# Methods and Recommendations for Archival Records of Game Development: The Case of Academic Games

Eric Kaltman
Noah Wardrip-Fruin
Expressive Intelligence Studio
University of California, Santa Cruz
{ekaltman, nwf}@soe.ucsc.edu

Christy Caldwell
University Library
University of California, Santa Cruz
caldwell@ucsc.edu

Henry Lowood
University Library
Stanford University
lowood@stanford.edu

## ABSTRACT
The future of computer game history lies in preservation. Most work has focused on the collection and maintenance of completed games, both digital and physical, but for game history to mature we need to look beyond singular objects and venture deeper into games' development history. There is a problem, however, in that most development practice and technique is hidden, either by corporate opacity or a lack of basic techniques for preserving game development documentation. Three issues stand in the way: first, while game studies needs access to documentation, and game developers want to see their work remembered (and perhaps later "remastered"), little work has been done to promote this common cause; second, archives and cultural repositories need guidance from game studies practitioners and game designers in strategies for interpreting development documentation; and third, game developers need to organize their process and documentation outputs with an understanding of future preservation (both for their own internal use and for external study). In this paper, we draw on previous archival movements in the history of science and technology to argue for a unified approach to the preservation of computer game documentation. Our work, derived from a case study of UCSC's game *Prom Week*, presents a first attempt at a communication strategy between game studies researchers, archivists, and developers in dealing with the organization, categorization, and storage of development records. We outline the development process, discuss numerous issues that surprised us in dealing with its documentation, and present recommendations and guidance for similar treatments of other game development work. This is an initial step towards a more structured interaction between academia, developers and archives in the informed retention of game documentation. Further efforts for other classes of games, like those developed outside academia, is necessary to create a holistic, discipline-wide understanding of the benefits of document preservation and use. By engaging in greater collaboration, game studies researchers, developers, and archivists, can begin to solve the problems facing the preservation of games' creative history.

## Categories and Subject Descriptors
K.8.0 [**Personal Computing**]: General – *games*

## General Terms
Documentation, Design, Human Factors.

## Keywords
game studies, game development, documentation, archives, preservation, history.

## 1. INTRODUCTION
As interest in the history of computer games grows, it is imperative that we take a hard look at the state of its long-term preservation. Significant effort is already underway to preserve singular games as executables and static collections of data [1,2]. Many companies now provide access to thousands of old games ported to newer architectures and hardware. Indeed, some collections of games are available immediately in a web browser, just a click away from any interested researcher. However, despite this focus on finished games, there has been no complementary effort focused on processes behind the creation of these games. This is, in part, due to the clandestine nature of game development, and of the general desire of any creator to mask the rough edges of a project, to hide "how the sausage gets made." But it is also a result of the inherent complexity of computer game development, and the technical, aesthetic, and design competencies necessary to understand it. This knowledge deficit, of both records and expertise, privileges the study of games as solitary creations, played by players and interpreted as texts, instead of as constructed objects. People research what is available to research, and for game studies that does not generally include development documentation and source code. And there are certainly important questions to ask of development process and records: Why did a certain design decision get made? What alternate design approaches were tried, and what lessons were learned from them? How did the relationships of development team factor into the final output? Did parties outside the main development group shape decisions? What did they do to solve that technical problem? Without documentary access these questions remain unstudied and unanswered.

The problem of documentary access for game studies researchers leads to an issue for any archival institution wishing to deal with development records. An impoverishment of scholarship about development process leaves archives with little guidance about how to handle development documentation and organization. Many leading institutions are beginning to ingest and archive the development records of seminal game designers and studios: the Ralph Baer Prototypes at the Smithsonian; Jordan Mechner and Will Wright at the Strong National Museum of Play; Steven Meretzky and Richard Bartle in the Special Collections of Stanford University Library. Although these institutions may have

the resources and staff to provide a detailed archival treatment, others looking to contribute to the storage of game development history might not be as prepared for the challenges associated with archiving technically sophisticated development collections. Additionally, as more design and development work is conducted entirely inside computers and networks, the need for strategies in dealing with born-digital documentation is growing.

Lack of proper storage and organization of development records leads, again, to a deficit of historical resources ripe for future game studies investigation. Additionally, the originators of development documentation, designers and developers, lose potential understanding of the history of their craft and its traces through time. It is thus the responsibility of the game studies, archival, and game development communities to steer future efforts towards a better, mutual understanding of game creation and process. This communication needs to happen from all sides — researchers should look more into game development process as a research topic, prompting more archives to seek development collections, and causing developers to orient towards long-term preservation of records.

As game studies researchers, developers, and archivists, we provide, in this paper, a first attempt at an integrated preservation of game development documentation. We first outline the solutions necessary to solve the 'archival problem' of informed storage of development documentation. We then take a case study of a locally produced academic game, *Prom Week*, and try to cover its documentary outputs, development processes, and storage challenges. Along the way, we briefly highlight the development process before diving into documentation issues that surprised and challenged us, and that would be a part of any future work on the preservation of development documentation. Challenges arose with access, migration, identification, and collection of born-digital game assets stored in *Prom Week*'s collaborative systems (cloud storage, version control, personal web space, and email). Commercial document systems are not archivally minded; they inhibit file creation and modification records, ignore (or overly focus on) revision histories, and are susceptible to large scale data loss. We proceed to outline our initial solutions for and analysis of their shortcomings. Finally, we provide a description of our process for storing *Prom Week* in our institutional repository, and provide recommendations for any future researcher, archivist or developer interested in furthering this work. In short, we outline the necessary steps that, if expanded to a larger set of games and their documentation, would change the direction of game research away from studying only final outputs and reverse game creation's ongoing hemorrhaging of its vital history.

## 1.1 Archival Process and Needed Expertise

Archival process involves two main activities: the organization of an archival collection, and the subsequent evaluation of the collected materials by an archive. Usually, an archival collection is not explicitly designed for ingestion into an archive (though we hope to make an intervention discussed below). In many cases archives receive collections in whatever state the owners left them, from meticulous to messy, and it is the archivist's job to sort through and organize them for long-term storage. A pre-ingestion disorganization is a significant roadblock to the understanding and clear categorization of archival documents. It is important that game developers become aware of the difficulties in saving their works to help reduce mishandling and destruction of their material. Further, many developers and researchers may not be aware of the range of documentation that future game studies scholars might find interesting, so we felt it necessary to elaborate. On the other end, many archivists (even those with technical specializations) might not be aware of the documentary context and provenance for many items in a game development collection, or the range of documentary information commonly found in development documentation.

In the world of archives, in addition to documentary provenance, any methodology must also lay out guidelines for understanding the relative value of documents in a collection. This notion of documentary *appraisal* is a core part of any archival process. Appraisal is the means through which an archivist decides on the relevance of different types of documentation; some artifacts can be ignored, and conversely, others must be handled with special care. The knowledge required for appraisal is generally very specialized, and unless game developers and game studies researchers aid in the creation of archival processes there is a great risk that historically significant items can fall through the cracks. The implications for computer game history are particularly dire if there is not a concerted effort to define and illuminate the challenges facing game development documentation.

## 1.2 Prom Week

Our work here is based on a case study of the game *Prom Week*, created by a team in the Expressive Intelligence Studio (EIS) in the School of Engineering at the University of California, Santa Cruz. *Prom Week* is a social simulation of the relationships between a group of high school students in the week before their senior prom. It is an academic research game that incorporates a new artificial intelligence framework, Comme il Faut (CiF), allowing the students in the game to remember past events and build unique and nuanced relationships. As both game and research software, we felt it could help in revealing both the process of game creation and its resultant documentary traces from an archival perspective. Our choice to use *Prom Week* as a case study is primarily due to it having been developed in our collaborators lab, and the unfettered access granted by the development team to all of the project's documentation and hosted material. Also, the development of *Prom Week* was primarily born-digital, with the development team sharing information through email and other network services.

# 2. PREVIOUS SOURCES

Development history, the history of the process of creating games, is just as important as history of the finalized software, and yet most historical effort only focuses on the games themselves. This prioritization of outputs is also present in academic contexts, where publications are prized and the work leading up to them is masked by the clean prose sent to peer-review, and where systems and software garner value through textual remediation. In the history of science and technology, efforts were made in the early 1980s to remedy what was perceived as a lack of documentary records about the most significant science and technology research of the 20th century. In 1983, the Joint Committee on the Archives of Science and Technology (JCAST) published a report, *Understanding Process as Progress: Documentation of the History of Post-War Science and Technology the United States* [3]. It provided a basis for the appraisal of documentation resulting from scientific research process, and argues – as we do for games – that preserving and understanding the documentation about how a project was conducted is as historically important as the final results. Game development, like contemporary scientific research, is an often-collaborative process, involving much exploratory and iterative work before one or more final products

are produced, and producing records through its processes that are likely to be unfamiliar to a non-specialist archivist. The JCAST report highlights three major problems in dealing with historical scientific data and records. These problems map, without much translation, to major issues in digital computer software records appraisal and preservation:

1. The amount of unpublished documentation in game development is not addressed (or categorized by) current archival practice, and it cannot be estimated based on experiences from other fields.
2. There is "an absence of professional consensus on guidelines for the appraisal and description of archival records of science and technology." This lack of consensus contributes to both ingest backlogs at repositories unversed in the material and, in some cases, might lead to the needless destruction of potentially valuable materials. Many institutions are unaware of what they have, what can be done with it, who would want it and what it is worth.
3. Too little is known about the potential users of game documentation or "about how adequately contemporary [archival] practices [meet] their needs."

Essentially, game development is a unique technical phenomenon, over which there is a lack of consensus about appraisal and description, and a lack of knowledge about future historical value. The JCAST report elaborates on the need to save scientists' and technologists' research journals, research data, and other findings, in addition to pre-publication works and reports. It has little to say, however, on the process for saving and recording software and other computational systems and other artifacts associated with game development. Digital assets and systems are not well covered in the JCAST report and now constitute of a majority of the documentation generated through game development.

To further reveal the developmental processes producing documentation, we relied on the idea of a "documentary probe" found in the Charles Babbage Institute's (CBI) 1989 report on the Control Data Corporation's records [4]. In that report different business processes were separated and explored through the use of targeted case studies of specific company products, like the CDC 1604 computer. All documentation relating to each product was located and reviewed to gauge the extent and diversity of the entire organization's records. This "documentary probe" allowed the research archivists to organically discover the business's internal processes through the examination of documentation and to, in turn, cast light on the roles and kinds of documentation created by those processes. Our *Prom Week* case study provides a similar framing mechanism, below, for game development, and particularly academic game development, as a set of processes and as a collection of documentation suitable for appraisal and storage.

Finally, Haas et al.'s *Appraising the Records of Modern Science and Technology: A Guide* provided insight into how to organize a full documentary appraisal of a research lab [5]. This included highlighting the organization and institutional processes at work behind document production, including the relationship between a research lab and its university, and the social interactions of individual researchers. Our full report provides more elaboration on the institutional factors affecting research documentation [6].

# 3. DEVELOPMENT PROCESS

In general, the development process for *Prom Week* mirrors similar efforts from a small independent game studio. There were multiple project leads (graduate students), and a large group of part-time contract workers (undergraduates), steering the development for nearly three years. Along the way, they made novel innovations for certain game systems, and garnered press and awards attention for their efforts. Our efforts began by contacting the developers to gain access to whatever documentation they could provide. *Prom Week*'s academic development context allowed us to conduct document collection and analysis relatively uninhibited. After gaining initial access to the game's development documentation, we organized further, documentary-based interviews with key members of the development team. Interviews centered on locating additional sources of documentation, elaborations on confusing documentary traces, and on the different processes used by the developers in making the game. Eventually, a set of six apparent development processes took shape, and we then organized our documentary appraisal around their specific outputs. Below, we elaborate on the key development processes leading to document generation, most specifically pointing out the different types of documentation that future game studies researchers and game developers would find useful. The processes are drawn from our case study of an academic computer game, but the main contours of the processes should be familiar to anyone engaged with game development. Further research into each of these processes from game studies researchers will definitely reveal even more subtle documentary traces and archival process considerations. What is presented here is simply an initial blueprint, which, if we agree to it as a field, could provide essential initial guidance to non-specialist archivists.

## 3.1 Idea Formation

Most academic research projects owe some debt to previous completed work. If not explicitly dependent on work completed previously, the direction of a project is still likely to be aligned with the specific research interests of a laboratory or research group. *Prom Week*, for example, took off from a foundation built from an accumulation of other projects that together represented years of previous efforts in the fields of artificial intelligence, sociology, game design, and other areas of work. More generally, we realized that academic game development projects not only take established research in new directions - whether explicitly or not - but also like many other games take account of established patterns of mechanics or affordances (such as modding) of games. New research sometimes begins without a distinct awareness of where it will ultimately lead. As a result, early project documentation is fundamentally tied to previous areas of study; every researcher or archivist hoping to deal with documentation of academic game development should be aware of the potentially deep connections any project has with its predecessors and the resulting archival trails that may document those connections.

Formal preparatory documentation includes: design documents and technical specifications, funding and grant materials, administrative documents, meeting notes and schedules, personal notebooks, collaborative online files, email lists, project websites and related blog posts, and project management documentation and services. Secondary documentation includes personal email correspondence, chat logs, social media interactions (like Facebook, Twitter, etc.), personal web sites and ephemeral online services (like scheduling services and group calendars). The delineation here is between items that are used to document the process for the development team and those that are documentary effects of the development team's efforts.

## 3.2 Physical Prototyping

Some game designers use physical prototyping for early design iteration and player feedback. Physical prototyping involves the construction of physical analogs for components of a game play system and emphasizes design failure and testing. This form of prototyping may not be a part of all development strategies, especially with non-entertainment software that does not have a graphical component. However, since most games and user interfaces require visual feedback it is usually possible to prototype small parts of a larger system. Physical prototypes are usually low quality, quickly designed demonstrations of game play systems or visual design.

While it is possible to store the remains of a particularly important physical prototype, it may also be possible to gather documentation about the design and art assets associated with the prototype as a substitute. The important thing is to document how the system played and the specifics of its rules. The low fidelity of the physical prototypes makes their corporeal remains less significant than the ideas and systems they describe. Prototypes also produce playtesting feedback, which, if recorded, can be valuable to understanding a game system's evolution. *Prom Week*'s prototype was unique in that it included an initial, computational version of its AI system. Aside from its digital component (whose appraisal is covered below), *Prom Week*'s prototype featured cards, a paper game board, and character sheets, all of which were designed in Adobe Illustrator, a vector graphics program. This allowed the prototype's assets to change on a whim with the development team printing out new ones as needed. Values for the complementary AI system also allowed for live tuning during play.

## 3.3 Digital Prototyping

Digital prototypes are smaller, less fulfilled versions of some larger, more complex technical goal, but they still often involve the need for organization of software development tools and workflows. It is at this point that the inherent complexity of technical development documentation comes to the fore. In documenting a digital prototype, an archivist may come across records for development environments, prototype iterations, platform specific resources, third-party software and libraries, custom peripherals and hardware, and version control and development management tools. Because of the digital prototype's existence as a software object, much of its documentation will be born-digital content based on either compiled binaries or source code files. However, due to the chaotic nature of prototype development many important early files may either precede version control's implementation or be viewed by the development team as unnecessary for inclusion in later documentation. Early work might be scattered across many locations because it occurred in the period before a stable goal and development trajectory formed.

## 3.4 Iterative Development

If a project is going to be the focus of a sustained development effort it will employ some type of development methodology to manage team communication, feature creation, and other development tasks. Iterative development focuses on creating less potentially unnecessary work by organizing tasks and development effort into shorter cycles of consistent improvement. Scheduling in this strategy is loose, and the onus is placed on the completion of specific tasks with the knowledge that they might end up changing or become more constrained as development continues. Iterative development tries to complete a certain minimum amount of functionality before committing to greater depth or complexity. In academic games and research, development strategies also run concurrently with academic publication and changes in research direction. *Prom Week*'s development progress is mirrored in publications introducing and relating different parts of the game's systems and design [7,8,9]. In this way, academic game development strategies and academic research are entwined throughout the duration of a project.

Full development processes incur additional overhead needed to integrate even more code, systems and features into the final product. Testing procedures, needed to make sure that a game's code is bug-free, require the use of testing frameworks, and systems for reporting bugs to the development team. Common development support software includes version control, bug and task management, and unit test frameworks. Additionally, the development of a complex computer game requires the simultaneous development of programs to help with content generation and the digital asset pipelines. These custom development tools will require the same types of archival appraisal and scrutiny as the primary game object.

It should be noted that iterative development is only one strategy for software development; there are many others but we chose to focus, for brevity, on the strategy used in *Prom Week*'s development.

## 3.5 Release and Dissemination

A computer game release represents the demarcation between the internal development process and the external experience of the game by a more general audience. Up until release, most of the documentation and design decisions hinge on the development team and the small community of those who have playtested and critiqued pre-release versions of the software. After release, the development shifts to accommodate feedback from a larger audience. More specialized game software may only have a limited release to a specific community of practice with more nuanced and targeted feedback. General release games exist in larger global media context and may receive international press attention and criticism.

After release, the technical and project documentation is also now at greater risk of abandonment and obsolescence due to a shift in the academic and/or professional focus among the development team towards newer projects.

## 3.6 Refactoring and Continued Development

In some cases, release signals the end of development, and in others the beginning of prolonged maintenance and improvement. Some projects rely on data generated by a software's release, and will continue generating research documentation and data for a significant period of time. This aspect of projects is potentially problematic for any preservation effort, since there is no clear means for continuous retention of data. Large scientific projects maintain and follow data management plans for continuous experiments, but if there is not a concerted effort by a development or research team to allow for long term preservation of data streams and/or continuous software updates, much of this subsequent work will be lost.

# 4. DOCUMENTATION CLASSIFICATION

Along with an analysis of the development processes, we organized the resulting documentation generated from our study into generalized categories. This section outlines the basic documentary classification for a development process, and then dives more deeply into specific, significant issues to be faced by

any future collaborative work between games researchers, developers and archivists. Certain aspects of the description may be apparent to each audience, but the general description is necessary to cement basic categorizations that will be referred to in later sections.

## 4.1 Software Development Files

Software development files fall, broadly, into three categories: source code, creative content, and external assets.

Source code documents are programming language text files that describe a game's logic, manage its presentation, and interface with a host platform's operating system and underlying hardware. However, just having the source code is not enough to run the game software. The source code must be translated from a human-readable programming language into a series of instructions understandable by a computer's processor.

Creative content includes audio, visual or other structured representation files used by the compiled source code in the execution of the game.[1] Sometimes creative content is included in the executable file, and at other times it remains separate (and is subsequently loaded and referenced by the executing program). These assets are distinct from source code in production method, size and consistency of format. They are produced by a wide variety of third party software, including audio creation, image manipulation and 3D modeling programs.

External assets are any files referenced or used by the executable that are not produced by the development team. Sometimes source code will be compiled into smaller chunks that share common functionality; these external libraries of code are then linked to the executable for use when the program is running. Programmers borrow or license other software libraries if they do not want to redesign functionality already created by other programmers. Any functionality provided by a game but not written by the development team may also contain source code or even further software dependencies for its operation. It may, therefore not be possible to access all of a game's functionality even if you possess all of the files created by the development team.

## 4.2 Research Files

Research documentation is any pre-publication commentary and data generated by the development team. Commercial game development incorporates various strategies for player tracking, gameplay metrics and achievements, and thus any information about data collection and retention is classed as 'research'. Some of this research might not be conducted by the core development team, and would require more specialized knowledge of corporate structures to locate and store. Academic game development is unusual in that the development team is simultaneously building a piece of software and a research agenda. The game system is a research output, and the details of its design and implementation are subject to publication as it is being constructed. Any research files that are available, especially if they may help future scholars interpret the files influence on the development process, should be retained.

## 4.3 Self-Documentation Files

Development teams create promotional materials for press and conference appearances. Screenshots and gameplay videos document a game and provide a preview of the game play experience. When source code is unavailable, unrecoverable or unable to compile, promotional materials might provide the only understanding of how a game functioned and what it was like to play. Many games may not ever document themselves in this fashion, and it should probably be standard practice to produce secondary video documentation of a system functionality and user interaction. Documentation may also be provided for outside developers if the system or codebase is robust enough for external use. This documentation is different from internal source documentation in that it is for the benefit of those outside the main development process. Documentation provided for users and developers unfamiliar with the development process is valuable to a preservation and archival effort. Any document designed to share knowledge about a system for a potential developer is significant to its future understanding by researchers and interested developers.

## 4.4 Further Documentation Study

Each documentary category begs for more thorough investigation from researchers, and better examples from development collections provided to archivists. To date, there is still relatively little discussion of particularly salient research areas. Better access to software development files could tell us about the history or evolution of computer game technical development and programming strategies, which in turn would benefit not only game studies, but software studies and even game developers. Similar treatment for the aesthetic and creative documentation would provide even more insights and potential study by game art historians, and other creative output scholars. More intensive focus on the ways that commercial and academic entities conduct player-centered research will open up space for social scientists, humanists, and historians of science and technology, to understand at a deeper level the relationship between development practice and player experience. Additionally, promotional and marketing materials constitute a significant and understudied component of most game development cycles. A closer understanding of their importance to the development process is crucial for a full picture of game creation and distribution. By collaborating to increase access and storage of important development documentation, researchers, developers and archives can provide a basis for a host of novel investigations in numerous humanist and technical fields.

# 5. DOCUMENTARY CHALLENGES

Our efforts to organize and appraise *Prom Week*'s documentation led to many surprising issues associated primarily with born-digital documentation. Digital documentation is subject to a future necessity for migration and remediation in a way that physical records are not. Digital documentation is subject to continual maintenance for its entire archival life. Documents stored digitally must be suitable backed up, and maintained through data migration either from physical medium, or to new locations on the cloud.

## 5.1 (Lack of) Document Organization

Born-digital development documentation is rarely organized in a completely coherent manner. The ease with which files can be created, moved and destroyed leads to increasing organizational entropy as development processes continue. Certain classes of documentation are also privileged, with software development files usually stored in a managed version control repository. Other documentation, like research files, project management, and marketing materials can collect across shared network storage, on third party cloud services, on personal servers, and individual hard

[1] Source code is also "creative", but requires different preservation strategies and is therefore a separate category.

drives. Organizing and collecting this information is made more difficult if the developers followed no explicit organizational strategy for document creation.

*Prom Week*'s version control repository held most of the project's finalized documentation. Cloud services hosted additional files related to demonstrations, research publications, and individual development efforts. Both locations contained a lot of unused or preliminary documentation, including of abandoned features and revisions. Shared cloud documentation had no discernible organization. Developers from the team helped establish the pertinence of some cloud files, others remained a mystery, lost in the development shuffle.

## 5.2 File Type Versioning and Obscurity

Finding software to interpret all the different file types was a problem for *Prom Week*'s appraisal. We ended up identifying 60 different file extensions associated with over 30 different programs. Additionally, many of the file types could be interpreted by multiple versions of a specific piece of software. For instance, Adobe Creative Suite programs, like Photoshop and Illustrator, share the same extensions for most of their versions. Some features in older files are not representable in newer versions of the software and vice-versa. This is an issue when you load an older file in a newer program; it attempts to change the file into a newer format, making it unreadable to the program that created the original file. Determining the versions of a specific file type are not straightforward, and though our analysis used multiple file identification tools, none were particularly suited to the diversity of game development documentation.

Some other file types are so obscure that they may remain unidentified. Two examples on the above list are .vue and .lel files. Without extensive research (something far beyond usual archival considerations) it was not possible to determine what the files were or what program created them. Starting with the .vue file, which is a Visual Understanding Environment file or a Tufts mind-mapping software program, our identification efforts met with significant difficulties. All common information available pointed to the .vue file being a 3D geometry file, which made no sense in the context of *Prom Week* (a 2D game). After examining the header information of a totally different file type, a .vpk file, we determined that the .vpk was an archive of .vue files and that it was associated with Tufts University. Only after making the Tufts connection could we determine the origin of the file extension.

The .lel file was even more obscure, and after significant web investigation, our best guess is that it was associated with a Windows operating system modification program. The online community DeviantArt provides user created programs to modify operating system themes and aesthetics. A program that modifies the Windows 7 login screen seems to have accepted .lel files. This is no longer verifiable, however, because there is no longer any trace of the program online. All links to available downloads no longer work. Without an explicit record of the types of programs used in development, it may be impossible to determine the content of future files.

## 5.3 Email Clarity and Confusion

Email correspondence is a major communication tool for any modern development team. As a tool for researchers, email records can reveal much about the history of project that even the developers may have forgotten or confused. Development teams organize email lists based on function, with general correspondence, content creation and technical implementation sometimes occupying individual lists. Many teams will use institutional or company-centric mailing lists and subdomains. This makes it possible to acquire a downloaded version of a project's email archive. Any email sent to a list should be available, with the obvious exception of emails sent between team members individually and not to a list. Given the existence of email parsing programs, as long as the email text is in a recognizable format or structure it should be possible for future researchers to interpret it. Email correspondence helps explain project process and structure, and also cements the development timeline. Developers are sometimes fuzzy on the exact timing and order of events; the email record can emphatically clear that up.

The *Prom Week* project took years, and the members of each list changed over the course of that period. The emails did account for everyone involved in development, and even helped core developers to remember when certain people joined and left the team. Due to the length of the project, some core development members were unsure of what each email list actually discussed. Some smaller lists were created for specific, short-term purposes and then changed focus or were abandoned.

## 5.4 Lost in the Cloud Services

Cloud services are any online document collaboration or backup system controlled or managed by an outside corporation. Common examples include Google Drive and Docs, Box.net and Dropbox. These services are convenient for sharing files amongst team members because they ensure a consistent and safe location for documentation. Although the services are not uniform in functionality, they all share common shortcomings that can create problems for digital archivists and long-term preservation. The types of files stored on cloud services tend to be organizational documents, demonstration files, and non-finalized creative content. Removing information from these services is paramount as they are not designed for long-term preservation or storage and are subject to market forces.

### *5.4.1 Access Restrictions*

Services require users to have accounts for upload and collaboration. Additionally, once an account is set up it needs to be granted access to specific files or directories. Institutions, therefore, need to maintain an active account on each service used by the development team, and then rely on team members to provide access. Hopefully, all development files are organized in a coherent fashion, otherwise document sharing might prove onerous for team members needing to individually share files, or locate them across multiple locations in a shared directory.

### *5.4.2 Migration and Loss*

Once files are located and accessible, it can be problematic to move them off of a service. One issue is ensuring that the file tree hierarchy and folder organization remain intact after a download. Maintaining file organization is important for understanding development process and for locating specific files. Many services provide robust features for search and organization based on tags, keywords, and other service specific functionality. In many cases these organizational structures will be lost when files are migrated. Additionally, most services will record file creation and last-modified information through timestamps embedded in a file's metadata. Because downloaded files may represent new creations from the perspective of the receiving operating system, this creation and modification information can be lost in transfer. Migration presents serious problems to file metadata integrity and should be handled with care when encountering files stored on cloud services.

### *5.4.3 Revision History*

Many cloud services provide some form of file version and revision tracking. While this functionality is convenient for users of a service (in case they make an accidental change or mistakenly delete a file) the information associated with revisions may not be accessible after removal from the service. In order to save space, it is common for cloud service providers to keep only a certain number of revisions or to slowly delete revisions for files that have not changed over long periods. It may be possible to pay for service enhancements to keep revision information permanently, but accessing and downloading previous revisions is generally onerous and technically challenging. In many cases knowledge of proprietary Application Programming Interfaces (APIs) are required to access revisions and other relevant metadata.

## 5.5 Dropbox

The *Prom Week* team used a shared Dropbox folder for documentation not already stored in source control. Every team member had access to this folder, and that is reflected in its haphazard structure. There is no consistent scheme for file folder names, each being named for a specific task, like a conference publication, or a specific person ("Ben's Stuff"). The Dropbox account contains over 4GB of documentation and over 4000 individual files. The documents cover every category listed in previous sections, although any source code is usually from demonstration versions of the game or unrelated programmatic tasks.

Dropbox also proved problematic because the ability to sync the service with a folder on a user's local file system. When we synced with the *Prom Week* directory, all file modification dates were updated the day and time that we last synced, erasing the previous creation metadata. In the online interface, the original creation and modification information is present, but could only be extracted using custom scripting, a task well outside any normal archival workflow. The scripts we wrote showed that such information could be removed from the service, but we do not have recommendations for the best way to make use of it, nor how to attach it to specific files. Extensive file revision functionality is available through Dropbox's Packrat service. The *Prom Week* team did not use that added (paid) feature and so their files have no accessible revision histories.

## 5.6 Google Drive

The *Prom Week* team stored around 40 files on Google Drive, and they mainly related to documentation of the development process. Things like task lists, prospective features, and design documents are among the most prevalent files.

A project lead shared their Google Drive folder with us after we set up an account on the service. All files were stored in a single folder. Google Drive's interface allows you to share individual files located anywhere on the service, making it potentially problematic if a number of development files are located outside of a shared folder. Google documents also have no fixed logical form. When downloading a document from the service, the online file is converted into a variety of formats based on user request. Therefore, migration and information loss must be immediately considered when dealing with these files. Google documents also record document histories and revisions, but they are sometimes overly specific. Additionally, Google Drive culls old revisions at regular intervals if there is no recent activity. If a file is being heavily edited, however, the revision history will list every minute edit by each individual. Clarifying a consistent policy for revision tracking will be paramount in dealing with a potential deluge of revision files in Google cloud services. Exporting eliminated all revision history and file modification information. As with Dropbox, we were able to recover that information from Google through their JavaScript API, but have not solved what to do with the metadata nor what final form it should take.

## 5.7 Version Control

*Prom Week* used version control to manage all source code and technical implementation documentation. The project used Subversion, a centralized version control system. Subversion, abbreviated as "svn", is the default version control software for the School of Engineering at UC Santa Cruz. A shared server hosted all student projects and managed access through standard university accounts. The *Prom Week* project lead granted us access to the source repository and we had no trouble copying all development and repository files. A slight note, during the course of our investigation the svn server changed domains. As a result most of the development team did not know exactly where their files were for a brief period. Luckily because *Prom Week* is still an active research project, team members almost immediately remedied the situation. An older project or one without recent activity might have been lost entirely.

*Prom Week*'s source files include numerous versions of the game, along with demonstration versions and related side projects. This makes finding the actual files responsible for the final game rather difficult, but provides a rich set of materials of potential interest to future researchers. Additionally, external dependencies are mixed in with source files, making it sometimes hard to identify files created by the research team. Many research dead ends and half-implemented features are present, still lingering in sub-directories, which again compounds the trouble of finding a singular set of development documentation while simultaneously offering important future insights. The repository is a representation of the hectic, dynamic environment of research software development. *Prom Week*'s documentary complexity is not unique, after years of development most software projects will contain a significant amount of dead code and abandoned effort. For appraisal it is most useful to just save everything in the repository, since it will not be immediately obvious which files are important to the project or future researchers.

# 6. STORAGE AND REPOSITORY

The University of California Library provides digital repository storage services through the Merritt digital repository. Files stored for long-term preservation are indexed and searchable based on title-level descriptors (like title and creator). Larger groups of files are compressed and uploaded to the service where content listings of individual files are automatically created. To upload *Prom Week*, its documentation was broken up into 24 zip archives containing all digital documentation for the project and recordings of the interviews conducted for this work.

1. Organizing all documentation into coherent chunks for compression

*Prom Week*'s documentation was grouped (by the development team) according to online services. That is, the original born-digital file organization consisted of: shared folders on Dropbox and Google Drive; email archives for development mailing lists; source code and finalized creative content in Subversion version control; directories from personal websites of the development team; and directories from team development blogs.

Individual interviews with the development team were stored, with topical indexes, in separate compressed archives. All data

stored on online services was left with its original file organization and hierarchy, even if that made organization less clear.

2. Creating a file manifest for the compressed documentation

The manifest is a spreadsheet describing the title (general description), original creation date and creator for each archive, in addition to a hashed checksum for file verification. Descriptions are necessary for search and indexing in the digital repository. Titles included an overview of the contents of each archive and (when applicable) the version of the storage software used.

3. Uploading finalized compressed content to the repository

After validating the files according to the generated checksums, each file is paired with its descriptive information and available for download from the repository.

*Prom Week*'s combined documentation included: an archive of all project mailing lists; the content's of the team's Dropbox and Google Drive folders; 13 developer interviews (with topical indexes); an Archive-It.Org web crawler archive of the team's online demos and blog; a current development snapshot from their Subversion repository, and a copy and dump of the entire repository itself. *Prom Week*'s development documentation totaled over 7GB and 17,000 files, all of which are now available for download.

# 7. RECOMMENDATIONS

Our organization and appraisal of *Prom Week* led to a realization that there are many possible ways to improve the archivability of game development documentation generally and at educational institutions specifically. Below is a set of the most salient recommendations that became apparent through our research. Most of the recommendations are directed at academic researchers, and focus on making them more aware of the archival issues associated with modern software development methods. But we believe that any organization, commercial or not, could benefit from these recommendations, as they lend themselves to better organization and classification of born-digital content.

1. Establish a laboratory- or department-wide data management policy that outlines how data will be collected, organized, described and archived, according to the standards of the intended institutional or discipline-specific data repository.
2. Produce data and documentation in formats that are easily archivable or can be converted to easily archivable form.
3. Store all relevant development documentation in as few separate locations as possible. Create a manifest or directory that describes and provides access to all project documentation.
4. If possible, ensure that your source repository is available via the open Internet in read-only form.
5. Provide source control check in comments that are descriptive and can be understood by repository managers who were not involved in your project.
6. Record software programs used in the creation of files and assets (including format and version information).
7. Conduct development correspondence on official group mailing lists as much as possible.
8. Assign responsibility for digital archives and institutional software development archives to a specific member on the document appraisal team. Ensure that this team member is well-versed in software development processes and documentation.
9. Instate data management plans for all game design and development projects and courses.
10. Develop instruments (such as transfer and data deposit agreements) and policies for ingestion, description, and access with regard to institutional software development records.

# 8. CONCLUSION

We have outlined an approach to aid understanding, appraisal and retention of documentation related to games and their creative processes. Game creation can be a complex and confusing task for developers, and even more so for archival and collections staff untrained in game development methods and practices. As a first step towards alleviating some confusion in dealing with game development records, we have illuminated a general process framework, provided an appraisal of the documentation produced, presented an example of a digital repository approach and enumerated recommendations to improve academic game archivability. Our work argues for a more unified practice to address the problems facing scholars interested in historical game documentation, developers concerned with their historical legacy, and the archives tasked with organizing and maintaining historical software collections, in general.

Future game studies work will be based on recovering and interpreting development records, and other historical documentation produced by game developers. Perhaps the most significant takeaway from our work is the need to understand that process documentation should be saved, that it can be stored for effective future use by researchers, and that input from developers and other experts is a key to better methods and practical use of development documentation. Many cultural institutions and archives can provide a relatively future-proof haven for development documentation, and it is important to explain to cultural institutions why game development documentation needs to be preserved, and to get developers to organize their information is a more preservable state. We have begun to communicate those needs in this paper, and are counting on future archivists, game studies scholars, game developers and others concerned with historical games to continue pushing for better methods and more direct case studies into all manor of games, general software, and other digital artifacts. Unless there are foundational strategies in place to ensure the safe storage and retention of historical game documentation in permanent institutional archives, much of game history is at risk.

# 9. ACKNOWLEDGMENTS

The work in this paper was supported by NEH Digital Start-Up Grant (HD-51719-13).

# 10. REFERENCES

[1] McDonough J. P. 2010. *Preserving virtual worlds: Final Report*. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign.

[2] Pinchbeck D., Anderson D., Delve J., Alemu G., Ciuffreda, A., and Lange A. 2009. Emulation as a strategy for the preservation of games: the KEEP project. In *DiGRA 2009-Breaking New Ground: Innovation in Games, Play, Practice and Theory*.

[3] Elliott, C. A. 1983. *Understanding progress as process: documentation of the history of post-war science and*

*technology in the United States*. Society of American Archivists.

[4] Bruemmer, B., and Hochheiser, S. 1989. *The high-technology company: a historical research and archival guide*. Charles Babbage Institute, Center for the History of Information Processing, University of Minnesota.

[5] Haas, J. K., Samuels, H. W., and Simmons, B. T. 1985. *Appraising the records of modern science and technology: a guide*. Massachusetts Institute of Technology, Cambridge, MA.

[6] Kaltman E., Caldwell, C., Wardrip-Fruin, N., and Lowood, H. 2015. *A Unified Approach to Preserving Cultural Software Objects and their Development Histories.* Center for Games and Playable Media, University of California, Santa Cruz.

[7] McCoy, J., Treanor, M., Samuel, B., Tearse, B. R., Mateas, M., and Wardrip-Fruin, N. 2010. The Prom: An example of socially-oriented gameplay. In *AIIDE*, 2010.

[8] McCoy, J., Treanor, M., Samuel, B., Mateas, M., and Wardrip-Fruin, N. 2011. Prom Week: social physics as gameplay. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 319–321.

[9] McCoy, J., Treanor, M., Samuel, B., Reed, A. A., Mateas, M., and Wardrip-Fruin, N. 2013. Prom Week: Designing past the game/story dilemma. In *Proceedings of the 8th International Conference on Foundations of Digital Games*.