

Congratulations!

You've inherited some code

You've just embarked on a big new collaboration, you've got to the end of the kick-off meeting and... wham! It's your job to take your collaborators' world-leading code and make it work on your group's machines. And then you find that the code is written in Fortran. And not just any Fortran: the actual, original, Fortran. Welcome to the world of software archaeology!

This briefing paper is targeted at software developers and project managers who deal with taking over code that was developed by others, and where no proper handover has occurred. This paper provides advice on taking over inherited and *dusty deck* code, and highlights some of the key pitfalls.

For advice on how to ensure that code is handed over properly, see the related briefing paper "Help! Your developer is running away!"

Why is this important?

In many cases, issues with older software arise when it is being used by a community, but no developer is working on it. This situation may occur because there wasn't sufficient funding to keep a developer, or because of changes in direction, or simply because the software has been stable for so long that the developer has moved on.

Now what happens if it becomes necessary to change the software? Maybe a change in the operating environment is needed or new functionality must be added. With no developer working on the project, it's not possible to make these changes unless a new developer takes over the code.

Taking over the code can be far from simple. Although users have continued to rely on the software, the software itself will have *decayed*: it will have become more prone to failure and lack of compatibility. Software decay happens because of changes to the system in which the software works. Upgrades to the operating system, deprecation of functionality within dependent libraries,

changes in the underlying programming language, and other factors change the system. The cumulative effect of these changes can stop the software operating as desired - and can even stop it working completely.

Take over code like an archaeologist takes over a site

Start with an inventory

Have you got all the code and documentation you can find? What development environment was used? Do you have any test data? This will also give you a handle on the magnitude of the task.

Secure the site

Shore up the version control system. Is the code repository structured in a sensible way? Are the repository accounts and permissions appropriate? If they don't exist, consider putting in place processes to build and test the code in a straightforward, reliable and repeatable way. Software project management tools such as Maven can assist here. This will greatly help developers starting on the project.

Assign specialists to each excavation task

Are there any members of your team who have good experience with each piece of technology? If so, matchmake their abilities with the technology they will work on.

Prepare for the next generation

Once the code is cleaned up, make sure that the next



generation of developers don't have to do what you've just done! Update the documentation, provide comprehensive test data and make sure it's accessible.

Think like an archaeologist

Look at the problem from different angles

Perhaps reverse-engineering tools or code visualisers might help. Integrated Development Environments (IDEs) such as NetBeans and Eclipse can *profile* code to let you see what is going on in real time. These tools can also identify parts of the code that might need to be refactored.

Try and understand history from their point of view of those that made it

Don't make assumptions. Why did the original developers write the code in the way they did? This approach is more likely to identify problems that are now systematic errors.

Work out how the inter-relation of the artefacts

You can build up a clear picture of dependencies by determining how the code fits together, and how it relates to the original developers use of third-party libraries or code. Creating an architectural view of the components and their interfaces is an invaluable aid for new developers, and IDEs can commonly perform static, as well as dynamic, analysis of the code to help you create this picture.

Work like an archaeologist

Record where you find artefacts

Keep notes as you excavate the functionally significant code, and the code which isn't being used. You can use paper, wikis, simple diagrams or blog posts. Your notes can be lightweight, because even lightweight notes can be used to start the process of documentation.

Use the correct tools for each artefact

Your toolkit may include grep, debuggers, profilers, logging and instrumentation, IDEs, reverse engineering tools, and even Google. Using the correct tools for each artefact means you're less likely to get stuck trying to excavate a piece of code.

Further information and useful resources

Overview of Software Archaeology

http://en.wikipedia.org/wiki/Software_archaeology

Significant Properties of Software Framework: a useful framework of things to investigate and discover

<http://bit.ly/fjXRHo>