Scenario Based Prototyping – When Open Source meets the Video Star

Paidi O'Raghallaigh
INFANT Research Centre, UCC
Cork, Ireland
P.OReilly@ucc.ie

Frédéric Adam
Cork University Business School, UCC
Cork, Ireland
FAdam@ucc.ie

ABSTRACT
Prototyping is crucial to the success of Information Systems Development (ISD) projects, especially those of a more equivocal nature. Prototyping efforts face inherent tensions between the need for producing high-fidelity complex prototypes and producing them quickly and at low cost. This paper describes how a number of ISD teams focused on stitching together relatively low-cost high-fidelity prototypes through the loose assembly of pre-existing open source software (OSS) components. Video recordings were captured of the role playing use of these prototypes by realistic persona in realistic scenarios. These videos were replayed to stakeholders in order to provoke a response and to capture their rich insights. We use the acronym OSP to represent this method of Open Source Scenario-Based Prototyping. Based on observations of the activities of these teams, the paper is in a position to describe a high level method for producing OSPs.

Author Keywords
Open Source Software, ISD; Prototyping; Rapid Prototyping; Video Prototyping; Patchwork Prototyping.

ACM Classification Keywords
H.1.1 **[Information Systems]:** Systems and Information Theory.

INTRODUCTION
The goal of prototyping is to assist in exploring the design space consisting of the many possible solutions to a given problem. Prototyping is increasingly viewed as crucial to the success of Information Systems Development (ISD) projects, especially those of a more equivocal nature. The goal of rapid prototyping is to develop a prototype in a fraction of the time that it would take to develop a working system and to learn as much as possible and as quickly as possible from the prototype. These lessons can then be built into a further version of the prototype. This process can be repeated in order to reduce the likelihood of arriving at a solution that does not meet the needs of the stakeholders.

Rapid prototyping can, therefore, result in considerable time and cost savings as well as reduce the risk of ineffective project deliverables. While the rapid production of low-cost high-fidelity prototypes has long been the Holy Grail of ISD [1], it remains elusive. In addition, while scenarios-based methods have long been advocated as a means of improving outcomes in ISD [2], scenario-based prototyping remains underutilised. The method described here has emerged from participating on a number of challenging ISD projects that were project managed by the lead author. The ISD teams in each of these projects focused on stitching together relatively low-cost high-fidelity prototypes through the loose assembly of pre-existing open source software (OSS) components. Video recordings were captured of the role playing use of these prototypes by realistic persona in realistic scenarios. These videos were replayed to stakeholders in order to provoke a response and to capture their rich insights.

We use the acronym OSP to represent this method of Open Source Scenario-Based Prototyping. OSP can be described as a general method of scenario-based prototyping in ISD projects. The use of open source software (OSS) is a critical element of the method. It has been found to be particularly effective when used during resource constrained projects where there are high levels of equivocality. The objective of this paper is therefore to: (1) Position OSP relative to other related forms of prototyping; (2) Present a high level method for undertaking OSP; and (3) Outline some of the benefits and limitations of OSP.

The next section provides some background to the concept of prototyping and presents two particular prototyping approaches (i.e. patchwork prototyping and video prototyping) related to OSP.

BACKGROUND TO PROTOTYPING
Dimensions such as form (physical versus abstract), fidelity (high-fidelity versus low-fidelity), scope (horizontal versus

OpenSym '17 Companion, August 23–25, 2017, Galway, Ireland

ACM ISBN 978-1-4503-5417-2/17/08…$15.00
https://doi.org/10.1145/3126673.3126678

vertical) and interactivity (dynamic versus static) can be a useful way of categorising the different types of prototypes used in practice. These categorisations can also help in understanding the trade-offs involved in picking one prototyping approach over another [1].

In terms of form (or representation), prototypes range from less-evolved forms (such as pencil and paper drawn concepts) to more evolved-forms (such as coded software components). Less-evolved forms of prototypes are relatively quick and inexpensive to produce but they do not support real interactions. On the other hand, more-evolved forms of prototypes are slower and more expensive to produce but can support real interactions. They are, therefore, suited to generating complex insights that come from observing actual use of the prototypes.

Another dimension is fidelity (or precision). Fidelity refers to the level of detail or accuracy of the prototype relative to the final solution. Low-fidelity prototypes are quicker and cheaper to produce but can be limited in their power to generate insights as the stakeholders may have difficulty imagining how the final solution will look, how it will work, how they might use it, or what it will do for them. On the other hand, high-fidelity prototypes are slower and more expensive to produce but leave little to the imagination.

A further dimension is scope (or extent). Horizontal prototypes encompass a wide breadth of functionality required from a solution. They are, however, shallow in that they only show particular layers (usually the user interface) of functionality. This helps *"... both the user and the programmer understand the breadth of the system without plumbing its depths"* [3]. Vertical prototypes, on the other hand, take a narrow slice of the functionality required from a solution and explore it in depth through each layer (from the user interface right through to the system layer) of functionality. This allows users to interact fully with a limited piece of functionality. Task-oriented prototypes provide the functionality to perform a single task or a series of tasks [4]. Scenario-based prototypes are similar to task-oriented ones, except that they provide the functionality to support a more realistic real world scenario in a real-world setting [4].

A final dimension is interactivity. Interactivity represents the extent to which the prototype can be interacted with. For example some prototypes are static and can be observed, whereas others are dynamic and can be interacted with. Static prototypes are likely to be quicker and cheaper to produce but are unlikely to generate complex insights that come from observing the prototypes in use. On the other hand, dynamic prototypes are slower and more expensive to produce but can generate complex insights from observing them in use.

Patchwork Prototyping

A challenging question for many ISD projects is whether to build (their own proprietary software), buy (commercial off the shelf software), adopt (open source software), or use a mix of these approaches. Regardless of what decision is made, OSS can play a role in the prototyping efforts of the projects. For example, even if the decision is made to build or buy a solution, OSS can be used to build prototypes to gather insights that inform the requirements for that solution. This nuanced role is largely ignored in the practitioner and academic literature.

Patchwork prototyping is *"...the combining of open source software applications to rapidly create a rudimentary but fully functional prototype that can be used and hence evaluated in real-life situations"* [3]. The OSS components can be stitched together because the code is open, accessible, and modifiable. Furthermore, the individual components are disposable since they can easily be discarded and replaced with others. It is advocated as *"...a rapid prototyping approach ... that shares the advantages of speed and low cost with paper prototypes, breadth of scope with horizontal prototypes, and depth and high functionality with vertical, high-fidelity prototypes"* [3]. The prototypes are released into the real environment and are used by stakeholders in their daily activities.

Because of their form, fidelity, scope, and interactivity patchwork prototypes can help stakeholders to see the breadth and depth of a solution without having to depend on their imaginations. They can, therefore, capture more realistic and informed insights that emerge from observing stakeholders interacting with the prototypes. They are especially useful in exploring ill-defined design contexts [3]. The patchwork prototyping approach is not, however, without some limitations. The patching efforts require the input of highly skilled programmers that must have experience with the development tools used to create the OSS components. Also, the OSS components can have security vulnerabilities that can compromise the servers on which the prototypes are hosted.

Video Prototyping

Video prototypes utilise video recordings as a mean of illustrating how users will interact with a solution [5]. Video prototyping uses paper prototypes or cardboard mock-ups to simulate the solution. The video recordings are organized around scenarios that illustrate how people might interact with the solution in realistic settings. The video storyline leverages a number of design artefacts (such as personas, scenarios, and storyboards) created earlier in the design process. For example, the design scenario acts as the foundation demonstrating how real people would interact with the solution in a realistic setting. The people in the scenario are represented as personas, whose characteristics are drawn from interviews and observations.

Some video prototypes use a narrator or voice over, others use title cards to explain what the personas are doing, either through natural dialog or through a 'talk aloud' procedure [5]. In the past, to produce a video might have required design teams to hire audio visual experts but this is no

longer the case with the availability of free, low cost, and open source video editing software, which is relatively easy to use even by non-experts. Video prototyping is, therefore, becoming a feasible and viable option for prototyping.

THE OSP METHOD

The OSP method entails the following steps, which are executed in an iterative fashion:

- Create the vision for the solution i.e. what benefits it is looking to create.
- Identify the OSS components required to backbone a prototype of the solution.
- Integrate, customise and configure the identified OSS components.
- Identify the scenarios and the personas for the video.
- Create a storyboard from the scenarios and personas.
- Produce and stitch together the required video clips.
- Deploy the video and solicit feedback from users;

The steps do not need to be executed in this exact sequence. An iteration can be performed in a matter of hours or days.

Create the vision for the solution

The stakeholders meet and they set out to create a cohesive vision for what the proposed solution is hoping to achieve. The team utilise innovation games, such as the billboard and the product box design games. Using these games, the team creates replicas of a physical billboard and a product box that presents the essence of their solution (in terms of a name, tagline, benefits and features). The product box is a more detailed version of the billboard. Both games are relevant even though the final solution may not require a billboard or a box. By imagining the marketing copy and packaging for their solution, the team is required to make decisions about important aspects of their vision that may otherwise be difficult to articulate. The team uses these outputs to market their vision to the larger stakeholder group and also to serve as a reminder of the bar they have to rise above in order to build the solution. This vision anchors all further decisions that need to be made in the OSP method.

Identify the OSS Components

The proliferation of production-scale OSS has increased to the extent that some observers are now suggesting that "Open Source is eating the Software World". OSS provides a vast repository of reliable, robust, usable, and feature-rich software. OSS options now span the full technology stack from applications to operating systems, platforms, and even hardware. OSS also includes a wide spread of applications for different functions (e.g. engineering, sales, marketing) and different domains (e.g. health, education, finance). The focus of the team here is on identifying OSS applications that can prototype the high-level benefits and required functionality identified in the previous step. Other than the required functionality, examples of other criteria used in selecting the appropriate OSS applications, might include: underlying technologies used to build the applications and the vibrancy of the community supporting the applications.

Integrate, customise and configure the OSS components

The decision to be made here is to maximise the functional fit while reducing the amount of coding work. Shallow integration is preferred over deep integration. Configuration is preferred over customisation. The key is to have components that provide high levels of functionality and that support high degree of configuration, whereby many features can be easily turned on and off. These settings can be used to alter the presentation of the application as well as altering aspects of its security and functionality. As we will see in a later step, the illusion of integration can be performed by stitching video clips together and this does not always require the code bases of the individual components to be integrated.

Identify the scenarios and personas

Scenarios and personas are used by the team in order to represent the people and the situations in which the solution will be used. They are important design inputs that assist with building understanding and empathy in the team for the people and situations that the solution is being designed for. The scenarios and personas illustrate how the solution and its set of features will be used by a realistic character in a realistic setting. The scenario must identify who is involved, where the activities take place, and what the user does over a specified period of time.

Create a storyboard for the video

Once the scenarios and personas are identified and described, the team develops storyboards. A storyboard is created to demonstrate the use of a solution in a specific scenario. The storyboard breaks down the scenario to show a sequence of rough sketches illustrating each event and the corresponding actions that will end up as clips in the video. The storyboard works best if it shows the scenario through the eyes and actions of a single persona. The storyboard is an important bridge between the scenarios created in the previous step and the video clips to be shot in the next step. The storyboard not only guides how the interaction will be shot but also to encourage the team to think more specifically about just how the solution needs to support the required interactions.

Record and stitch together the video

Once the storyboard is created, the team converts it into simple video clips (such as a person representing a persona using an app) with a voice over to explain what is happening. The illusion of integration across steps and components can be performed by stitching video clips together. Therefore the separate OSS components are integrated through video editing rather than through coding. Each video clip can capture the use of a disparate component. The video clips are then stitched together to makes it look like the user, data, and control is passing seamlessly from one component to another.

The next section offers insights into some of the benefits and limitations of using OSP.

BENEFITS OF OSP

By utilising pre-existing OSS components, OSP prototypes are relatively cheap and quick to build or modify, and they are high fidelity. Due to their loose integration, they require less technical and programming expertise than patchwork prototypes. The underlying OSS components can be 'integrated' by stitching together video clips rather than code bases. In some situations, they can be modified using configuration settings rather than requiring any programming effort. In addition, the videos show how users will interact with the solution in a given context. This facilitates the gathering of deep insights.

LIMITATIONS OF OSP

Unlike paper and pencil prototyping that can be done by almost anyone, OSP does require some technical skills in terms of installing, configuring and possibly customising the OSS components. The visual coherence of patchwork prototypes can instil a belief among stakeholders that the solution is almost complete [3]. A similar outcome has been noted with the use of OSP and the premature raising of 'false hopes' about the time and resources required to complete the solution. Stakeholders view the OSP as almost finished due to the fact that the video demonstrates a functional piece of software being used by a 'real' user in a 'real' context. This can result in stakeholders focusing on small design details rather than the overall question of the appropriateness of the solution [6]. The process can therefore get side-tracked into questions of how the solution should look rather than questions of appropriateness i.e. what it should do and how it should do it. In addition the video recording can represent a normative view [c.f. 6] of how the solution should be used and it typically shows flawless execution of interaction tasks. Unfortunately, the stakeholders observing the video do not typically get the opportunity of reporting on their actual use of the solution and on their actual experiences with the solution. Instead their views are based on how they imagine they would use the same solution in the same or a different context to that presented in the video.

COMPARISON OF APPROACHES

The Table below compares OSP to the other types of prototyping previously discussed in this paper. As we can see OSP combines attributes of both patchwork prototyping and video prototyping but also differs in some regards. Similar to patchwork prototyping, OSP results in a high fidelity prototype built from OSS components. Integration, however, happens mainly at a video presentation level rather than at a coding level. It, therefore, tends to be quicker and cheaper than patchwork prototyping. Unlike patchwork prototyping it is scenario-based and is presented by way of a recording of a role play. On the other hand video prototyping is a recording of a role play of a low-fidelity paper based prototype.

	Patchwork Prototyping	**Video Prototyping**	**OSP**
Form	Digital	Paper	Digital
Fidelity	High	Low	High
Scope	Horizontal-based	Scenario-based	Scenario-based
Interactivity	Dynamic	Fixed	Recorded Dynamic
Speed of prototyping	Slow	Rapid	Fast
Integration	At code level	At concept level	At presentation level

CONCLUSION

At first glance, patchwork prototyping or OSP does not require OSS; the same general process could be followed by using other software. However, OSS provides OSP and prototyping in general with a number of benefits. High-profile OSS is often of high quality and it can contain years of invested design and coding effort that is encapsulated in it's codebase. Given that it is built from the collective experiences of a community, less effective designs have already been tried, tested and discarded [3]. Additionally, most OSS has very active development communities who are willing to provide technical support for the software. Without open access to OSS source code, developers would be limited in how well they can patch together different modules, the features they can enable or disable, and their ability to integrate components together. All of this facilitates the rapid production of high-fidelity prototypes at relatively low cost and little effort [3]. One final point of note is that through this process, ISD teams appropriate OSS in a rather different way to what may have been intended when the software was originally being developed and possibly in ways in which the community never expected. Thus, innovation is not only present during the design and development of OSS but also in its novel use as we see here [3, 7].

ACKNOWLEDGMENTS

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2272.

REFERENCES

[1] Floyd, I. R., Jones, M. C., Rathi, D. and Twidale, M. B. *Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software*. IEEE, City, 2007.

[2] Sutcliffe, A. G., Maiden, N. A., Minocha, S. and Manuel, D. Supporting scenario-based requirements engineering. *IEEE Transactions on software engineering*, 24, 12 1998), 1072-1088.

[3] Jones, M. C., Floyd, I. R. and Twidale, M. B. Patchwork prototyping with open source software. *Handbook of Research on Open Source Software: Technological, Economic, and*2007), 126.
[4] Beaudouin-Lafon, M. and Mackay, W. Prototyping tools and techniques. *Human Computer Interaction-Development Process*2003), 122-142.
[5] Mackay, W. E. *Video Prototyping: a technique for developing hypermedia systems*. City, 1988.
[6] Zwinderman, M., Leenheer, R., Shirzad, A., Chupriyanov, N., Veugen, G., Zhang, B. and Markopoulos, P. *Using video prototypes for evaluating design concepts with users: a comparison to usability testing*. Springer, City, 2013.
[7] Thomke, S. and Von Hippel, E. Customers as innovators: a new way to create value. *Harvard Business Review*, 80, 4 2002), 74-85.