

A Single Open Source Repository For Every Use Case



Jenny Evans, University of Westminster
Tom Renner, Haplo

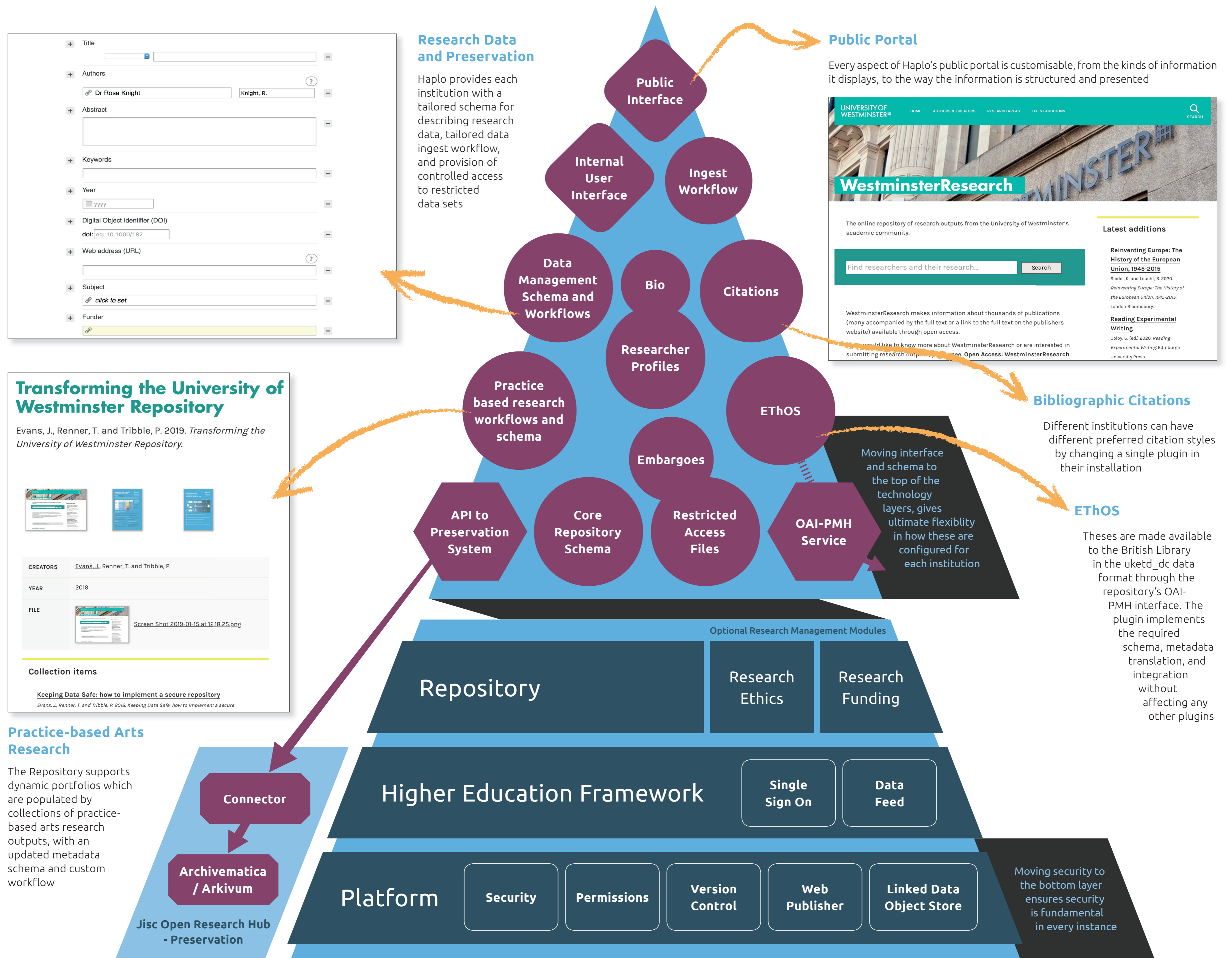
evansj@westminster.ac.uk tom.renner@haplo.com



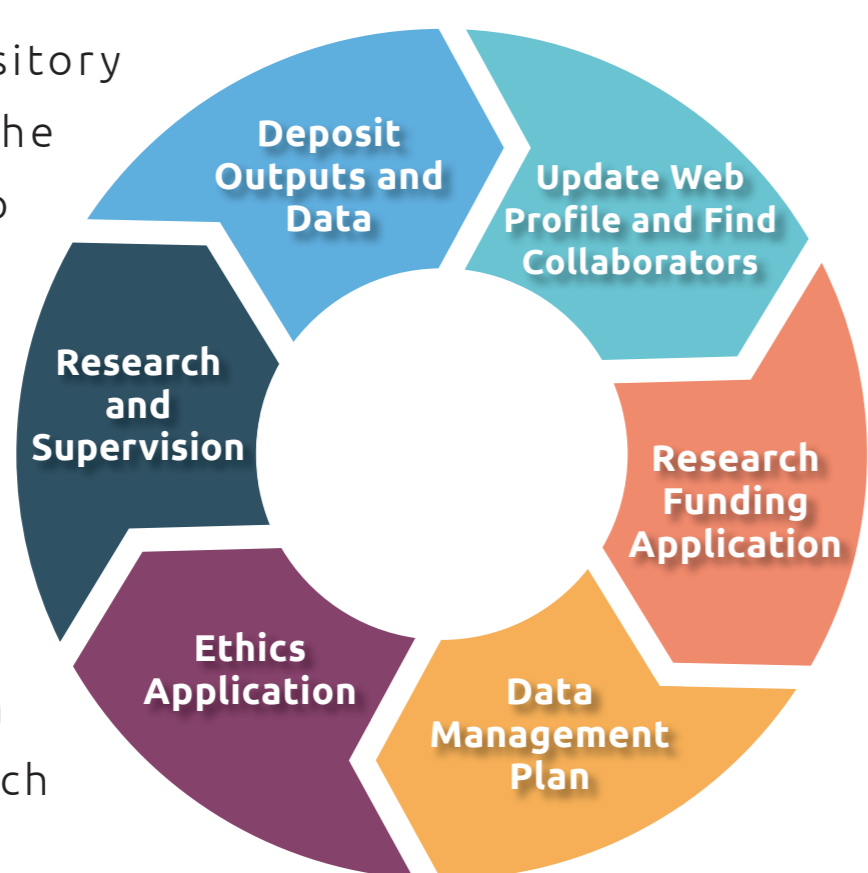
The University of Westminster and Haplo have collaborated to build a next generation single repository for all research, which addresses research data management alongside requirements for all other research outputs, and has enabled high levels of user engagement.

Haplo's innovative technical architecture provided ultimate flexibility in how we setup our repository enabling us to have meaningful conversations with end users about their requirements, confident that we could provide exactly what our researchers' need.

Researchers were actively involved in the design and development of the repository, ensuring the repository delivered what they required and engendering a sense of ownership of the repository.



As the Haplo Repository is integrated into the university's wider Haplo Research Management System, researchers are provided with a single interface for managing their research project from conception through to deposit of research outputs.



A Haplo Repository is made up of over 111 discrete plugins used to customise each repository with institution specific policies, workflows and functionality.

By breaking the elements which make up a repository into layers of many small parts, Haplo can support extensive flexibility in which are used and in how each Haplo system is built.

The ordering of the layers differs from how repository systems are traditionally built.

The lower layers, common to all repositories, provide fundamental

repository functionality, such as security which enables a high level of security throughout the system.

The elements institutions most often wish to customise (such as the interface, policies and schema) are in the top layers for easy configuration.

This approach is enabled by the underlying Haplo Platform, which allows many plugins to exist in the same repository without conflict. It does this by providing well structured ways for plugins to apply policy and communicate with each other, along with functions to manage multiple schema and combine plugin UI into a single coherent UI.