




Evolution of Information System Design Methodologies: The IFIP Conference Management Problem Revisited

Anthony I. Wasserman^(✉) 

Carnegie Mellon University – Silicon Valley, Moffett Field, CA 94035, USA
tonyw@acm.org

Abstract. Hardware and software technologies have evolved greatly throughout the history of computing. This paper illustrates some of those differences by comparing a modern approach to information systems design to the papers presented in a 1982 conference that showed 13 different software design methodologies applied to the problem of creating an information system to manage a technical conference, including submission and review of technical papers and overall organization of the conference. Those solutions predated the World Wide Web, the advent of personal computers, graphical user interfaces, agile development methodologies, and various modern tools. The goal of this paper is to create the same application, taking advantage of four decades of advances in methodologies and tools.

Keywords: Methodologies · Process · Waterfall · Agile · CRIS

1 Background

1.1 Advances in Computing Technology

The world of computing has changed drastically over the decades. Going back four decades, the IBM PC was new in 1982, and the Macintosh came out two years later, bringing mass adoption of graphical user interfaces, which came to Microsoft Windows 3.1 in the early 1990s. The Internet evolved from the ARPANET, which adopted TCP/IP in 1983, leading to the beginning of the World Wide Web in 1991. Smartphones and other mobile devices gained Internet connectivity much later, starting with the NTT DoCoMo phones and i-mode service in 1999. This ubiquitous connectivity has also led to dynamically scalable hosted platforms and applications running in the cloud instead of on fixed hardware.

Software has also evolved rapidly. The first relational databases suitable for commercial use became available around 1980, and were often accessed across a client-server network that connected terminals and personal computers to servers. The Unix operating system was released in 1974, but only received wider use after the release of Version 4 of the Berkeley Software Distribution (BSD Unix) five years later. New programming languages and tools emerged, gradually replacing legacy languages such as FORTRAN and COBOL. Many information system developers used Oracle's

PL/SQL to manage their Oracle databases, which gained a leading share of the database market. Today, a vast quantity of unstructured data is managed by tools such as Hadoop and MongoDB which do not use SQL and relational models.

As the hardware and software foundation changed, so did the processes by which information systems were designed and developed. Schema design focused on relational models of data, and object-oriented design approaches emerged. More importantly, the flaws of the traditional waterfall model of development became apparent, and have been increasingly replaced by iterative and agile development. Beyond that, many organizations have built continuous integration and deployment into their agile processes, combining their development and IT operations (DevOps).

More detail about all of these changes is beyond the scope of this paper. The main point is that systems are designed, developed, and deployed much differently today than they were in the early 1980s.

1.2 The 1982 CRIS Conference

It's very unusual to find old system design artifacts, even when the resulting systems remain in use. The existence of a published set of designs for a single application is particularly rare, and provides an excellent basis for comparison.

In 1980, IFIP Working Group 8.1 (Design and Evaluation of Information Systems) organized a project known as CRIS (Comparative Review of Information Systems Design Methodologies) whereby people could submit a paper showing how their methodology could be applied to the design and development of an information system that could be used to manage all aspects of a technical conference. Thirteen papers were accepted (including one by this author), and seven (including this author's) were presented at a 1982 conference [1].

In general, the accepted papers (excluding this author's) focused primarily on the analysis and design phases of solving the problem, with the methodologists creating elaborate conceptual data models and extensive functional specifications, often including formalisms such as pre- and post-conditions. Most of the methodologies used a method-specific graphical representation to show such things as objects, relationships, and control flow. Many of the solutions were careful to include "edge cases", exceptional situations that could occur in managing the conference.

Some of these methodologies were being used in commercial settings to design systems, but the majority were still in a research stage, with the conference management application allowing the methodologists to test and refine their approaches. Taken as a group, they represented the state of the art in information systems design circa 1982.

Even then, however, it was clear that the waterfall model of system development had significant weaknesses, including the often-lengthy interval between requirements gathering and system availability. Both the requirements and the available technology often changed during the development period. The high rate of system development failures (late delivery, over budget, poor usability) led to extensive efforts in the information systems and software engineering communities to improve the success rate.

The concept of agile development was the most significant approach to emerge from this work. While most readers will be familiar with agile approaches to development, we include here some of the most significant aspects of agile methods as a prelude to showing its use in addressing the CRIS problem.

2 Agile Development

2.1 Basic Concepts

The Agile Manifesto [2] appeared in 2001, and challenged all of the traditional development processes with its preference for individuals and interactions over processes and tools, and working software over comprehensive documentation. The 17 co-authors of the Agile Manifesto enumerated 12 principles of agile software development. Among the more significant principles are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timetable
3. Business people and developers must work together daily throughout the project.

Agile development downgraded the value of the detailed design documents found in the CRIS papers, and other similar methods of that era. The expectation was that the “business people” had the domain expertise and that developers needed to work closely with them to make certain that the code reflected the customer’s requirements. While not every application was well-suited for agile development, the conference management example was a good fit for this approach, while designing the avionics for commercial aircraft was not. The leaders of the agile movement also gave a lot of attention to developers and teams, recognizing the importance of technical excellence. They encouraged empowered, self-organizing teams to produce the best architectures, requirements, and designs. Teams would regularly reflect on how to become more effective, then tune and adjust its behavior accordingly.

The Scrum [3] method was developed by Ken Schwaber and Jeff Sutherland in the 1990s, and has emerged as the most widely used agile method. In a Scrum project, a team, led by a Product Owner and a Scrum Master, would break the project down into sprints, each covering a period of 2–4 weeks. Each sprint begins with a planning meeting to identify the product features (and/or non-functional enhancements) that the team intends to complete during the upcoming sprint. The end of a sprint is marked by deliverable code: a working piece of software. Agile processes rely on a “release early, release often” approach, so interested users can participate in early evaluation and testing of the emerging product, giving feedback to the team. Teams use “burndown” charts to show progress through the sprint, and often use a lightweight management tool, such as Trello or monday.com, and a collaboration tool, such as Slack or Mattermost, for communication and coordination.

Along with the change in process from waterfall to agile, the process for gathering requirements changed, with a much greater focus on “pain points” identified through

multiple interviews with potential users and customers. Blank introduced the concept of Customer Discovery as the first of his “four steps to the epiphany [4].” Informally, the idea is for a product designer to “get out of the building”, i.e., away from the IT department and into the user’s world, ideally in the context of where users are experiencing their pain points. That increases the likelihood that a proposed solution, i.e., a system, will address a significant user issue, and that the user will be more likely to adopt the resulting solution. This concept is also central to the Rapid Contextual Design process [5].

2.2 User Stories

The Customer Discovery process normally proceeds through interviews with people who are seeking solutions to specific problems. For example, the task of buying a new vehicle has multiple pain points, including: 1) the difficulty of creating a short list of candidate vehicles from the large number of options in the market; 2) the effort needed to do research and to visit dealerships to make a choice; 3) the negotiation over the purchase or lease price, and 4) disposing of the owner’s old vehicle. (There are other pain points associated with buying a used vehicle.)

The above list provides a foundation for selecting one or more problems to be addressed in a solution, i.e., a system. The planned features for the system can be addressed in short, structured phrases, known as user stories, that describe them from the perspective of the user or customer of the system. For example, some user stories might include: 1) as a user, I want to be able to specify a type of vehicle and see a list of makes and models of that type; 2) as a user, I want to be able to further refine that list to those in a specified price range, and; 3) as a user, I want to create tables and charts that help me compare the refined list.

The product developer presents these user stories to potential users, including those who were initially interviewed, along with others having domain expertise and the ability to validate the ideas or suggest modifications, and other stakeholders. Obtaining these different viewpoints may lead to adjustments in the user stories and thus makes it more likely that a resulting system will satisfactorily address the problems identified in the interview process.

This activity of gathering of user stories is particularly effective for the development of customer-facing applications, where human users with interact with the application.

2.3 User/Developer Communication

Communication between the development team and the various stakeholders is a central issue in gathering system requirements and designing an application. In the early days of information system development, this communication usually took the form of a single lengthy document created at the end of “systems analysis”. Many of the information system design methods described in the CRIS Conference papers used complex graphical notations to illustrate the intended system behavior. These notations tended to emphasize the data model associated with the system over the user-centered functionality to be performed by the system. That approach addressed the needs of the

system developer and the data modeler, but made it difficult for a potential user to comprehend the planned behavior of the system and thus to suggest changes prior to system development. Agile development, by contrast, emphasizes user-centered communication mechanisms, and makes data modeling an iterative process [6], where each sprint includes evolution of the data model.

One such communication technique is storyboarding, similar to that used by screenwriters and other story tellers, which shows user scenarios in a format resembling comic strips. One popular approach uses two such storyboards for each user story, creating two scenarios, one representing how a process works currently, and the other showing how the process would work in a proposed system. In that way, a potential user can see the surrounding context for each user story in the system. There are numerous tools available for this task, including StoryboardThat [7].

Another approach to user/developer communication is through use case diagrams, originally developed by Ivar Jacobson for his object-oriented software engineering method ObjectOry [8], also included in the Unified Modeling Language (UML) [9]. A use case diagram presents a high level view of a system, bounding its scope and showing the actors and the activities to be performed by the system. The diagram itself doesn't show any control flow or the detailed steps of each activity, which are elaborated later.

Figure 1 shows a use case diagram for an online library system, as created by the developers of the Visual Paradigm modeling and management tools [10]. Without going into detail, one can see Borrower and Librarian actors, a Maintenance subsystem, and a total of 10 different activities. The diagram is well-suited for discussion between a development team and those looking for a solution, i.e., librarians and borrowers.

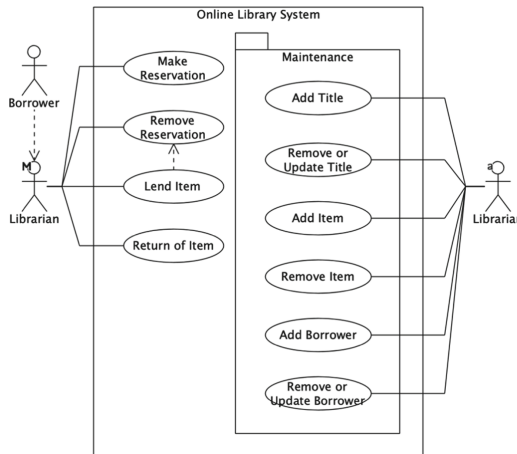


Fig. 1. High level use case diagram for online library system

Use case diagrams and user stories serve a similar purpose. For example, the information in the use case diagram could be addressed by ten user stories, such as: 1) as a Borrower, I want to make a reservation for an Item, and; 2) as a Librarian, I want to

add a Title to the system. In both cases, these descriptions serve as mechanisms for communication between the development team and its stakeholders. The team would receive feedback about any problems with the concept, and make any needed adjustments. (There are several problems in the model of Fig. 1.)

The high level descriptions are just a starting point. Each user story or activity must be elaborated, i.e., described in more detail, to show its associated steps. For example, the steps for Lend Item might include: 1) validate Borrower; 2) assign Item to Borrower's record, and; 3) determine the due date for the Item. Those steps enumerate the "normal" operation for that story. Beyond that, it's also necessary to enumerate the exceptional conditions, i.e., the conditions that would prevent the normal operation, and the business rules governing that use case.

Note that the identification and elaboration of the user stories is iterative and does not require a complete description of their steps or the exceptional cases before proceeding. It's not unusual for agile developers to implement the normal case first and return to complete the implementation later. Also, it's common to select a subset of the user stories to be implemented during a specific sprint. In this case, the first sprint might focus on the Maintenance subsystem, since it contains many of the basic activities needed to provide services to Borrowers.

2.4 Business Rules

In following an agile approach, the requirements are typically developed iteratively by working with likely users, customers and experts on the application domain. They have the knowledge about constraints on data values, relationships among data types, industry or government regulations, conditions that trigger associated events, the business process to be implemented by the system, and other relevant information. For example, the rules associated with "validate Borrower" above might include: 1) checking to see that the Borrower's library card has not expired, 2) preventing a Borrower from having more than 10 items checked out at a time, and; 3) preventing a Borrower with unpaid fines to check out any items.

Software developers rarely have such a detailed level of domain knowledge, but proper operation of the resulting system makes it essential to capture such information in the code or in a business rules "engine" that separates the rules from the program logic. Many of these rules emerge from the elaboration of user stories or use cases.

2.5 User Interface Prototyping

An alternative approach to communication between developers and stakeholders is to develop mockups (or live) versions of the intended user interface for the system. Some people gain a better understanding of the features of a planned system by seeing how they will use it. The mockup is designed to show the layout and content of screens, whether for a standalone application, a web application, or a mobile app, and letting potential users view it and possibly work directly with it to see how it would be used in practice. With the appropriate tools, it is possible to design and modify these user interfaces very quickly so that the designer can be highly responsive to user requests and easily demonstrate the planned user experience.

Building these mockups requires the designer not only to have an appropriate tool, but also an understanding of the features to be provided to the users, with the expected inputs and outputs. To do that, the user must usually work through the use cases or user stories and then manually transform that understanding into the user interface design. One effective technique maps a user story into a user command, which may be implemented by text input, menu selection, gesture, or voice input.

In effect, the user interface mockup becomes an alternative approach to requirements gathering and validation. Some users get a better sense of the system features by working with such mockups rather than with a set of user stories or use case diagrams; there are many tools available for building such prototypes, including Balsamiq Wireframes [11]. That approach is central to the author's User Software Engineering methodology, as presented at the CRIS conference [1, 12].

There are some risks in proceeding with the user interface mockups and bypassing the step of validating user stories with stakeholders. Users of the mockups may have an incorrect impression of progress, thinking that the application is largely built when only the proposed façade has been designed. They may also be less able to find errors or omissions in the overall system. Balsamiq's product helps to overcome this problem, since their wireframes are lower fidelity rather than highly precise.

3 An Agile Approach to the CRIS Conference Problem

The CRIS Conference Problem, as originally stated, is reproduced in the Appendix. Because of space limitations, we will give primary attention to the stories associated with the Program Committee. This approach is in keeping with an agile process, where the stories of the Organizing Committee are addressed in later sprints.

3.1 User Stories

Working from the problem statement leads to the enumeration of the following user stories:

- 1) As a Program Committee Chair, I will prepare a list of people to whom the call for papers is to be sent;
- 2) As a Program Committee Chair, I will record the letters of intent from people intending to submit a paper;
- 3) As a Program Committee Chair, I will register the papers when they are received;
- 4) As a Program Committee Chair, I will distribute the papers among the referees;
- 5) As a Program Committee Chair, I will collect the referees' reports and select the papers accepted for the conference program, and;
- 6) As a Program Committee Chair, I will group the selected papers into sessions for presentation, selecting a chair for each session.

A knowledgeable user would quickly find some important gaps in the problem statement. These include failure to create a Call for Papers, failure to invite people to serve as referees, failure to establish a deadline for referees to return their reviews, failure to identify a decision process when too few reviews are received, absence of a

process for notifying authors about the acceptance or rejection of their papers, failure to establish a deadline for submission of the camera-ready copies of the accepted papers, and more. The problem statement refers to referees, with the implicit (and possibly erroneous) assumption that they constitute the Program Committee. Each of these gaps leads to the creation of new actors, user stories, and possibly to modifications of the existing user stories. This, we might add more user stories:

- 7) As a Program Committee Chair, I will invite people to serve on the Program Committee and be referees for the submitted papers.
- 8) As a Program Committee Chair, I will prepare a Call for Papers and ask Program Committee Members to review it prior to sending it out.
- 9) As a Program Committee Member, I will prepare reviews of papers sent to me by the Program Committee Chair and return those reviews before the review deadline.
- 10) As a Program Committee Chair, I will notify authors about the acceptance or rejection of their submitted paper(s), and provide them with copies of the reviews.
- 11) As a Program Committee Chair, I will provide authors of accepted papers with information about the procedure for submitting the camera-ready version of their paper.

It's also possible to view some business rules from this list. For example, the system needs a rule to avoid conflicts of interest, making sure that referees are not assigned a paper that they or a close colleague submitted. It's also likely that rules are needed about the maximum length for an accepted paper, the number of papers in a session, and many other constraints and rules.

In this way, it is possible for user stories to serve as the basis for creating a clear, complete, and consistent set of requirements, as well as to identify serious errors in the original problem statement. Even the most rudimentary effort to validate the original problem statement with domain experts and prospective users would easily highlight these gaps, business rules, and missing user stories, long before anyone built complex system models or wrote any code. Because of ongoing interaction with users, the agile approach works better for user-centered development than any of the methods, including this author's, described by the 13 methods in the original CRIS Conference, which collectively failed to find any of the problems so quickly identified here.

3.2 Use Case Diagrams and Their Elaboration

While user stories are a highly effective mechanism for user-centered design, they are not intended to capture system behavior, data produced and consumed by the system, or all of the interactions among the actors and system. A use case diagram can address these issues, as well as illustrating the scope of the system.

Figure 2 shows an incomplete use case model for the IFIP Working Conference, reflecting the user stories shown above and some system actions needed for basic operation of the system, such as providing login/logout capabilities for users, who may be members of the Organising and Program Committees, authors, and members of the sponsoring Working Group(s) and Technical Committee(s). This diagram may be changed frequently, as the application requirements are better understood. Note that these operations, as with the user stories, are shown at a high level of abstraction.



Fig. 2. Preliminary use case model for IFIP working conference problem

Each of these use cases needs further elaboration so that it can be validated and subsequently implemented. For example, the use case “Send review to Program Chair” might be elaborated as:

1. Fill in standard review form with referee name and scores for quality, originality, and reviewer knowledge of subject area.
2. Write comments about the paper to be transmitted to the author by the Program Chair
3. Write optional comments about the paper to be seen only by the Program Chair.
4. Recommend whether the paper should be accepted or rejected.
5. Transmit the completed review to the Program Chair.

While the first implementation of this use case for a Minimal Viable Product [13] may address only the “normal” case, it must eventually address the alternative or exceptional cases, such as:

1. The referee is unable to complete the review before the deadline and does not submit a review.
2. The referee asks a colleague to do the review in their place.
3. The referee is unfamiliar with the subject matter of the paper and is unable to prepare a review.

Such exceptional conditions are typically uncovered through the same process of interviews that provides the normal flow of the use case.

3.3 Evolving the Data Model

In earlier solutions to the IFIP Working Conference problem [1], database schema definition played a key role in the early stage of development. With an agile approach, however, the data model is iteratively derived from the use cases and their elaboration, as noted in [6]. Most of the earlier models were relational or entity-relationship, but the advent of NoSQL database management systems [14] provides new ways of storing and managing large volumes of unstructured data. It is much easier to modify the model of unstructured data as developers address each new sprint.

Switching the perspective from the relational approach to the non-relational approach provides the opportunity to store data according to its logical meaning without having to transform the various objects into a set of tables with interrelationships. In that way, the use case models and/or the user stories yield a set of objects that can be stored directly and whose properties can be defined.

Thus, the use case models refer to objects such as authors, invitees, users, letters of intent, papers, reviews, and sessions. For each of those objects, the requirements gathering process provides information about their properties:

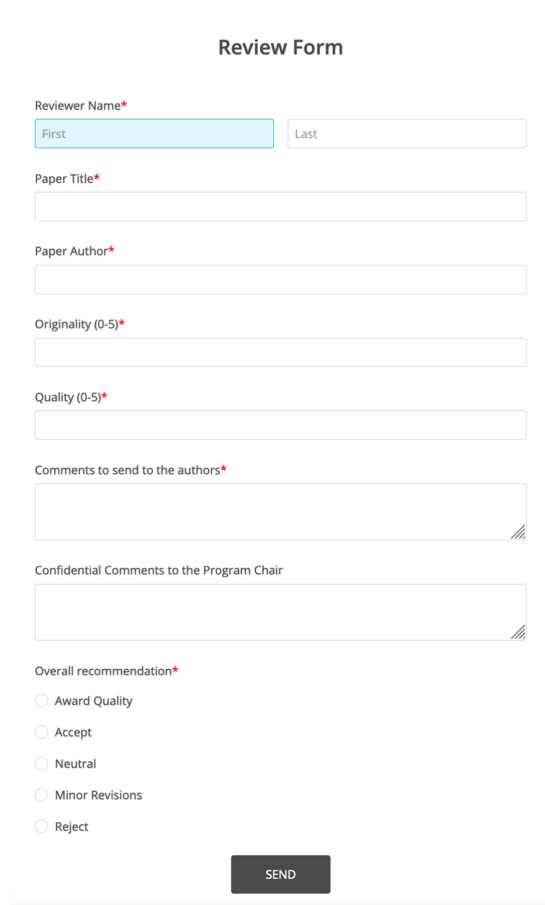
```
Paper (title, abstract, keywords, authors, corresponding_author,
email, paper_body, reviews, decision)
Session (title, location, date_and_time, length, chair, papers)
```

Once again, these definitions will be refined over time as more is learned about the application and its exceptional situations. For example, it's possible that the Working Conference will have invited papers as well as submitted ones, where the invited papers are automatically accepted without review and scheduled, often in a specially designated session.

3.4 User Interface Mockups

As noted in Sect. 2.5 above, preliminary designs of the user interface are an effective complementary technique for identifying requirements, as well as assuring the usability of the completed application. For example, as shown in Fig. 3 using 123formbuilder.com [15], it's easy to build a "live" high-fidelity mockup of the referee

review form that can be tried and reviewed by a user. Many user suggestions can be quickly implemented so that the user can iterate on the design, including doing a/b experimentation of alternatives [16].



The image shows a web form titled "Review Form". It contains the following fields and options:

- Reviewer Name***: Two input fields labeled "First" and "Last".
- Paper Title***: A single-line text input field.
- Paper Author***: A single-line text input field.
- Originality (0-5)***: A single-line text input field.
- Quality (0-5)***: A single-line text input field.
- Comments to send to the authors***: A multi-line text area with a diagonal slash icon in the bottom right corner.
- Confidential Comments to the Program Chair**: A multi-line text area with a diagonal slash icon in the bottom right corner.
- Overall recommendation***: A list of radio button options:
 - Award Quality
 - Accept
 - Neutral
 - Minor Revisions
 - Reject
- SEND**: A dark grey rectangular button.

Fig. 3. Mockup of a referee review form

There are also numerous tools for building mockups for mobile devices, as needed for mobile app design. Figure 4 shows a preliminary version of a mockup using Balsamiq.

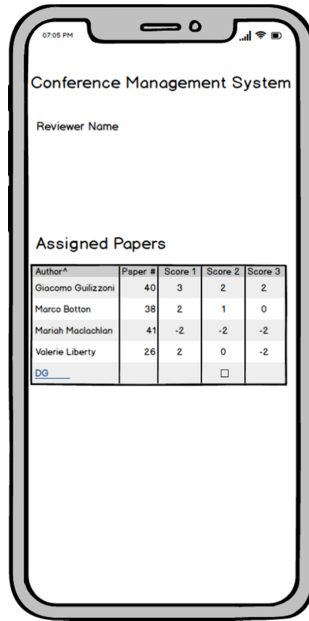


Fig. 4. Mockup of reviewer screen with Balsamiq

3.5 Agile Development with Scrum

As the Product Owner [17] addresses the requirements in the problem statement and reviews them with domain experts and potential users, it’s possible to begin implementing the application itself. Following the basic concepts of Scrum described in Sect. 2.1 above, the Product Owner and the Scrum team would start by selecting a set of user stories or use cases for the first sprint. Each sprint would have a backlog of tasks to be completed within the time allocated for the sprint.

One possible approach in the IFIP Working Conference example is to group the Program Committee activities into four categories:

- 1) Set up basic system infrastructure: user management, establish program committee, manage author submissions (letters of intent and papers);
- 2) Processing of submitted papers: logging submissions, assigning papers to referees, collecting completed reviews, deciding which papers to accept;
- 3) Author notifications: send letters of acceptance and rejection, send publication instructions to corresponding authors of accepted papers, and;
- 4) Session formation: define session titles, appoint session chairs, assign time and location, assign papers to session.

Each of these stories would be refined as needed, with the Product Owner working with domain experts to find and resolve issues. The user stories associated with the activities of the Conference Organizing Committee could be developed in subsequent sprints.

It's long been known that early detection of requirements errors can provide major cost savings over detection of problems later in the process or after application deployment. Even with today's tools for continuous integration and continuous delivery (CI/CD) [18], early error detection can reduce the number of bug fixes and allow the developers to devote more time to enhancements and new features.

With this decomposition of the conference system implementation, the system can be delivered in small pieces. By the end of the second sprint (Processing of submitted papers), there should be enough functionality to release it on a limited basis as a Minimum Viable Product, with the remaining sprints adding valuable capabilities while building out the robustness of the system and addressing the exceptional conditions that were omitted earlier.

4 Conclusion

Revisiting the IFIP Working Conference example after almost forty years shows how greatly information system design methodologies have changed in the interim, driven by agile methods, the Internet and the World Wide Web, as well as by huge advances in computing hardware, networking, and displays. Today's ubiquitous personal computers had just entered commercial use in 1982, and mobile "smartphone" applications were nearly two decades away. Many of today's most popular applications could barely be conceived, let alone implemented, in that earlier era.

The growth of open source software has also made a substantial contribution to rapid development of information systems and other applications. As one example, it's possible to use an off-the-shelf event management system, the open source Conference Organizing Distribution [19], built on the open source Drupal content management system [20], that comes very close to providing all of the features (plus some others) of the problem statement for the IFIP Working Conference. With access to the source code, one could consider modifying that code for the IFIP Working Conference system rather than going through the agile process shown here.

The original CRIS Conference deserves credit for bringing greater attention to the early stages of the design process. Today's methodologies for system design place even greater emphasis on meeting user requirements, as well as assuring a positive user experience with the application. Those aspects of information system design will certainly endure through future generations, even as new technologies emerge.

Appendix – Problem Definition

Here is the original problem definition that was used in the call for submissions to the CRIS conference.

1. Background

An IFIP Working Conference is an international conference intended to bring together experts from all IFIP countries to discuss some technical topic of specific interest to one or more IFIP Working Groups. The usual procedure, and that to be considered for the present purposes, is an invited conference which is not open to everyone. For such conferences, it is something of a problem to insure that members of the involved Working Group(s) and Technical Committee(s) are invited even if they do not come. Furthermore, it is important to ensure that sufficient people attend the conference so that the financial break-even point is reached without exceeding the maximum dictated by the facilities available.

IFIP Policy on Working Conferences suggest the appointment of a Program Committee to deal with the technical content of the conference and an Organising Committee to handle financial matters, local arrangements, and invitations and/or publicity. These committees clearly need to work together closely and have a need for common information and to keep their recorded information consistent and up to date.

2. Information System to be designed

The information system which is to be designed is that necessary to support the activities of both a Program Committee and an Organising Committee involved in arranging an IFIP Working Conference. The involvement of the two committees is seen as analogous to two organisational entities within a corporate structure using some common information.

The following activities of the committees should be supported.

Program Committee

1. Preparing a list to whom the call for papers is to be sent.
2. Registering the letters of intent received in response to the call.
3. Registering the contributed papers on receipt.
4. Distributing the papers among those undertaking the refereeing.
5. Collecting the referees' reports and selecting the papers for inclusion in the program.
6. Grouping selected papers into sessions for presentation and selecting chairman for each session.

Organising Committee

1. Preparing a list of people to invite to the conference.
2. Issuing priority invitations to National Representatives, Working Group members, and members of associated working groups.
3. Ensuring all authors of each selected paper receive an invitation.
4. Ensuring authors of rejected papers receive an invitation.
5. Avoiding sending duplicate invitations to any individual.
6. Registering acceptance of invitations.
7. Generating final list of attendees.

3. Boundaries of System

It should be noted that budgeting and financial aspects of the Organising Committee's work, meeting plans of both committees, hotel accommodation for attendees and the matter of preparing camera ready copy of the proceedings have been omitted from this exercise, although a submission may include some or all of these extra aspects if the authors feel so motivated.

References

1. Olle, T., Sol, H., Verrijn-Stuart, A.: Information Systems Design Methodologies: A Comparative Review. North-Holland, Amsterdam (1982)
2. Manifesto for Agile System Development. <https://agilemanifesto.org>. Accessed 22 Nov 2020
3. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond (2004)
4. Blank, S.: Four Steps to the Epiphany, 2nd edn. K&S Ranch, Uvalda (2013)
5. Holtzblatt, K., Wendell, J., Wood, S.: Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design. Morgan Kaufman, San Francisco (2004)
6. Agile/Evolutionary Data Modeling: From Domain Modeling to Physical Modeling. <http://agiledata.org/essays/agileDataModeling.html>. Accessed 22 Nov 2020
7. Storyboard That: The World's Best Free Online Storyboard Creator. <https://www.storyboardthat.com>. Accessed 22 Nov 2020
8. Jacobson, I.: Object-Oriented Software Engineering. Addison-Wesley, Reading (1992)
9. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Reading (2004)
10. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. <https://www.visual-paradigm.com>. Accessed 22 Nov 2020
11. Balsamiq. Rapid, effective and fun wireframing software. <https://balsamiq.com>. Accessed 22 Nov 2020
12. Wasserman, A.: The user software engineering methodology: an overview. In: Information Systems Design Methodologies: A Comparative Review, pp. 591–628. North-Holland, Amsterdam (1982)
13. What is a Minimum Viable Product (MVP)? | Agile Alliance. <https://www.agilealliance.org/glossary/mvp>. Accessed 22 Nov 2020
14. No SQL Tutorial. <https://www.guru99.com/nosql-tutorial.html>. Accessed 22 Nov 2020
15. Online Form Builder with Drag & Drop. <https://www.123formbuilder.com>. Accessed 22 Nov 2020
16. Optimizely: The World's Leading Experimentation Platform. <https://optimizely.com>. Accessed 22 Nov 2020
17. What is a Product Owner? <https://www.scrum.org/resources/what-is-a-product-owner>. Accessed 22 Nov 2020
18. What is CI/CD? | Opensource.com. <https://opensource.com/article/18/8/what-cicd>. Accessed 22 Nov 2020
19. UseCod.com | Open Source Event Management Software. <https://usecod.com>. Accessed 22 Nov 2020
20. Drupal | Open Source CMS. <https://drupal.org>. Accessed 22 Nov 2020