, PowerBI for reporting needs	• Delphi is no longer widely supported and availability of developers and resources are becoming limited. The transition of the STUDIO tools from Windows applications to PHP web based tools, appears appropriate, allowing migration when redevelopment of tools is required and supporting the move to more common and readily available skill sets of PHP from the less common Delphi technology stack • Management appear to have established effective planning workflow to enable development decisions taken as either part of BAU work, or requiring business justification	• Continue the removal of the Delphi based Windows application to replace them with web-based systems
Functionality		• STUDIO is an internal application used only by DRAGON staff. It holds all of DRAGON's customer information, schedules activities, and includes API calls to supplier systems • We understand from discussions with Management that STUDIO is used as a core business workflow tool, to support business processes for prospect management, integrated CRM, online quotation tool and a lifecycle contract management suite	• While we understand from conversations with Management that STUDIO supports end to end functionality and workflow across the business, we have not verified the appropriateness of the solution against any business needs or requirements • However, the scope of the system does appear to be broad and used across the Enterprise, therefore there appears to be a clear dependency on the continued operation and capability of the solution to support current business operations • The need for enhancement documentation to support end users, system users and future development activity should be considered	• No actions identified

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Functionality (cont.)		• In addition, Management explained that STUDIO supports customer identification, management of client contact and appointments, price quotation building based on required office phone hardware and services required, scheduling and management of customer installations and post installation support and service provision. Management indicated that following business functions which are supported by the STUDIO suite of tools. – Sales Order Process – Installation process (configuration to deployed ATM) – Monitoring and Helpdesk function – Call allocation – Parts usage – Customer Interaction – Commissions – Settlements – Engineering – Cash Management – CIT	• See previous page	• No actions identified

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Technology		• STUDIO was historically developed as a Windows application using Delphi, a PASCAL based platform. However, new STUDIO development activity is now being completed as a web-based platform using PHP along with the SLIM framework • Management have established a target of migrating to the new architecture, to aid easier recruitment of new developers with more PHP skills and documentation available in the market and public domain. In addition, the use of the new platform is considered to more effectively allow for access via both desktop machines and also mobile/tablet formats	• Delphi is no longer widely supported, and availability of developers and resources are becoming limited • The new platform using PHP should also allow for easier auto-deployment, which is already being used by the development team	• Continue the phasing out of the Windows suite to reduce risk of resources and skills availability
Development (Management)		• DRAGON historically used an open-source ticketing system for development, OS-ticket. OS-ticket is no longer used but is kept for reference. JIRA was introduced in 2022 to assist the team to better manage its activities and pipeline of development requests. In addition, Confluence is also used to document high level activities • There appear to be comprehensive workflows which have been mapped within JIRA to process the requirements from request, through review and to final approval • We understand that Items are released on an as needed basis, there is no fixed schedule for release	• In our opinion, JIRA and Confluence are typically appropriate tools support project management workflow processes and aid in management of the development function	• Continue JIRA use and evolve the available reporting within the tool for activity reporting and to establish routine development metrics and KPI performance analysis • Any expansion of the STUDIO suite of tools to support the enlarged Group or Buyer's operations will require scheduling and resource planning to determine costs and timescale to deliver

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Development (Testing)		• We understand that historically DRAGON has not completed any scripted testing against the code base	• Scripted testing provides efficiency improvements and aids development, by confirming that what has been developed is correct and helps during deployment, to ensure that any changes do not have any impact other parts of the system, considering different scenarios • Without testing scripts that can be run as part of deployment, there is typically limited verification to ensure changes do not introduce issues	• Management should consider Introducing tested scripts against the critical parts of the system, aligned to a test first or test as you develop methodology into the workflow
Development (Database)		• The datasource used is a Microsoft SQLServer database, the route into the database from the codebase is achieved only via Stored Procedures. There is a second SQLServer which is used for data warehousing and reporting • We understand that both the Delphi code and the newer the PHP based code only manipulate the data via stored procedures	• We would consider that data management via only stored procedures reflect good practice from a development perspective, allowing for separation between the application and the data layer • The use of a second SQLServer for dedicated reporting use is good practice, it both takes the load off the main server and also allows for creating data warehouse tables	• No actions identified
Source control		• The STUDIO source code and version history is stored in Git. The centralised repository is a self-hosted one from Bitbucket • For the old Windows based system, all the source code is held in a single repository. In the new web-based system, different parts of the system are held in their own repositories • We understand that the QA branch is deployed to the testing infrastructure and the master branch is deployed to live	• Git and Bitbucket are widely used source code repositories, together they offer an appropriate system to keep track of changes in code and for driving auto deployment. For which we would consider that there is readily available resources • In the new code base, the separation of different functions into separate repositories reflects good practice, which allows for a microservices approach where each function can be managed and deployed separately	• No actions identified

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Hosting		• The application is deployed, to the TNS servers, via Jenkins. Changes and new additions are added directly to the QA branch where it is auto-deployed to the testing environment. When merged into master it is then auto-deployed to the live environment • Auto-deployment only occurs with the web-based PHP systems, deploying to the testing and live web servers. The Windows based application suite requires manual compilation and is deployed manually to a shared drive	• Jenkins is widely used for auto-deployment, with readily available open source documentation and support in the public domain • However, the development team is considering migrating to Bitbucket Pipelines. Bitbucket Pipelines is from Atlassian, the same vendor that provides JIRA and Confluence. Therefore this may provide improved integration with JIRA and use the similar looking interface • Although Jenkins is widely used, moving to Bitbucket Pipelines appears reasonable with potentially more functionality and integrates with the rest of the Atlassian suite	• No actions identified

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

Summary of findings
Proprietary systems - STUDIO

Activity	RAG	Observations	Considerations	Resolution / action required
Development (Security)		• From a security standpoint, the following testing methodologies are not performed consistently during the development lifecycle of Studio applications: – SAST – Static security testing to identify potential vulnerabilities within the code – DAST – Dynamic testing to identify vulnerabilities by using common signatures throughout the application stack – Credentials in Code – Automated testing to verify that credentials, API keys or sensitive URL endpoints have not been accidentally committed • We noted during the demonstration of Jira, Confluence and BitBucket, that MFA was not implemented as part of the authentication process • It was commented that GROUP conduct penetration tests against their applications prior to any major releases	• The Studio suite of applications is critical to business operations at GROUP. Studio stores and processes sensitive data like ATM BIOS passwords, disputes data and more. Insecure code could lead to the exposure of sensitive data through leaks or as a result of active exploitation	• Ensure MFA is enforced for logins to third-party development tools and supporting applications • Consider introducing SAST, DAST and secure code review as standard steps in the SDLC. Many of these checks can be automated • Tooling that can integrate in to the CI/CD process should be considered. Typically the cost of such tools will be based on volume of applications, volume of lines of code and specific features such as dynamic testing etc • Introduction of tooling either included in commercial offerings or available as open source that can identify credentials in code should be considered and run during the code review

Key:
- Significant risk, cost or complexity
- Moderate risk, cost or complexity
- Low risk, cost or complexity

T. Code review - methodology (1 of 3)

Development and Code review

- Our approach to completing the review of the STUDIO platform code quality considers a combination of non-functional attributes of software code, including structure, design, technology, maintainability, reliability, and methods, which together provide an indication as to how well code is written (1)
- Factors including knowledge of the development staff, existing software development environments and processes, and approach to project management can affect the code quality of a software project

Development and Team Resources

- The size and how stable the team of developers is can impact the resulting quality in a number of areas. Developers skilled in using specific coding languages and technologies required for a project generally produces better quality code than a team of developers without a background in those technologies. A small number of developers working on a project maintains consistency in approach and knowledge of the structure, functionality and logic implemented without the need to maintain more formal knowledge bases across multiple resources
- However, with larger code development teams developers who possess a robust coding background, but who have little experience working with a certain coding language or technology used for a project may need to spend before they can write code using that technology, especially if the two code languages are very different
- The amount of time the developer may need to spend learning could result in project delay or code quality issues if the team employs too few other developers skilled enough to complete necessary functionality in line with the project schedule

Software Development Environment and Processes (SDEP)

- Establishment of a proper code development environment and code development processes enable developers to adhere to coding best practices and more effectively collaborate on code changes, which usually contributes to code quality. Code development processes which can be beneficial for code quality include:
  - Version Control: Version control allows developers to review, track, and share code changes. Version control systems also allow developers to maintain code history and link code changes to defects logged in bug tracking systems, which aids in issue analysis and prevention of recurring errors in the software
  - Static Code Analysis: Static code analysis tools provide automated analysis of source code, which developers can use to improve code quality during development. These tools can identify errors in code and variance with coding conventions, aiding developers in creating secure and reliable code that is maintainable by current and future resources
  - Manual Code Review: Manual code reviews encourage discussions of code and design changes prior to implementation or modification. Developers may perform formal code reviews before code is checked-in to a version control branch ready for deployment, or when investigating results reported by a static code analysis tool. Developers who needs support implementing a feature may request an informal code review to provide feedback the code they have created
  - Testing: Developers should write tests for code to provide QA and to document the code's intended functionality. Projects will usually be subject to unit, functional, and integration testing prior to release the live environment

T. Code review - methodology (2 of 3)

Software Development Environment and Processes (SDEP)

- Source Code Documentation: Where developers include comments to document their source code, which can improve software maintainability. Developers should maintain these comments when adding new code or changing code. Additionally, developers may need to follow a specific source code documentation format, depending on their chosen

Logic and libraries

- Aligned to code complexity, the design strategy and coding requirements, will be heavily dependant on the need to address and code specific features and logic within the code to deliver the required functional goals, or to overcome specific technical challenges. The use of existing and properly licensed source solutions (libraries, OpenSource, Modules) can increase delivery speed and simplify. However, where new IP, innovation or inherent technical challenges or new features are being provided, there may be a need to solve these through the creation of new proprietary code elements

Code technology and choice

- The coding tools used for development depend upon the requirements for the product (i.e. platforms and environments being used), the availability of new/latest technologies to be selected as being the most appropriate, and the skills and experience of the development team in place
- The number of different technology stacks employed, and their respective maturity from a technical perspective, with contribute the overall complexity the code, and represent potential impacts on quality

Project Management

- Clearly defining business requirements for a software project allows developers to write code that correctly implements intended functionality. Poorly defined requirements drive unclear design specifications, which can impact the progress of resources and contribute to poor code quality and incorrectly implemented functionality
- Unrealistic project timelines can also impact code quality. Completion of a project in accordance with unrealistic schedules or budget constraints may cause developers to create less than ideal code solutions or to implement without proper testing

Indicators

- Some of the indicators we consider to determine code quality include complexity, technical debt, code duplication, and coupling:

  - Code Complexity: a measure of how easy code is to understand and maintain. Smaller blocks of code containing less functionality are easier to understand and maintain than large code containing complex functionality. Complicated code which uses many different looping or nesting structures increases code complexity and can contribute to poor code quality
  - Code Duplication: Code duplication refers to similar or identical sections of reused or refactored source code. Code duplication can reduce maintainability and increase time needed to modify functionality if the same section of code must be modified in multiple parts of a program. Thus, code duplication may indicate poor code quality. However, a developer may decide to use duplicate code in cases where it improves code readability, with a trade-off of reduced maintainability

T. Code review - methodology (3 of 3)

Indicators (cont.)

- Removing duplicate code and functionality to reusable methods or classes that can be called when needed can eliminate or at least reduce instances of code duplication
  - Coupling: Coupling is a measure of interdependence between sections of code. Tightly coupled code connects functionality between two otherwise unrelated sections of code. As a result, modification of tightly coupled code may affect functionality in unintended ways and impact how well the program functions as a whole, which can contribute to poor code quality
  - Technical Debt: Technical debt refers to work needed to correct previous code. Therefore, technical debt can contribute both to project delay and to poor code quality, as many temporary solutions never receive necessary revisions

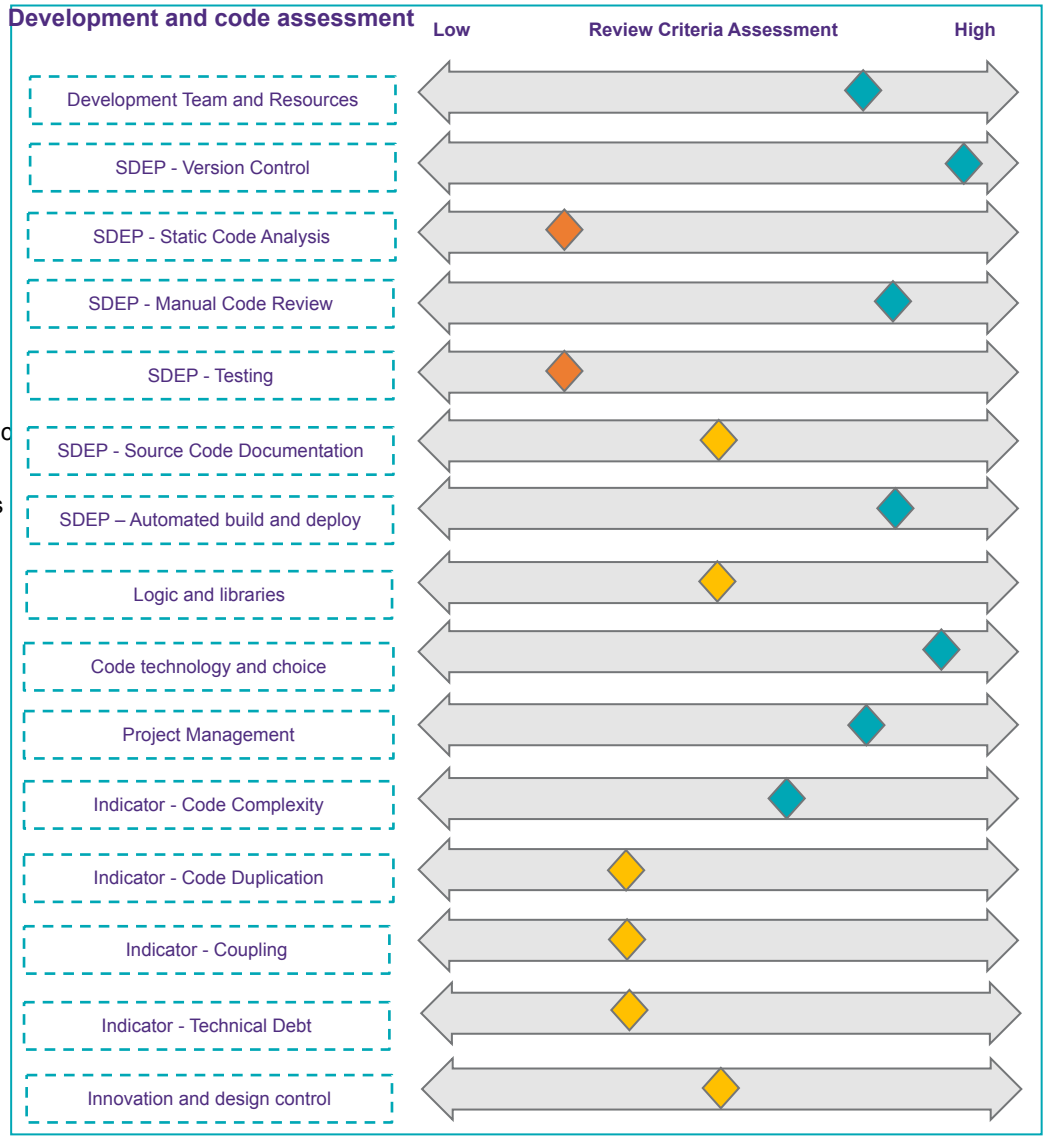
Innovation and design control

- The adherence and delivery of the target outcomes and capabilities of the product being developed will be dependant upon the degree of engagement with the technical design authority, availability of clearly defined product strategy and roadmaps which articulate the intended outcomes, and adherence to delivery of these objectives through regular review and verification of the original goals
- This process can be managed through any SDLC, with Agile providing a more flexible approach to adjustments and changes to the outcomes based on SCRUM sessions within each sprint, or formal change control processes where a more traditional Waterfall approach is used, with both requiring significant participation of the product managers and design authorities

U. Code review - assessment (1 of 3)

Summary of our review IT Due Diligence Findings

- The table to the right shows our view of the STUDIO suite of modules' coding and development practise, and the approach taken against each of the criteria areas we define in our code review methodology. In summary, our comments from our review are:
- Development Team and Resources
  - Long standing development team, who appear to understand the business well. There is a risk that the support of Windows based apps suffer from the lack of Delphi developers, but this is being dealt with via the migration to a web-based system with development done using PHP
  - **SDEP - Version Control:** Version control is a Git repository hosted at Bitbucket. The Development team understands its usefulness and it is used to control the release process
  - **SDEP – Static Code Analysis:** Beyond what is on offer within the code editor, static code analysis is not being used
  - **SDEP- Manual Code Review:** Senior members of the team act as the gatekeepers to the main code base with code review done on a pull request
  - **SDEP – Testing:** There is no automated code testing, so testing is reliant on manual tests which are completed by the project owners
  - **SDEP – Source Code Documentation:** There appears to be a minimal amount of code documentation, however important elements are documented on Confluence
  - **SDEP – Automated build and deploy:** Automated build and deployment is driven by a successful merge to the master branch in Git. Merge to master triggers a Jenkins deployment script, this then auto deploys to the live server setup

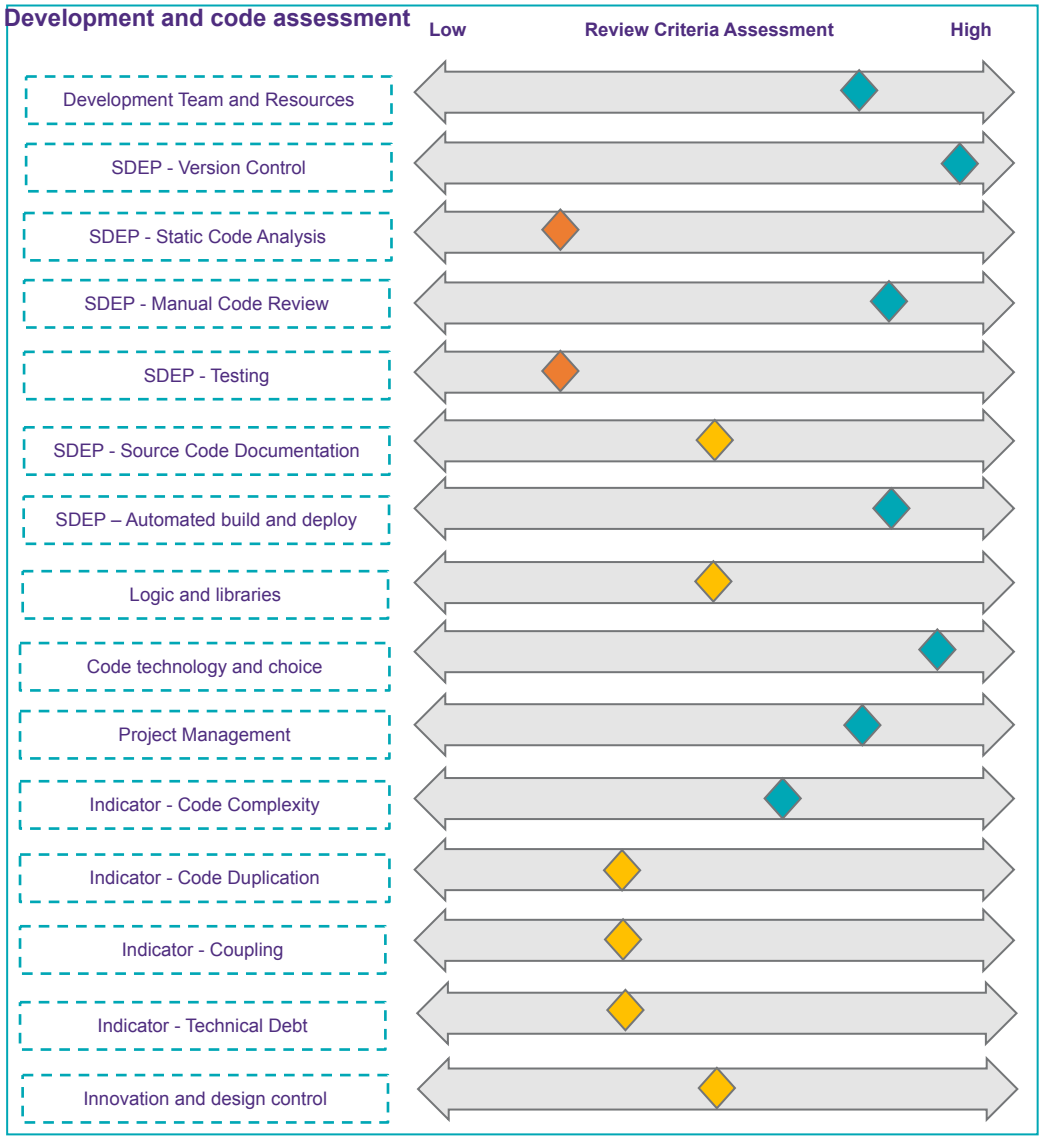


Notes: a. High reflects significant degree of demonstrated best practise, Low rating indicates limited evidence or no evidence

U. Code review - assessment (2 of 3)

Summary of our review IT Due Diligence Findings

- Comments continued -
  - **Logic and libraries:** Libraries being used do not extend beyond the PHP and SLIM framework offerings. We did not see evidence of proprietary, non standard, elements being used or developed
  - **Core technology and choice:** PHP and SLIM framework, hosted internally are reasonable technology choices for this application. We would consider that it would not be difficult to find external programmers to support the development if required
  - **Project Management:** The Project appears to be well managed. It is managed using JIRA with documentation done on Confluence
  - **Metrics – Code Complexity:** In our opinion the code is not hugely complex and is well laid out for reading and understanding to determine purpose and approach
  - **Metrics – Code Duplication:** There is no formal checking for code duplication, the team relies on following best practices to drive down code duplication. There are some and they are being dealt with during enhancements to that part of the code
  - **Metrics – Coupling:** This is also reliant on the team adhering to best practices and dealt with during general enhancements
  - **Metrics Technical Debt:** The team recognises that there is technical debt but it is not formally listed in the backlog. They are being dealt with during general enhancements

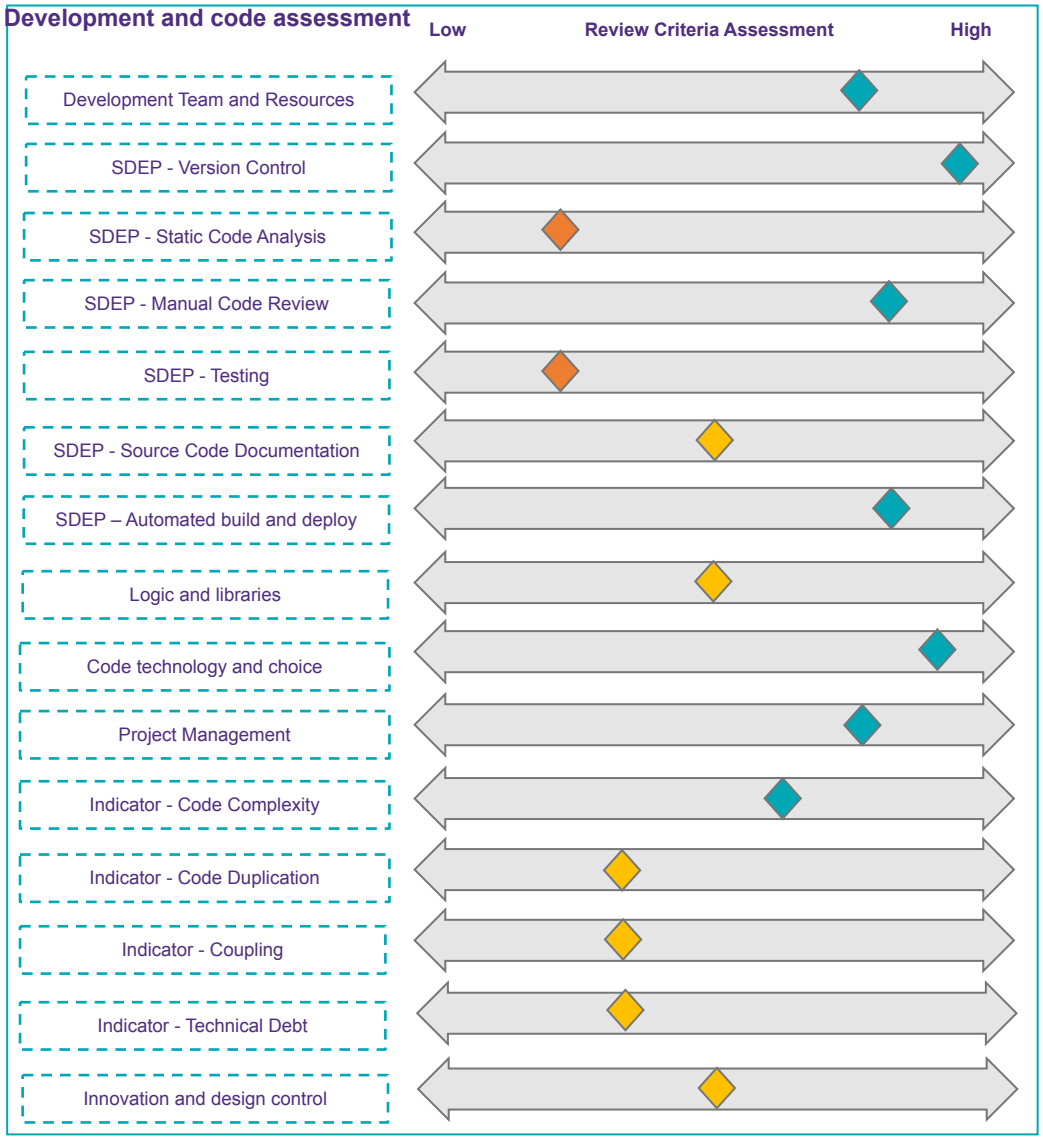


Notes: a. High reflects significant degree of demonstrated best practise, Low rating indicates limited evidence or no evidence

U. Code review - assessment (3 of 3)

Summary of our review IT Due Diligence Findings

- Comments continued:
  - **Innovation and design control:** While there are some examples of specific developments to overcome challenges, technical innovation is limited but the code base is well designed



Notes: a. High reflects significant degree of demonstrated best practise, Low rating indicates limited evidence or no evidence