

TASK AND INTERRUPTION MANAGEMENT
IN
ACTIVITY-CENTRIC COMPUTING

STEVEN JEURIS

PhD Thesis in Human-Computer Interaction

Submitted: July 2016

Published: March 2017

Steven Jeuris: *Task and Interruption Management in Activity-Centric Computing*, PhD Thesis in Human-Computer Interaction, © March 2017

INSTITUTE:

IT University of Copenhagen
Software Development Group
Pervasive Interaction Technology Laboratory

SUPERVISOR:

Professor Jakob E. Bardram

ABSTRACT

Throughout history, the design of interactive computing systems has always been inhibited by technological limitations of the time. For a truly groundbreaking paradigm shift to occur (reshaping the very nature of human-computer interaction), a tremendous amount of research and engineering is required. Therefore, most computing systems instead build on top of an existing stack of contemporaneous technologies, inescapably adhering to their underlying interaction paradigms. When left unquestioned, such an incremental approach inadvertently shoehorns system design into preexisting notions of how computing systems work. Thus, it is likely that design decisions imposed by technological constraints of the past have needlessly been carried over to modern-day systems. With information technology now forming a major part of our daily lives and giving rise to new emerging design challenges, it is prudent to address these not in isolation, but by fundamentally reevaluating the current computing paradigm.

To this end, activity-centric computing has been brought forward as an alternative computing paradigm, addressing the increasing strain put on modern-day computing systems. Activity-centric computing follows a top-down approach to design using the full context of human activity as the starting point of analysis. The focus no longer lies on individual technologies, but on how computing systems are used as mediators within the broader context of human intentionality, thus also taking into account the encompassing community, environment, and dependencies on other technologies. Users can aggregate resources, work, and collaborate on them within goal-oriented workspaces that are meaningful to the user, as opposed to having to adhere to data structures imposed by specific technologies. Such systems have been deployed successfully in a variety of different domains, including healthcare, experimental biology, and software engineering.

However, several recurring open issues have been identified based on the deployment and evaluation of different activity-centric computing systems. Broadly speaking these impact the scalability and intelligibility of current research prototypes. In this dissertation, I postulate that such issues arise due to a lack of support for the full set of practices which make up *activity management*. Most notably, although *task and interruption management* are an integral part of personal information management, they have thus far been neglected in prior activity-centric computing systems. Advancing the research agenda of activity-centric computing, I (1) implement and evaluate an

activity-centric desktop computing system, incorporating support for interruptions and long-term task management; (2) provide empirical data on the overhead of switching between activities when using contemporary desktop computing systems; and (3) implement a software architecture facilitating developers to aggregate resources handled by independent applications into one central activity manager.

PUBLICATIONS

Most of the work presented in this dissertation is the result of an elaborate review and exploration of activity-centric computing, also discussed in earlier publications which my colleagues and I have worked on. In sections where content of this earlier work has been incorporated, I will refer to the specific publications.

- [1] Jakob E. Bardram, *Steven Jeuris*, and *Steven Houben*. “Activity-based computing: computational management of activities reflecting human intention.” In: *AI Magazine* 36.2 (2015), pp. 63–72. DOI: [10.1609/aimag.v36i2.2585](https://doi.org/10.1609/aimag.v36i2.2585).
- [2] *Steven Jeuris* and Jakob E. Bardram. “Dedicated workspaces: Faster resumption times and reduced cognitive load in sequential multitasking.” In: *Computers in Human Behavior* 62 (2016), pp. 404–414. ISSN: 0747-5632. DOI: <http://dx.doi.org/10.1016/j.chb.2016.03.059>. URL: <http://www.sciencedirect.com/science/article/pii/S0747563216302308>.
- [3] *Steven Jeuris* and *Steven Houben*. “Temporal Model for Reflective Multitasking.” In: *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '13. 2013.
- [4] *Steven Jeuris*, *Steven Houben*, and Jakob E. Bardram. “Laevo: A Temporal Desktop Interface for Integrated Knowledge Work.” In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14. Honolulu, Hawaii, USA: ACM, 2014, pp. 679–688. ISBN: 978-1-4503-3069-5. DOI: [10.1145/2642918.2647391](https://doi.org/10.1145/2642918.2647391). URL: <http://doi.acm.org/10.1145/2642918.2647391>.
- [5] *Steven Jeuris*, *Paolo Tell*, and Jakob E. Bardram. *co-Laevo: Supporting Cooperating Teams by Working 'within' Shared Activity Time Lines*. Tech. rep. ITU-TR-2016-193. IT University of Copenhagen, June 2016. URL: <http://en.itu.dk/Research/About-ITUs-Research/Technical-Reports/Technical-Reports-Archive/2016/TR-2016-193>.
- [6] *Steven Jeuris*, *Paolo Tell*, *Steven Houben*, and Jakob E. Bardram. “The Hidden Cost of Task Switching: How Well Do Window Managers Support Sequential Multitasking?” In: In submission.

*Don't repeat yourself (DRY):
Every piece of knowledge must have a single,
unambiguous, authoritative representation within a system.*
— Andrew Hunt and David Thomas [130]

ACKNOWLEDGMENTS

While writing these acknowledgments, I am faced with an unusual conundrum. As a software engineer, I have learned to resent unnecessary repetition¹, but the only way to fully express my gratitude to all those I am indebted to, will inevitably require me to ‘repeat myself’. Still, in the spirit of this profession, and to reduce the risk of leaving anyone out (which I undoubtedly will), allow me to resolve this dilemma in the most *generic* way I know: I want to start out by thanking everyone that has inspired me in this line of research, has supported me throughout the process of working on this dissertation, and will continue to be there for me throughout the rest of my career. My apologies lest you have been left out.

First and foremost, not only would this dissertation have been impossible without the support of my advisor—Jakob E. Bardram—the very topic, activity-centric computing, would likely have dwindled out of existence, if not for his sustained commitment to this line of research. It takes character to keep pursuing a vision which by many within the field is considered to be worn-out. In addition, you have created a vibrant research environment within which I could perform my work, and it was sad to see you leave the Pervasive Interaction Technology Laboratory (PIT lab) behind. However, it is a testimony of your perseverance to study topics you truly care about that you became the director of the Copenhagen Center for Health Technology (CACHET), thereby returning to your roots. Thank you for your professional guidance, and giving me the opportunity to work on this research!

Second, I would like to thank all colleagues from the PIT lab for the many shared discussions we had. As in Steve Job’s ‘parable of the stones’ ([Appendix A](#)) it is “through that group of incredibly talented people bumping up against each other, having arguments, having fights sometimes, making some noise” that the ideas presented here came to fruition. In particular, I am indebted to the co-authors on my publications—Steven Houben and Paolo Tell—who directly helped shape this dissertation, and with whom I shared many inspiring design sessions, in the lab and bars alike. Special thanks to Steven

¹ Dave Thomas: “Most people take don’t repeat yourself (DRY) to mean you shouldn’t duplicate code. That’s not its intention. The idea behind [DRY](#) is far grander than that.” (<http://www.artima.com/intv/dry.html>)

Houben for inviting me to the PIT lab, which is how I got interested in pursuing this PhD. The past few years, I have also really enjoyed the company of fellow PhD students Morten Esbensen and Shahram Jalalinia, during summer schools, seminars, and conference trips. I would also like to explicitly thank Florian Biermann, Jacob Cholewa, Sebastian Büttrich, Thomas Pederson, John Paulin Hansen, Tim Hankins, Dominik Grondziowski, Siemen Baader, Mathias Schmidt, and Mathias Pedersen for their active presence in the lab, thereby creating a fun and engaging work environment.

Third, I would also like to restate my gratitude to my former supervisors of my master thesis—Peter Werkhoven and Ingrid van Zaanen—who enabled me to commence early work in this line of research and thereby paved the way for future work. Also the participants during the empirical studies presented here are worthy of mention, without whom the results would be rather succinct.

Lastly, without the continued support of family and friends, both in Belgium and in Denmark, I likely would not have been able to stay motivated (or well-fed) to turn this dissertation into something I can truly be proud of.



IT UNIVERSITY OF COPENHAGEN

This research has been funded primarily by the Danish Agency for Science, Technology and Innovation under the project “Next Generation Technology for Global Software Development”, #10-092313 and the collaboration with Steven Houben by the EU Marie Curie Network iCareNet under grant number 264738.



CONTENTS

1	INTRODUCTION	1
1.1	Research question	4
1.2	Research method	5
1.3	Overview of contributions	7
I	A DESIGN SPACE FOR ACTIVITY-CENTRIC COMPUTING	9
2	COMPUTER-MEDIATED ACTIVITY	11
2.1	History of activity-centric computing	13
2.2	The search for theory	16
2.3	Implications for design	18
2.4	Conceptual models	19
2.5	Motor themes	21
3	INTERACTION FRAMEWORK	25
3.1	Common language	26
3.2	Layers of abstraction	27
3.3	Types of interaction	33
3.4	Completing the mosaic	35
4	ACTIVITY MANAGEMENT	39
4.1	Information fragmentation	41
4.2	Computational activities	43
II	DESIGN AND TECHNOLOGY	47
5	ACTIVITY-CENTRIC COMPUTING SYSTEMS	49
5.1	Task, window, and file management	50
5.2	Desktop systems	53
5.3	Ubiquitous computing systems	57
6	LAEVO AND CO-LAEVO	59
6.1	Activity life cycle	60
6.2	Personal information management	62
6.3	The activity time line	65
6.4	To-do list and interruptions	68
6.5	The cooperative activity life cycle	69
6.6	Shared activity hierarchies	71
7	DEDICATED WORKSPACES TOOLKIT	77
7.1	Architecture	79
7.2	Workspace manager	80
7.3	Plug-in manager	82
III	EMPIRICAL STUDIES	87
8	LAEVO EVALUATION	89
9	TASK SWITCHING IN SEQUENTIAL MULTITASKING	93

9.1	Multitasking continuum	94
9.2	Tasks and task sequence	97
9.3	Study 1: comparative study	100
9.4	Study 2: in-depth analysis	108
9.5	Task resumption and construction time	121
9.6	Cognitive load and performance	125
9.7	Threats to validity	126
IV	DISCUSSION AND CONCLUSION	129
10	SCALABILITY AND INTELLIGIBILITY	131
10.1	The hidden cost of task switching	131
10.2	Integrated knowledge work	133
10.3	The marks are on the knowledge worker	135
11	FROM COMPUTATION TO ACTIVITY	139
11.1	Limitations: radical innovation	142
11.2	Future work: a long-term goal	144
V	APPENDIX	147
A	THE PARABLE OF THE STONES	149
B	LAEVO EVALUATION MATERIAL	151
B.1	Laevo manual	151
B.2	Laevo diary study questions	159
	BIBLIOGRAPHY	161

LIST OF FIGURES

Figure 1	The first graphical window management system, part of the Smalltalk programming environment.	2
Figure 2	Triangulation model for human-computer interaction.	5
Figure 3	An overview of the main activities and deliverables discussed in this dissertation.	6
Figure 4	A coarse depiction of the expansion of user interface design and associated scientific disciplines over the course of history.	12
Figure 5	Activity system model: the unit of analysis in activity theory.	13
Figure 6	An overview of workspaces in the Rooms system.	14
Figure 7	Conceptual model: the designer’s model, the system image, and the user’s model.	19
Figure 8	The Xerox Star user interface which commercialized the desktop metaphor.	21
Figure 9	Strategic diagram for CHI indicating a lack of motor themes.	22
Figure 10	Two dimensions for a theoretical framework in human-computer interaction.	25
Figure 11	Individual sciences operating on their own plane of knowledge, and the interaction between them.	26
Figure 12	Using instruments, additive and subtractive techniques are used in order to shape objects.	31
Figure 13	A character (‘data material’) is converted into pixels (‘resource’).	31
Figure 14	‘Subject’, ‘object’, and ‘outcome’ in activity theory overlap to some degree with the activity abstraction dimension.	34
Figure 15	Overview of task, window, and file management, highlighting conflicts which arise between them.	53
Figure 16	TaskTracer for Windows XP.	54
Figure 17	Giornata for OS X.	54
Figure 18	A historical overview of activity-centric computing since 2003	55
Figure 19	co-ActivityManager: an activity-centric desktop computing system supporting communication and collaboration	56

Figure 20	ReticularSpaces is an activity-centric smart space environment.	57
Figure 21	ActivitySpace supports moving and sharing resources across multiple devices.	58
Figure 22	Modern knowledge work consists of archiving, multitasking and planning. Four fundamental practices, related to these processes, determine how an activity evolves over time.	60
Figure 23	Laevo is a <i>temporal</i> activity-centric desktop interface supporting activity management.	63
Figure 24	The activity context library and tray icon within a dedicated activity workspace.	64
Figure 25	The activity overview through which activities are accessed and managed in Laevo.	66
Figure 26	Interruptions and to-do items in Laevo.	68
Figure 27	Implications for design in cooperative activity life cycle management.	69
Figure 28	An example activity tree for a PhD student.	72
Figure 29	An overview of the newly introduced cooperative features of co-Laevo.	73
Figure 30	User profile and activity access in co-Laevo.	74
Figure 31	The dedicated workspaces toolkit architecture.	79
Figure 32	Class diagram for WorkspaceManager.	80
Figure 33	Example of one concrete AbstractWorkspaceManager, the VirtualDesktopManager.	81
Figure 34	Suspending a workspace in a more recent version of Laevo.	81
Figure 35	VirtualDesktopManager plug-ins which are managed by PluginManager.	83
Figure 36	Different activity scopes in Laevo.	91
Figure 37	Multiple instances of one activity in time in co-Laevo.	92
Figure 38	The disengagement and resumption stage during a task switch.	97
Figure 39	The four tasks used during the sequential multitasking studies.	99
Figure 40	Task sequence followed in the sequential multitasking studies.	100
Figure 41	Laevo during the comparative sequential multitasking study.	102
Figure 42	Traditional desktop environment during the comparative sequential multitasking study.	103
Figure 43	Overview of average resumption time per individual task switch under both conditions.	105
Figure 44	Overview of average resumption times per participant under both conditions.	105

Figure 45 Cumulative task switch time for ten participants during the comparative sequential multitasking study. 106

Figure 46 Breakdown of averages for each scale used as part of the Raw TLX test in the comparative sequential multitasking study. 107

Figure 47 Window manager features of Windows 7. . . . 110

Figure 48 Data analysis of task switches in ChronoViz. . 113

Figure 49 Average disengagement and resumption time per participant. 114

Figure 50 Percentile breakdown of actions and intents during task switches per participant, for both the disengagement and resumption stage. 116

Figure 51 The average percentage of time spent during task switching on reorganizing the workspace. 117

Figure 52 Detailed breakdown of representative task switches for recurring observations. 119

Figure 53 Box plots for task resumption times under both studies. 121

Figure 54 Average breakdown of reorganization actions during task resumption. 122

Figure 55 Average breakdown of reorganization actions during task disengagement. 124

Figure 56 Box plots for Raw TLX measures under both studies. 125

Figure 57 Information overload experienced during task switching using the taskbar. 133

Figure 58 Real-world use of Laevo by the author of this dissertation. 134

Figure 59 A lack of intelligibility can lead to a discrepancy between context and intent. 136

Figure 60 The activity life cycle is supported in Laevo through the management of activities on a time line and by constructing work within the context of dedicated workspaces. 141

Figure 61 Incremental innovation compared to radical innovation 143

Figure 62 The design of NLS supported dedicated workspaces.146

LIST OF TABLES

Table 1	Examples of objects that are part of human activity along different layers of abstraction. . . .	30
Table 2	Decomposition of the interaction framework into 28 different research topics.	35
Table 3	Examples of relevant research topics for each of the categories within the interaction framework.	36
Table 4	Different categories of computing systems positioned within the interaction framework. . .	42
Table 5	The activity-centric computing principles represented within the interaction framework. . .	43
Table 6	The full range of interactions activity-centric computing should address.	44
Table 7	Task, window, and file management are an integral part of knowledge work, generally supported by independent tools.	50
Table 8	Laevo and co-Laevo integrate workspaces with task and workflow management respectively, which as a whole constitutes activity management.	59
Table 9	Practices which influence the activity life cycle, positioned within the interaction framework. .	61
Table 10	The dedicated workspaces toolkit is an intermediate approach to supporting the construction of arbitrary workspaces.	78
Table 11	The 'activity-centered' and 'multiplexing' principles of activity-centric computing cover support for sequential multitasking through the use of dedicated workspaces.	93
Table 12	Counterbalancing in the comparative sequential multitasking study.	102
Table 13	Overview of <i>t</i> -tests comparing productivity and accuracy of all tasks between dedicated workspaces and a traditional desktop environment.	108
Table 14	Overview of <i>t</i> -tests comparing productivity and accuracy of all task between the first session and the second session participants took part in.	108
Table 15	The questionnaire assessing computer literacy and degree of multitasking users engage in on average.	110
Table 16	Framework used to analyze task switches. . . .	112

Table 17	Time and percentage of total task switch time spent on errors for each participant.	115
Table 18	Breakdown of averages for each scale used as part of the Raw TLX test in the in-depth study on sequential multitasking.	115
Table 19	The makeup of reorganization work during disengagement and resumption.	117
Table 20	Features used for ‘navigation to opened’. . . .	120
Table 21	Predicted time spent on reorganizing the workspace per day for participants in the second study on sequential multitasking.	132
Table 22	The current activity-centric computing principles outlined within the interaction framework. This highlights how Laevo and co-Laevo target previously unaddressed issues of information fragmentation.	140
Table 23	Historically, computational support for different levels of abstraction has not always been separated.	146

LISTINGS

Listing 1	Setting up a WorkspaceManager.	81
Listing 2	Settings for Chrome, which has a status bar as separate window.	83
Listing 3	The persistence provider for Notepad.	84

ACRONYMS

ABC	activity-based computing
AI	artificial intelligence
API	application programming interface
CHI	Conference on Human Factors in Computing Systems
CSCW	computer-supported cooperative work
DRY	don't repeat yourself
DW	dedicated workspaces
ERP	enterprise resource planning
GSD	global software development
GUI	graphical user interface
HCI	human-computer interaction
IDE	integrated development environment
IPC	inter-process communication
IS	information systems
MEF	Managed Extensibility Framework
OOP	object-oriented programming
PIM	personal information management
UIST	User Interface Software and Technology
UX	user experience
VDM	virtual desktop manager

INTRODUCTION

Applications: what a terrible term. What a terrible concept. Applications have little to do with the tasks that people are attempting to accomplish. Look. We don't do word processing; we write letters, or memos, or reports, or notes to ourselves.

— Donald A. Norman [101]

Sketchpad [127], the first interactive computing system supporting direct manipulation of graphical data, dates back to the early sixties. It paved the way for a new area of research originating as a sub-branch of computer science, called human-computer interaction (HCI)—the topic of this thesis. Aside from inciting the interest of researchers to design novel user interactions, its implementation also inspired the design of the currently predominant programming paradigm [75], object-oriented programming (OOP). Thus, early on in the history of computer science the design of graphical user interfaces (GUIs) became interlinked with the development of the required underlying technologies. In fact, the concept of a graphical window management system (still in use by modern desktop computing systems) was first introduced in the programming environment of Smalltalk [129] (Figure 1). This demonstrates how much the design of user interfaces relies on, and is influenced by underlying architecture [42]. It should then not come as a surprise that many abstractions and interactions exposed in contemporary user interfaces reflect the underlying hardware and software architecture upon which they have been built.

This is a suboptimal approach to user interface design. For user interfaces to be intuitive, they should reflect a preexisting conceptual understanding users have of the world, as opposed to being bound to arbitrary technological abstractions. Ironically, one of the most prevalent concepts in computing has no clear conceptual counterpart in everyday life: *applications*. An application first and foremost represents the short-lived execution of predefined routines, operating on data loaded into the working memory of a computer. In contrast to tools within a physical environment, software applications introduce an artificial dependency between data and the tools that operate on it; i. e., data and tools can only be accessed in unison, from within a specific application. This is distinctly different from real world tools; e. g., a hammer serves multiple purposes, and to use it one does not need to be in the workshop where it is stored. Unfortunately, ‘application’ is but one example of an ill-suited technological abstraction which is still exposed to users in contemporary computing systems.

The design of interactive systems is influenced by underlying architecture.

Technological abstractions are unintuitive, yet common.

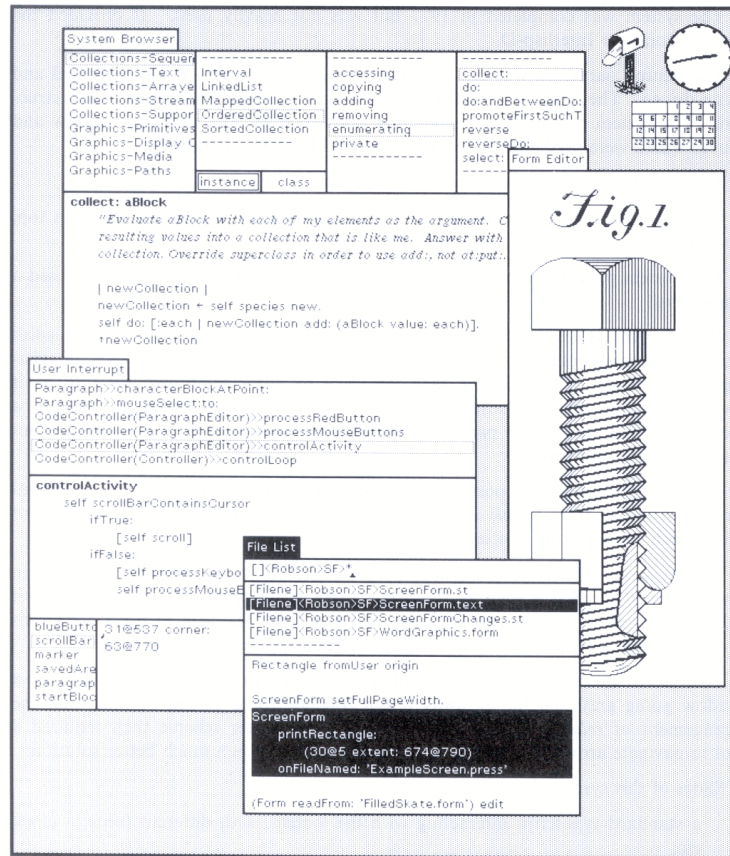


Figure 1: The first graphical window management system, introduced as part of the Smalltalk programming environment in the 1970s [129].

HCI is moving away from a technological perspective on design towards a contextual one.

As per the quote by Donald A. Norman at the start of this chapter, this is not a new observation. Many studies point towards the inadequacies of contemporary computing systems. Ever since Lucy Suchman reported on how office workers struggled to use an ‘expert help system’ attached to a large complex photocopier [126], it became clear that providing support for strict predefined procedures does not account for the intricate details which can be observed in work practice. Therefore, it has long been the goal of HCI to move away from a technological perspective on design, towards a contextual one. To this end the full context within which a new computing system will be used is considered during design, including the social setting and work environment. Moreover, end users of the system are usually encouraged to participate in the design process. This provides the opportunity to account for otherwise easily overlooked scenarios which could severely impact the effectiveness of the final system. However, as indicated by the similarity of present-day desktop computing systems to the first graphical window management system (Figure 1), and the recent adoption of ‘apps’ in mobile computing, we clearly continue to inherit concepts of a primarily technology-focused past.

A different approach to design is needed if we are to overcome preconceived notions of how computing systems should work. The predominant paradigm to interaction design in the early 1980s assumed that human actions follow predetermined algorithmic plans which can be modeled in a computing system. Suchman's observations provided a convincing critique to this cognitive paradigm, broadening the scope of investigation for interaction design. However, her work was influenced by a profoundly atheoretical subfield of sociology, ethnomethodology; it did not provide the groundwork for a new theoretical foundation. Without theory however, it is hard to compare, abstract, and generalize results, thus HCI has been on the lookout for a new theoretical framework to guide its research. One such framework, 'activity theory', has been explored extensively in interaction design for the past two decades [74, 97]. In line with Suchman's recommendation, the main unit of analysis in activity theory is the purposeful interaction of a subject with the world, incorporating the full context of human activity. This framework is therefore often used as an 'analytical lens' to inspire the design of new systems and to highlight possible issues which might otherwise be overlooked.

This focus on 'activity' as the key unit of analysis has also been adopted in 'activity-centric', or 'activity-based' computing systems. Activity-centric computing has been brought forward as an alternative computing paradigm, aiming to provide direct support for human activities, rather than through intermediate abstractions such as files and applications [101]. Although not all activity-centric computing systems have been inspired by activity theory, the common underlying assumption is the same: activities reflect a useful conceptual understanding users have of the world, thus they should be incorporated directly into the design of interactive computing systems. A *top-down* approach to design is followed, based on a thorough understanding of human activity, as opposed to a *bottom-up* approach to design, guided primarily by the underlying technologies and inherent restrictions upon which a system is built.

This new paradigm has been applied successfully to several different domains, ranging from ordinary desktop computing, to software development, to nomadic hospital work. It focuses on several aspects of human activity. First, by providing explicit support for users to structure their work within the context of activities, mechanisms can be put in place to more easily switch between parallel ongoing work as part of multitasking. Second, documents and other resources belonging to collaborative activities can easily be shared among participants by placing them within a shared activity context. Lastly, given that users nowadays own (or need to use) multiple devices from which the same resources need to be accessed, a shared activity context can also be used to access resources from multiple devices.

Activity theory provides a potential theoretical framework for HCI.

Activity-centric computing is an alternative to the current outdated computing paradigm.

Resources can be organized and accessed within the context of activities.

1.1 RESEARCH QUESTION

*Open issues in
activity-centric
computing.*

The overarching goal of this dissertation is to advance the research agenda of activity-centric computing. To do so I investigate two of the open issues formulated as part of a workshop on how to move from a theoretical understanding of activities towards providing computational support for them [14]. Although the workshop was held over a decade ago, the issues mentioned are still very prominent today, as summarized in a recent review of activity-centric computing [20].

ACTIVITY LIFE CYCLE: One particularly challenging issue of activity-centric computing is that it is often hard to keep activities separated from one another. There are no clear demarcations between the start of one activity and the end of another. For example, while working on the implementation of a new feature, a software developer might temporarily investigate an unrelated bug report he just received; as a result, irrelevant source files would end up within an activity labeled “Implementation of feature X”. This can be the source of much confusion. In addition, at times it can be difficult for users to decide when to create a new activity; e. g., should a new activity be created per feature which needs to be implemented, or with more granularity, when looking up related documentation?

ORGANIZING AND MANAGING ACTIVITIES: Another challenge concerns scalability; it is by no means clear how to scale current research systems so that they could be deployed in a real-world environment. For example, an activity-centric medical record system in a modern hospital would need to handle a significant number of patients, users, physical artifacts, and activities. If not carefully designed, a user would in no time accumulate a significant number of activities that would rapidly become obsolete. Hence, tools and methods for managing, linking together, and navigating a complex web of activities are needed.

Encompassing these two issues, the central research question of this dissertation is:

How can support for the full life cycle of activities, from creation to completion, be incorporated into an activity-centric computing system?

It entails the following two underlying research questions:

- R1 *What constitutes the activity life cycle? How do users manage their activities in contemporary computing systems, and what influences the creation and lifetime of an activity?*
- R2 *How can support for long-term activity management be included in an activity-centric computing system?*

Unfortunately, activity-centric computing is missing a systematic goal which can help guide the design of new systems. There is no clear guidance on how to address these research questions; should they be addressed in isolation, or are they part of a bigger research agenda? Although inspiration can be drawn from a broad range of research fields, including [HCI](#), personal information management ([PIM](#)), computer-supported cooperative work ([CSCW](#)), and information visualization, it is unclear how activity-centric computing is interlinked with other research. Therefore, [Part i](#) of this dissertation will first introduce a theoretical framework for interaction design, within which a goal for activity-centric computing is formulated. Thereby, the technological contributions presented in [Part ii](#) are clearly framed in light of prior work. Subsequently, [Part iii](#) and [Part iv](#) will present and discuss new empirical findings, and reflect on a more long-term goal.

Part i introduces a conceptual framework for interaction design, formulating a goal for activity-centric computing.

1.2 RESEARCH METHOD

Mackay and Fayard discuss the differences between *deductive* and *inductive* approaches to the scientific method. While the deductive model starts from *theory* and the inductive model starts from *observation*, both acknowledge the existence of these two ‘worlds’: the theoretical and the empirical [86]. The process of *design* on the other hand is quite different: throughout several iterations, prototypes are developed into finished products. Mackay and Fayard argue “that scientists strive to understand whereas designers strive to create. [HCI](#) is an interdisciplinary field that must do both.” To this end they introduce a triangulation model ([Figure 2](#)) outlining how different methods can be used to address a given problem in [HCI](#) [86]:

Within HCI, multiple methods need to be adopted in order to address a single question.

At the theoretical level, we can create and revise interaction models based upon observations of users interacting with artifacts. At the empirical or real-world level, we can observe how people interact with various technologies and develop models of use. In both cases, we can draw from theory and observation to instantiate new artifacts, ranging from early simulations to working prototypes to products.

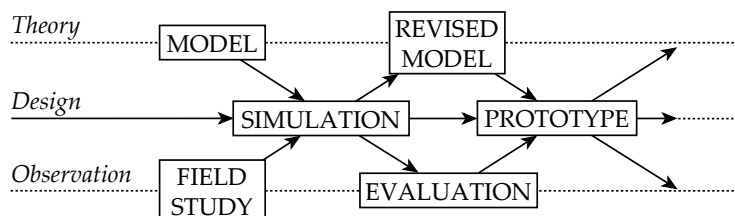


Figure 2: Triangulation model for [HCI](#) by Mackay and Fayard [86], adapted from original.

This dissertation contributes to theory, design, and empirical findings.

In order to reflect on the different methods used throughout this dissertation, I will outline the overall process within Mackay and Farvard's triangulation model. Figure 3 shows the *theoretical, design, and empirical* contributions described within this dissertation.

THEORY: Based on a PIM literature review (and a pilot study of an early system prototype) a conceptual framework—the *activity life cycle*—which describes the different states and transitions of human activity on a desktop computer while multitasking, is constructed. As part of a bigger research agenda, this contribution is situated within a theoretical framework for HCI—the *interaction framework*. It highlights how the activity life cycle contributes to an overarching goal for activity-centric computing.

DESIGN: Through iterative design, multiple prototypes of a desktop system for multitasking and interruption handling have been created—*Laevo*—incorporating support for the full activity life cycle. Based on an extended evaluation of this prototype, as well as associated experimental studies, a reusable software toolkit to aggregate resources handled by independent applications within *dedicated workspaces* is presented. Lastly, the design of *Laevo* is extended to include support for cooperating teams. This system—*co-Laevo*—has not been evaluated yet, but its design is based on accumulated knowledge throughout the project and its contribution is framed within the interaction framework.

OBSERVATION: During a two week qualitative in situ field study *Laevo* was *evaluated* by six participants. One of the features of *Laevo* was experimentally evaluated in a comparative study involving 16 participants. This study provides quantitative insights into how well *dedicated workspaces* support *sequential multitasking*. Contributing to these findings, a second study investigated multitasking in a traditional window manager, thereby highlighting shortcomings of current systems.

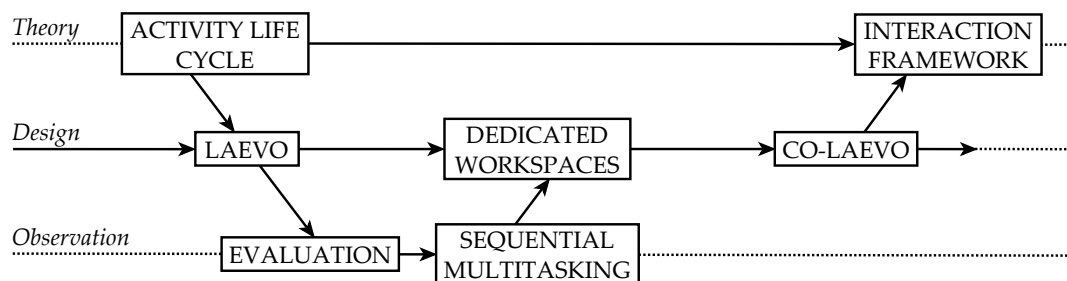


Figure 3: An overview of the main activities and deliverables discussed in this dissertation.

1.3 OVERVIEW OF CONTRIBUTIONS

This dissertation is a monograph within which all of my work on activity-centric computing, both published and previously unpublished, is presented as a whole. It is subdivided into four parts. In [Part i](#), a theoretical framework for [HCI](#) and activity-centric computing is presented (the interaction framework), which will serve as a roadmap throughout the remainder of the dissertation. In [Part ii](#) the technological contributions are discussed in light of prior work and framed within the interaction framework. In addition to clarifying the novelty of the contributions presented here, this provides a clear roadmap for future work on activity-centric computing. In [Part iii](#), I present an evaluation of the system introduced in [Part ii](#), and two experimental studies investigating multitasking on a desktop computer. Lastly, in [Part iv](#), I will relate these empirical findings to open issues in activity-centric computing. In the remainder of this introduction, for each of the contributions (outlined in [Figure 3](#)), I will mention where they have been published before and where they will be discussed within this dissertation.

This dissertation is subdivided into four parts:

[Part i](#) *A design space*

[Part ii](#) *Design and technology*

[Part iii](#) *Empirical studies*

[Part iv](#) *Discussion and conclusion*

INTERACTION FRAMEWORK: The *interaction framework* is a theoretical framework for [HCI](#) within which different perspectives on the ‘user interface’ can be positioned and related to each other. This is previously unpublished work and is presented in [Chapter 3](#), after which it will serve as a roadmap throughout the remainder of this dissertation.

ACTIVITY LIFE CYCLE: The *activity life cycle* was first described as a “temporal model for reflective multitasking” in a workshop on conceptions and experiences of time at the Conference on Human Factors in Computing Systems ([CHI](#)) [63]. This conceptual model depicts the practices and processes of human activity, identified based on a literature review of [PIM](#) (presented in [Chapter 5](#)). It formed the conceptual basis for the design of an activity-centric desktop computing system—[Laevo](#)—of which a publication includes an updated version of the model [64]. In this dissertation I will reintroduce the model in [Chapter 6](#) as part of presenting [Laevo](#).

LAEVO: *Laevo* is a personal desktop computing system which addresses open issues in activity-centric computing. A review of these issues (and activity-centric computing, [Chapter 5](#)) was published in a special issue of the [AI Magazine](#) on activity recognition [20]. *Laevo* addresses these prior open issues by recognizing the need to support the full activity life cycle. The resulting design and *evaluation* of the system was published and presented at the conference on User Interface Software and Technology ([UIST](#)) [64]. In this dissertation, *Laevo* will be discussed

alongside co-Laevo in [Chapter 6](#). The evaluation of Laevo is presented in [Chapter 8](#).

SEQUENTIAL MULTITASKING: The experimental study on dedicated workspaces (used in Laevo) was published in the journal of *Computers in Human Behavior* [62]. A follow-up study investigating how well traditional window managers support multitasking is in submission to the same journal [66]. Both studies are presented here in [Chapter 9](#) as a whole.

DEDICATED WORKSPACES: The software toolkit in support of *dedicated workspaces* will be described in [Chapter 7](#) of this dissertation, and has not been published before.

CO-LAEVO: *co-Laevo* extends on Laevo by including support for co-operating teams. Its design is described in a technical report at the IT University of Copenhagen [65]. In this dissertation co-Laevo is presented alongside Laevo in [Chapter 6](#), serving as an indication of how the conceptual model from Laevo can easily be extended on.

Part I

A DESIGN SPACE FOR
ACTIVITY-CENTRIC COMPUTING

COMPUTER-MEDIATED ACTIVITY

[T]here is a continuity to the outward movement of the computer's interface to its external environment, from hardware to software to increasingly higher-level cognitive capabilities and finally to social processes.

— Jonathan Grudin [51]

Historically, the development of computing systems has followed a *bottom-up* approach. First the underlying hardware had to be built, before programming languages could be created to operate on them, before rich new user interfaces could be implemented. This is quite a natural progression, since progress on lower levels opens up opportunities for innovations higher up. Consequently, user interface design has also shifted its focus over time: originally there was a focus on the hardware interface, followed by a focus on software, whereas most of HCI research today focuses on the work settings and its associated social context. The computer interface thus seems to ‘reach out’ into the environment as technology evolves [51]. As a result, the type of ‘users’ that interact with user interfaces has also changed over time. The first user interfaces were mainly used by engineers and programmers, whereas modern personal computers are ubiquitous, and need to support groups of users with a wide variety of professional backgrounds. This expansion has led to the multidisciplinary nature of HCI, which draws upon research in fields like computer science, cognitive science, design, and more recently social science (Figure 4).

Within such a multidisciplinary field, it can be challenging to find common ground. Each discipline brings along its own theories and associated methods, which can pose a challenge when interrelating findings from opposing views. For example, cognitivism dominated HCI research in the early 1980s. With its focus on information processing, effectively equating humans to computers which follow predetermined plans, it was presumed that all human tasks could be modeled using task analysis (e. g., GOMS). This cognitive approach brought along with it powerful predictive models, including Fitts’s law (still used to model pointing tasks), but failed to consider broader aspects of *user experience (UX)*. Therefore, later in the 80s, design experts emphasized the importance of the user, introducing usability, accessibility, and pleasure as important determining factors for effective human-computer interaction. Around the same time, researchers interested in collaborative computing opposed the methods from cognitive science used in HCI [10] and banded together to form a new (yet overlapping) research area—CSCW—which focuses on the social and organizational context of interactive computing systems [119].

As technology evolves, the context studied in user interface design expands.

HCI is studied by various disciplines, each with their own theories and methods.

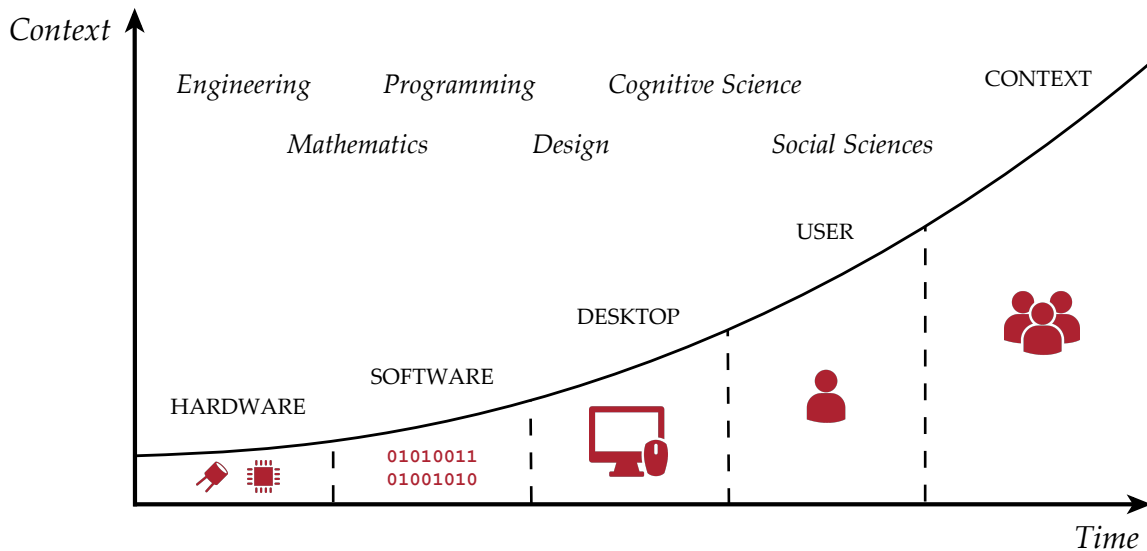


Figure 4: A coarse depiction of the expansion of user interface design and associated scientific disciplines over the course of history. (Icons designed by Freepik.)

The primary concern of interface design is to support human activity.

As wide and varied as the current field of **HCI** is, what unifies it is reflected in its name: the interaction of humans with computers. Unlike the dominant cognitive paradigm from the 80s, user interaction nowadays is generally considered to be more than mere information processing between a user and computer; rather, as popularly phrased by Bannon, users are considered to be ‘human actors’ with “a set of values, goals and beliefs about life and work” [9]. The user’s point of view is thus adopted as the main perspective during design. This sentiment is reflected by Kaptelinin, who goes so far as to suggest a new term for the discipline: *computer-mediated activity* [71]. This emphasizes the primary concern of **HCI** in supporting human activity, and reframes the role of interactive computing systems as that of mediating tools within the broader context of human intentionality.

Activity-centric computing aims to support the full context of human activity.

One line of research—*activity-centric computing*—(studied in both **HCI** and **CSCW**) emphasizes this view even further and postulates that providing computational support for human activities is fundamental to the design of effective interactive computing systems. Within this line of research, ‘activity’ is commonly interpreted along the lines of (or directly inspired by) the main unit of analysis in activity theory [44]: the purposeful interaction of a subject with the world. As part of a community, mediated by rules and division of labor, the subject works on objects through the use of tools, towards a desired outcome (Figure 5). What constitutes an activity is user-specific “since it relies on which intention is expressed by or is meaningful to the user” [20] (e.g., writing a thesis, or developing a piece of software). Thus, activity-centric computing tries to tailor to many different types of users and activities, which is why it often is envisioned as a general purpose computing system.

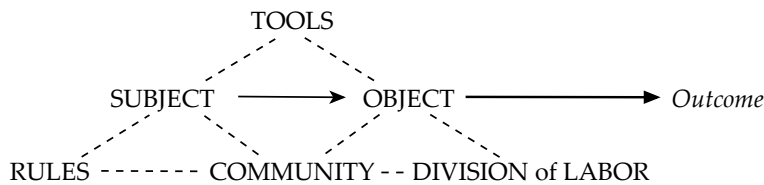


Figure 5: Activity system model: the unit of analysis in activity theory. Figure adapted from original by Engeström et al. [44].

2.1 HISTORY OF ACTIVITY-CENTRIC COMPUTING

The idea for activity-centric computing has been around for over three decades. As part of a ‘hardy band of souls’¹ at Apple Computer, Don Norman was one of the first to name the concept, which they referred to as ‘activity-based computing (ABC)’ [101, Ch. 4]:

ABC is an ambitious idea which has been around since the 1980s.

The basic idea is simple; make it possible to have all the material needed for an activity ready at hand, available with little or no mental overhead. Tools, documents, and information are gathered together into packages maximally designed for the particular activities in which they participate, without interfering with other activities.

Their design (which never made it in to product) was based on an understanding of the psychology of human activities and interruptions. Already in the 1980s it was suggested that systems should [93]:

1. Support easy suspension and resumption of activities²: (1) without interfering other tasks; (2) activity state should be saved so tasks can be continued where left off; (3) and that users should be reminded of unfinished tasks.
2. Support concurrent activities by allowing to simultaneously view different components related to a complex task.

About 10 years later (while working at Apple), Norman envisions that “activity spaces could be shared with other people or copied from one machine to another” in which case there is a need to “figure out how to coordinate the work so that one person’s actions do not interfere with another’s”. Activity spaces would allow the user to “add or subtract tools as needed” and “software companies might provide specialized activity spaces” which come prepackaged with the necessary tools needed to execute specific procedures [101, Ch. 4]. From a cursory reading of early HCI literature it thus becomes clear that activity-centric computing is not a novel idea and that it was quite ambitious from the start.

¹ During 1993–1998 Don Norman was vice president of the ‘Advanced Technology Group’ (Apple’s research arm) and part of a group working on ABC: Thomas Erickson, Charlie Hill, Austin Henderson, Dan Russell, Harry Saddler, and Mitch Stein.

² ‘Activity’ and ‘task’ seem to be used interchangeably by Miyata and Norman.

Workspaces allow grouping resources related to one activity together.

In fact, a similar proposal with a similar vision predates Norman's definition (and even the widespread adoption of windowing systems) and was based on the empirical analysis of command-line histories [11]. To address problems that users encountered when attempting to accomplish several different tasks in a single session, Bannon et al. introduced the notion of a 'workspace'³: "an environment dedicated to allowing easy user manipulation of activities to achieve a particular goal or set of functionally related goals" [11]:

From the users' perspective, workspaces must have highly dynamic internal structures which can be modified as users reformulate their goals. From the system's perspective, a workspace contains tools and data relevant to the users' goals and, in addition, provides a record of the ongoing activities or processes resulting from applying a set of tools to a set of data structures. The internal structure of the workspace thus reflects both the users' goals and the software tools that the computer system can provide for accomplishing these goals.

An analysis of Interlisp-D window usage followed soon after and highlighted similar results: users regularly switch between multiple activities, identified as sets of windows that are repeatedly used in unison. This inspired the design of the now canonical Rooms system [55], supporting some of the envisioned workspace features that later came to be known as 'virtual desktops' (Figure 6). However, this is but a small subset of what activity-centric computing entails.

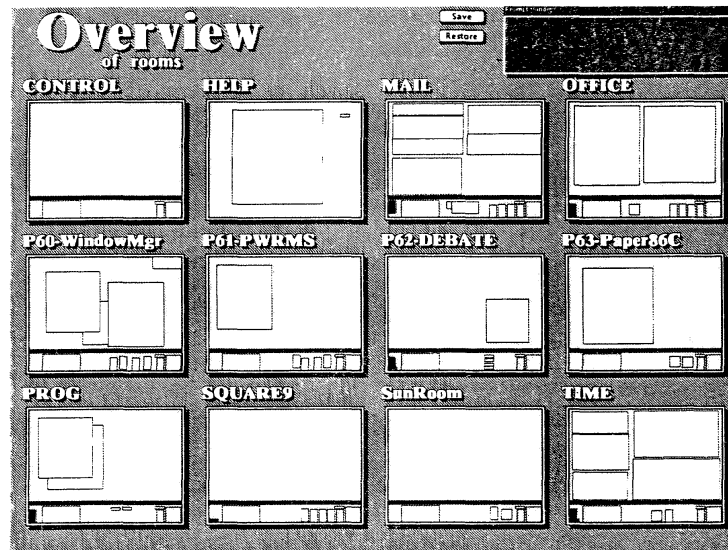


Figure 6: An overview of workspaces in the Rooms system [55].

³ A possible cause for confusion here is that the notion of 'workspace' by Bannon et al. corresponds to that of 'activity space' by Miyata and Norman, whereas 'task' and 'activity' are used quite liberally, including to refer to actions within a workspace.

The functional equivalent of workspaces (allowing the user to group user interface elements together which can be retrieved in unison) existed long before Rooms came along. To go back in time: a conceptually similar system (named 'Room') was introduced by Chan prior to Rooms [35]; Smalltalk-80 (the pioneer of graphical window management systems) supported 'project' workspaces [48]; and even Engelbart's NLS system from 1968 was flexible enough to allow the user to set up a list of activities which linked to associated workspaces [43]. What sets activity-centric computing apart is the intent assigned to a workspace by the user and the computational support provided for it. More than just an aggregation of resources, activity-centric computing aims to incorporate support for the full context of human activity, including collaboration. Therefore the distribution and management of activities needs to be supported as well. Although Rooms seems to be inspired by a similar vision, its implementation does not venture beyond providing basic support to group resources.

Outside of research the concept of activity-centric computing never really caught on. The first mass-marketed personal computers did not even support the more basic notion of multiple workspaces. Therefore not unexpectedly, more than 20 years after Bannon et al. suggested to provide support for easy suspension and resumption of activities [11], studies were still reporting on the complexities of multitasking in contemporary computing systems [37, 49, 89]. As a result the concept acquired yet another name: González and Mark referred to the need to support multiple 'working spheres' [49]. Although most contemporary operating systems now finally support an implementation of workspaces similar to Rooms (only recently introduced in Windows 10), just a limited subset of the envisioned functionality originally proposed by Miyata and Norman is incorporated [93]. Window management systems primarily support the second requirement for human activity (allowing to simultaneously view different components related to a complex task), whereas the workspace functionality seems to be tacked on, not fully supported by some applications.

This provides the historical context for activity-centric computing. An overview of this line of research (also the topic of this dissertation) will be provided in [Chapter 5](#), but here, I will first introduce a conceptual framework which will serve as a research agenda. Given the extensive scope covered by this research, a matching theoretical foundation needs to be equally broad. It should cover a general understanding of human activity and thus be applicable to multiple settings and domains. Although such theories exist (as will be summarized next), they are not *prescriptive*; they can inspire the design of specific computing systems by highlighting key points of interest but do not lay down generalizable implications for design. Moreover, they tend to carry with them rich and complex philosophical backgrounds, which, as I will argue here, might not be relevant for design.

Goal-oriented workspaces are but a small subset of activity-centric computing.

At present, workspaces are widespread, but activity-centric computing remains limited to research.

This part introduces a conceptual framework for activity-centric computing.

2.2 THE SEARCH FOR THEORY

Postcognitivist theories in HCI are descriptive.

Theories within HCI can be divided into two main categories: *descriptive* (and explanatory) and *prescriptive* (and predictive) theories. Theories within the former category primarily serve as analytical tools (facilitating the thought process), whereas theories in the latter can make predictions about performance and provide specific guidelines for design. Both are useful and serve their distinctive purpose, however, ever since HCI has distanced itself from the cognitive information processing paradigm, theory has been primarily seen “as an explanatory device, and as an aid to understanding, rather than as a predictive instrument” [114]. E. g., during ethnographic observations theories act as an analytical lens, highlighting the social and organizational context of human activity. Rich descriptions of work practice are created as it occurs (in situ), before and after the introduction of a new or existing technology. This allows to identify concrete issues which need to be addressed in system design. However, with the focus of ethnography on the details of practice, such observations are highly situation-specific and are thus hard to generalize [33]. They form a valid critique of specific technologies but do not pave the way to improving overall system design. Even more so, ethnomethodology, as popularized by Suchman in an early observational study [126], explicitly shies away from generalizing results. Postcognitivist theories in HCI (such as activity theory, actor-network theory, distributed cognition, and phenomenology) are thus predominantly descriptive and avoid reducing human activity to a set of core concepts which need to be supported within a computing system [74, Ch. 9]:

If postcognitivist theories have taught us anything, it is that the closer we look, the more complex ordinary activities appear, and the less amenable they are to reductionist accounts.

The design of general purpose computing systems requires prescriptive models.

Regardless of the intrinsic complexities of human activity, in order for software engineers to be able to construct a common underlying architecture in support of them, there is still a need to reduce activities to a simplified computational representation. The very nature of software engineering—constructing reusable components that build on top of one another—requires a reductionist approach to system design. Postcognitivist theories thus seem to be at odds with software engineering. Although descriptive theories are particularly suitable as heuristic tools during the observation of existing systems (sensitizing the ethnographer to the complexities of situated action), they can not *prescribe* central features which need to be supported in a general purpose computing system, i. e., a personal computer. Postcognitivist theories can guide incremental innovation by highlighting specific problems with existing technology, but do not outline a broader vision for future technologies.

There is a problem with such a ‘localized’ view on design, where technologies are tailored entirely to specific practices. Creating isolated technologies easily disregards issues of interoperability, further contributing to readily observable problems in computing like information fragmentation [25]. Each application ends up storing, managing and viewing information items within separate application-specific collections. What is needed is a more general understanding of human activity: common findings across studies which can directly impact the design of a broader range of systems and help to design them in such a way so they work well together. In other words, what is needed is a general conceptual understanding of the ‘physics of practice’; what are the core concepts of human activity that need to be supported? Such a collection of theoretical concepts could be used by system designers to expose as computational constructs, providing an intuitive and coherent user experience across applications.

To this end, more important than being able to *explain* human activity, we need to understand how users *experience* it; the two do not necessarily coincide. For example, although activity theory decomposes activities into underlying actions and operations (a useful construct for analysis), users do not express their goals as such. It does not make sense to represent the inner structure of a particular theory within the user interface. Therefore, we can sidestep the “boiling cauldron of competing social science perspectives” [114] (and their associated philosophical backgrounds) and focus on the task at hand: designing *computer-mediated activity*. For the purpose of interaction design, it is quite sufficient to understand the mental models users have of the world. Such conceptual models (as will be detailed later) can form part of a *prescriptive* theoretical foundation for HCI from which general implications for design can be derived.

Theory also serves a broader purpose. Other than providing a clear goal to direct the design of interactive computing systems, theory can unite communities through the introduction of shared concepts and help make strategic choices on how to proceed [74, 81, Ch. 2]. Emphasizing this even further, Barthelmeß and Anderson state [21]:

The value of any theory is not “whether the theory or framework provides an objective representation of reality” [13], but rather how well a theory can shape an object of study, highlighting relevant issues.

Although current postcognitivist theories can be used as such, they carry with them rich and complex philosophical backgrounds which might inhibit cooperation; conceptual models do not, and can more easily be adopted as a ‘common language’ between opposing views. In such a scenario, the main role of descriptive theory is no longer to *explain* human activity, but to validate, extend on, and introduce new *prescriptive* conceptual models.

Designers need to understand the core concepts of human activity for which to provide support.

Conceptual models are prescriptive.

Descriptive theory can validate, extend on, and introduce new prescriptive conceptual models.

2.3 IMPLICATIONS FOR DESIGN

Ethnography can contribute to design in more ways than simply reporting on empirical accounts.

The apparent conflict between the ‘localized’ view of ethnography and system design has been recognized for quite some time. Nonetheless, some implications for design can be derived through ethnography. Button and Dourish discuss three ways by which ethnomethodology in particular can influence design, which they coin ‘technomethodology’ [33]. First (and most commonly employed), the ethnomethodologist works closely with the designer, providing direct input based on observations done in the field. Second, letting an ethnomethodological account speak for itself, the designer draws inspiration from reading the analysis of the work situation constructed by the ethnomethodologist. However, neither of these two approaches are directed towards creation of the reusable theoretical concepts that we are after. More in line with this objective, a third approach suggests “to align system design not so much with the details of *specific working practices*, as with the details of the *means by which such working practices arise and are constituted*” [33]. They posit that the ‘generally operative social processes’ which are identified in ethnomethodology can be adopted as resources for design. These are the analytical insights which are the true important implications for design [39]. In essence, Button and Dourish seem to argue that ethnomethodology should develop precisely the theoretical foundations interaction design is looking for [71, 74, Ch. 2].

Ethnographic findings are mostly highly specific, or overly broad sensitizing concepts.

However, from an overview of 25 years of ethnography in design it becomes clear that discussions on how to effectively interrelate the two and on how to construct such a general understanding of the ‘nature of work’ are still actively ongoing [27]. What is evident, however, is the stark contrast between how the majority of findings in ethnography are reported and the distilled format in which design researchers anticipate to receive them. Based on interviews with 12 expert design researchers Sas et al. summarize [117]:

In contrast with ethnographic research emphasizing “thick descriptions” and HCI textbooks highlighting communicating devices, i. e. personas and scenarios, our findings show limited evidence of these types but a strong interest in “short descriptions” of fieldwork data.

As hinted at earlier, such results do commonly make up the ‘implications for design’ section at the end of HCI papers which report on ethnographic field studies [39]. These results can be broadly categorized as either being domain- or technology-specific findings, or broader sensitizing concepts highlighting relevant social aspects concerning technology use (e. g., social awareness) [117]. However, it is quite rare for findings to relate to generalizable conceptual models users have of the world [104], which arguably have thus far had the greatest impact on system design (e. g., the desktop metaphor).

2.4 CONCEPTUAL MODELS

Early on in the history of HCI, Norman and Draper introduced the notion of using clear coherent ‘conceptual models’ as an appropriate approach to system design [104]. When interacting with a system, users form a mental model which they use to anticipate and reflect on system behavior. It is thus beneficial for designers to expose preexisting conceptual models users might have in the user interface, either from the real world or from using prior systems. When the user’s mental model coincides with that which is introduced by the designer into the system, their need to rely on system documentation is diminished. In a way, the designer ‘talks’ to the user through the system to convey its functionality; an image of the designer’s conceptual model is embedded into the system, from which the user gradually shapes a mental model which is relied upon during user interaction (Figure 7). A widespread example of this is the user interface on desktop computers: folders and documents placed on a digital desktop are designed to resemble a physical desk in an office workspace.

Conceptual models are powerful generalizable tools for system design.

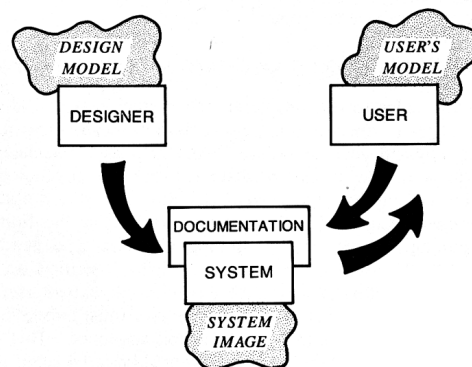


Figure 7: Conceptual model: the designer’s model, the system image, and the user’s model. [104].

Conceptual models entail three different types of constraints which can prohibit, discourage, or encourage certain user interactions [102]:

Behavioral constraints guide user interaction.

PHYSICAL CONSTRAINTS: These make it physically impossible to perform a certain action. For example, it is impossible to move the cursor outside of the screen.

LOGICAL CONSTRAINTS: By implicitly exposing the underlying design model, users can deduce available or necessary actions themselves. For example, users can anticipate more text is available in a partially visible text document.

CULTURAL CONSTRAINTS: These are conventions that have evolved over time, and ideally are a good fit with human cognition. For example, clicking and dragging a scrollbar will reveal content beyond what is visible.

Well designed systems incorporate consistent and natural conceptual models ...

The trick to good system design, then, lies in finding such appropriate conceptual models and ensuring they are consistent throughout all of the user's interactions with the system. Consistency is what allows the user to accurately predict system behavior. Therefore, particularly appropriate conceptual models are those that coincide with mental models users already have of the 'natural' world; such models are consistent beyond the user interface, and help to bridge the gap between the physical and digital world. This is in line with Weiser's concept of '*ubiquitous computing*' where he explores the boundaries of this concept by anticipating computers will become 'invisible'. He states that "[m]achines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods" [137].

... that are broadly applicable.

Another important aspect to good design is to choose conceptual models which can be applied to a broad range of situations as this helps to reduce the amount of interactions users need to rehearse and recollect. To demonstrate, it suffices to look back at the design methodology of the highly influential Xerox Star [68]. The Xerox Star "adheres rigorously to a small set of design principles" in order to "unify the nearly two dozen functional areas of Star, and allow user experience in one area to apply in others" [123]:

We have learned from Star the importance of formulating the fundamental concepts (the user's conceptual model) *before* software is written, rather than tacking on a user interface *afterward*. Xerox devoted about thirty work-years to the design of the Star user interface. It was designed *before* the functionality of the system was fully decided. It was even designed *before* the computer hardware was built. We worked for two years *before* we wrote a single line of actual product software.

The conceptual models used in contemporary systems are outdated.

The Xerox Star interface (Figure 8) thus reflects one of the earliest commercially influential attempts to follow a *top-down* approach to design, based on the perspective of the user, as opposed to the traditional technology-driven *bottom-up* approach to design. It brought many useful concepts still present in modern day computing systems to market, most importantly the desktop interface and 'universal commands' like 'copy' and 'delete'. However, as an 'office information system' it was modeled after the metaphor of a physical office, a concept now largely antiquated; the majority of information is no longer stored in physical files, computers are no longer chained to office desks, and computing devices of varying shape and size have made their way into our everyday lives. Computers have thus become ubiquitous, but unlike Weiser's vision they have yet to become invisible. Therefore, the current challenge for HCI is to find new appropriate conceptual metaphors, tackling this broader context in which computational devices are used.

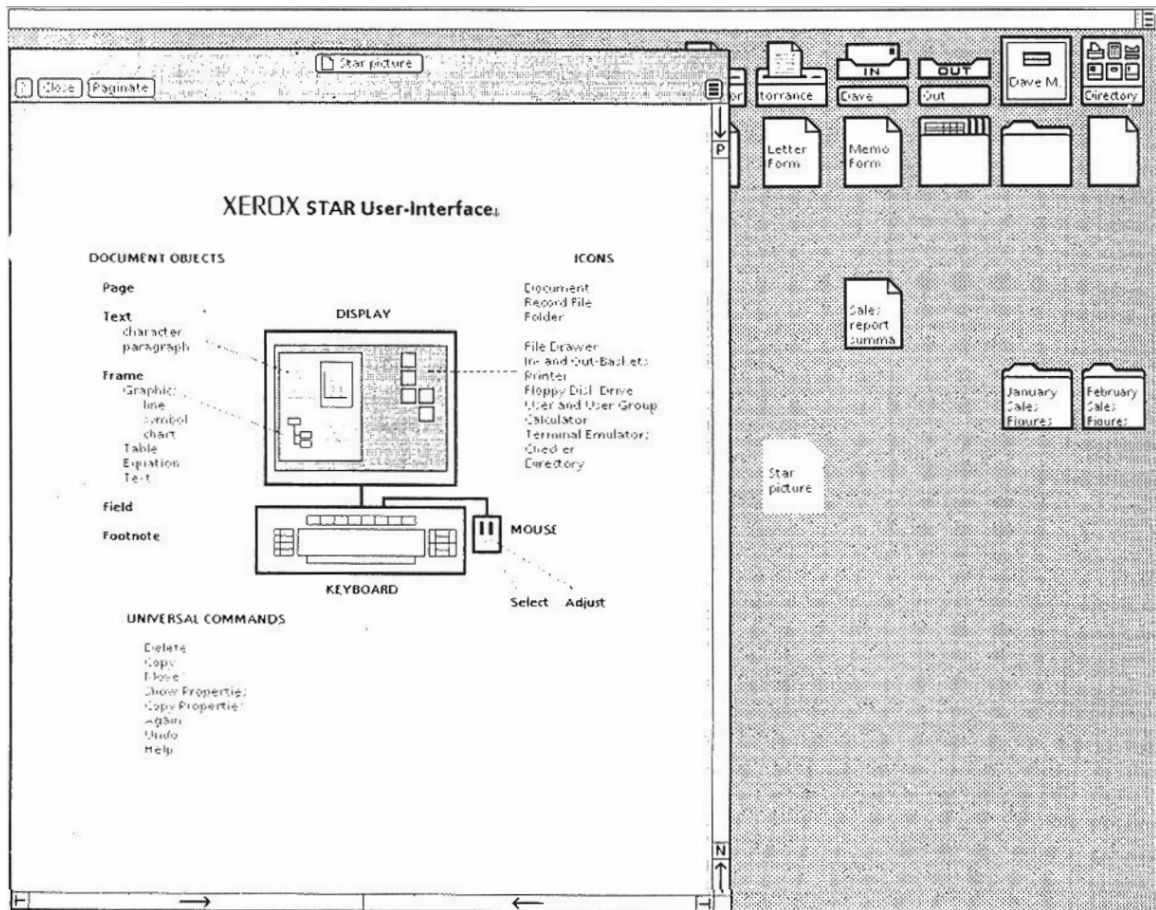


Figure 8: The Xerox Star user interface which commercialized the desktop metaphor [68, 123].

2.5 MOTOR THEMES

Long-term design efforts such as that of the Xerox Star are now rare. Where the Star had an open playing field, largely unrestricted by prior technologies, the design of contemporary systems is inhibited by concerns of backwards compatibility (more likely to get adopted by users). In fact, even though the Star was highly influential in shaping modern computing, the workstation itself did not become a commercial success. Instead, information technology is governed by a fast-paced competitive market (even in research) with a constant push for short-term innovations. In an analysis of proceedings at CHI it is shown that “when a new technology comes along it seems that researchers start from scratch leading to relatively isolated research themes” [85]. Old topics are constantly being replaced by newer ones, with a limited amount of insights being handed over. There are no *coherent* and *central* ‘motor themes’ which can drive scientific research⁴.

HCI lacks motor themes which represent accumulated knowledge.

⁴ In a plenary session at CHI 2016 Alan Kay even states: “Well I don’t think we have a field. I think it is still basically a pop culture probing around; there are little pockets of this and that, but it doesn’t act like any field that is taking its larger mission seriously. It is kind of a caricature.” ([youtube.com/watch?v=S6JC_W9F8-g](https://www.youtube.com/watch?v=S6JC_W9F8-g))

Motor themes are needed to make a bigger impact on society.

Motor themes are represented within a *strategic diagram*, depicting research themes studied within a field along two dimensions: *centrality* and *density* (Figure 9a). Centrality indicates to which degree a theme interacts with other research themes. Density represents the strength of links within a specific research theme (internal cohesion). Motor themes are research themes which are both central (impacting the entire field) and dense (well-developed themes) [79]:

A theme begins its life with low centrality and density in the Chaos quadrant. As the theme becomes more central to the community, it moves to the Bandwagon quadrant. The theme eventually matures its internal cohesion and moves to the Mainstream quadrant, where the motor themes of a community lie.

In their analysis, Liu et al. show that no such themes exist within the CHI community (Figure 9b), which is representative of multiple competing perspectives in HCI. This “should be a very worrying prospect for a scientific community” [79]. Motor themes are needed to drive an overarching research agenda, allowing themes to advance sufficiently into the mainstream, i. e., to make a bigger impact on society.

IDENTIFYING MOTOR THEMES To summarize, in this chapter I have so far highlighted the multidisciplinary nature of HCI, a research field which studies *human activity* mediated by computational technologies. The field is plagued by a lack of *generalizable implications for design* as a result of its focus on short-term innovation whenever new enabling technologies arrive. However, I introduced *conceptual models* as an important and historically successful approach to system design. I will conclude this chapter by arguing how a renewed focus on the general conceptual models that users form of the ‘natural’ world can help in accumulating knowledge, and how such models can act as *motor themes* to foster collaboration across disciplines.

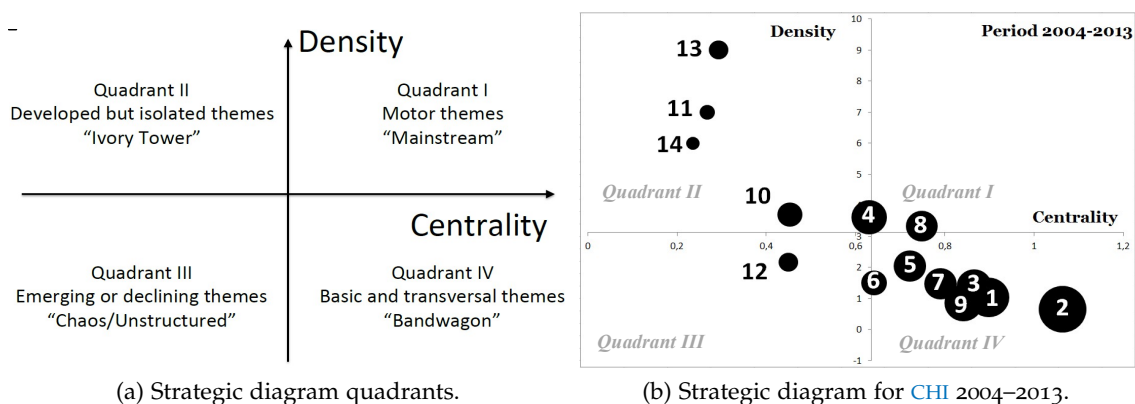


Figure 9: Strategic diagram for CHI indicating a lack of motor themes [85].

Ironically, even though the human is what unifies the different disciplines within [HCI](#), Liu et al. show in their review that there is “no discernible research theme emerging on this topic” [85]. A more intricate reading, as presented in this chapter, does reveal that attempts are made to guide research in this direction. However, given that no common theoretical framework exists, suggestions on how to accumulate knowledge on the nature of human activity seem to be fairly secluded and do not build on top of one another: e.g., where ethnomethodology speaks of ‘generally operative social processes’, system design speaks of ‘conceptual models’, to list but two examples highlighted earlier. Regardless of terminology, there seems to be a general consensus that knowledge should be accumulated around a context-free interpretation of human activity, as opposed to concrete instantiations of it. We should strive to understand the general processes which define human activity: the ‘physics of practice’.

As per the quote by Grudin at the start of this chapter, another constant within the field has been the ‘interface’. Although the entities between which interfacing takes place have evolved over time, the notion of an interface is inherent to [HCI](#); without the interface, no interaction can occur. Thus, it forms a suitable motor theme on which knowledge can be accumulated. However, “the interface is not a unitary concept”, and discussing it requires navigating a ‘terminological minefield’ [81]. Along with the expanding context of [HCI](#) came different perspectives on what constitutes the interface, again, each building on top of their own fairly secluded insights. In between all of this diversity, however, a glimpse of harmony shines through. No matter what abstraction level the interface operates on, common desirable properties persist. E.g., consistency has long been a goal within design. Likewise, on a much lower abstraction level, ‘conceptual integrity’ is an important characteristic of a well-designed [API](#) [26].

Human activity and the interface, then, are two motor themes which could form the basis for a research agenda in [HCI](#). Although taken at face value this might seem trivial (given the name of the field), there is a limited amount of accumulated knowledge, applicable to multiple settings and technologies, revolving around these themes. The brief examples listed here do indicate that commonalities between differing perspectives within the field exist, but in order to relate, contrast, and develop such findings they need to be framed within a shared theoretical framework. A shared framework does not imply reducing different perspectives to one overarching view; separate conceptualizations are still a possibility, and moreover, extremely valuable. However, insights need to be integrated through a common formalization of what the objects under study are. To this end, the two motor themes identified here form a suitable starting point, but they need to be dissected further into more narrow elements of study in order to become truly operational.

The human is a constant in [HCI](#) which can be used to accumulate knowledge.

The interface, although diverse, has common desirable design properties.

Motor themes can form the basis of a theoretical framework.

*Conceptual models
can be used as
shared concepts to
enable cooperation
across disciplines.*

Naturally occurring conceptual models, inherent to human nature, are a particularly interesting candidate to be made part of such a framework. Although they were introduced from the perspective of design, they sit in between the technical and empirical: they can guide the implementation of software systems by outlining the necessary interactions to be supported, but can also be placed under close scrutiny through ethnographical observations within the natural world. At first sight this might seem to conflict with grounded theory (as sometimes employed by researchers within the social sciences), however, even though grounded theory starts its analysis from the ground up, it does not eliminate the possibility of relating findings back to a common theoretical framework after data analysis has been finalized. Norman revealed the importance of ethnography in understanding conceptual models and their entailing behavioral constraints (including cultural conventions) [102]:

How do you know if the user shares the conventions?
Why, with data, of course. This is something that cannot be decided by arguments, logic, or theory. Cultural constraints and conventions are about what people believe and do, and the only way to find out what people do is to go out and watch them—not in the laboratories, not in the usability testing rooms, but in their normal environment.

Conceptual models, as part of overarching motor themes, can thus provide suitable 'hooks' across disciplines, enabling the construction of a shared body of knowledge.

INTERACTION FRAMEWORK

To become [an integrated field of studies], HCI should be based on a conceptual scheme powerful enough to incorporate both human beings and computer technology within a coherent theoretical framework.

— Victor Kaptelinin [71]

Different perspectives in HCI correspond to common motor themes (discussed in Section 2.5) which are studied at different levels of abstraction: e. g., cognitivist approaches study the mental operations which make up human activity, whereas sociological inquiry investigates the impacting factors of its surrounding context. Regarding one as more important than the other does nothing but stymie progress. Both perspectives (as well as others) are integral to constructing a complete understanding of human activity. Although the interface is ‘reaching out’ [51], it does not disappear from whence it came. A first important dimension for a theoretical framework in HCI should therefore cover different levels of abstractions of human activity. Each level can still formalize its own conceptualization of the ‘interface’, but all should be directed at a common understanding of what purpose it serves, answering the question “Why does interaction occur?” This forms a second important dimension for a theoretical framework in HCI: types of interaction. Both dimensions are depicted in Figure 10.

An integrative framework in HCI covers interaction, as part of human activity.



Figure 10: Two dimensions for a theoretical framework in HCI: levels of abstraction of human activity, and different types of interaction.

This two-dimensional framework only outlines the scope and complexity of studying the *interface*. It incorporates the two motor themes of HCI identified in the previous chapter, but without breaking it down into further subcomponents it can not act as an integrative framework to unite differing perspectives within HCI. In this section I will therefore decompose both dimensions into meaningful conceptually coherent concepts, which will act as a common thread throughout the remainder of this dissertation.

3.1 COMMON LANGUAGE

*Disciplines
construct their own
plane of knowledge,
but can interact
with others using
shared terminology.*

Tackling the multidisciplinary nature of HCI is by no means an easy task. Looking for inspiration we can turn to Otto Neurath, a philosopher of science who spent several years of his life on constructing a ‘unified science’ [109]. As part of ‘the movement for the Unity of Science’, he proposed not to strive for a ‘super-science’ (based on a prior and independent philosophy), but rather that “the special sciences will themselves supply their own synthesizing glue” in the form of unifying concepts which can be identified across disciplines [99]:

A large collection of terms have been gathered by the various sciences during the centuries, and it is necessary to examine this collection from time to time, for terms should not be multiplied beyond necessity. . . . Distinct terms occur in different disciplines which nevertheless may have the same function; and much fruitless controversy may arise in trying to find a distinction between them.

Such common concepts denote a ‘horizontal component’ which links the different sciences to each other within a two-dimensional conception of unified science: there is a “horizontal consistency and ‘harmony’ among the sciences”, however, “none, [Neurath] felt, should be considered more fundamental than others” [109]:

Although individual sciences might evolve toward hierarchical axiomatization, and the task of logically reducing some sciences to others was worth while, unified science as a whole, Neurath felt, should be conceived not as a pyramid but as a ‘mosaic’ [98].

Each science is tethered to its own horizontal plane of knowledge, along a second (vertical) dimension. Where planes overlap a common language should be introduced in order to enable accumulating knowledge which transcends individual disciplines. This metaphor is depicted in Figure 11, extending on the earlier introduced framework.

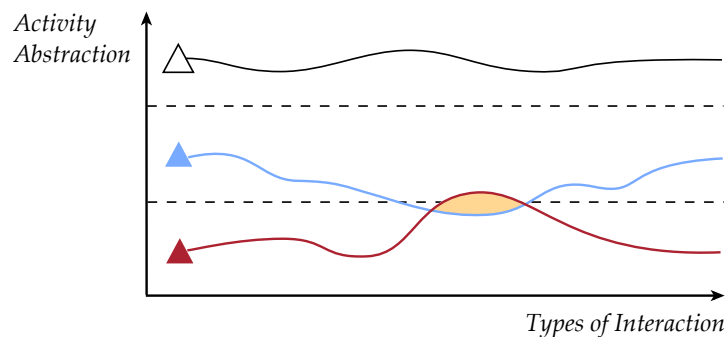


Figure 11: Individual sciences operating on their own plane of knowledge, and the interaction between them.

Considering different levels of abstraction in HCI “a hypothesis can be made that the problems and debates within HCI research ... are due to a change or enlargement of the research object of HCI from one level to another” [81]. Although this broadening of context is a crucial part of a maturing field of research, it does not mean that prior levels should be subsumed by newly introduced research philosophies. Integrating different perspectives within HCI is a collaborative research effort, and no single subdomain can rightfully claim a seat on the throne. As Neurath states while arguing for a unifying common language [100]:

Debaters on comprehensive scientific problems are ... like lawyers who have to take a side. Each of them intends to strengthen his own arguments and to weaken the arguments of the aggressor—but no judge is in the chair. ... Finally we find ourselves all together in the same ship and are co-operating even when we think we are fighting one another.

To be able to cooperate, Neurath suggests common terminology should be based on ‘everyday’ language, derived from physical observation, so that terminology from any specific science can be translated to it [100]. While I agree with Kaptelinin that “it is desirable to rely on a single homogeneous conceptual scheme powerful enough to cover various levels of human-computer interactions” [71], I do not share his (and others’ [80, 81]) optimism that a preexisting framework (e. g. activity theory [74, 97]) can be adopted to this end. Instead, I share Neurath’s view that such a conceptual scheme can only be built from the ground up, based on cross connections (shared concepts, yet with different terms) which can be identified among different research disciplines. For example, within information systems (IS) research, each abstraction level operates on its own models, concepts, methods, and background theories. A common language in the form of ‘transformation procedures’ enables integration between the different subdomains where needed [81], thus not too dissimilar from Neurath’s view of unified science. Similarly, HCI should aim to construct such a common language, devoid of any particular epistemology or philosophy, in order to foster collaboration across disciplines.

3.2 LAYERS OF ABSTRACTION

Categorizing HCI research as the investigation of the interface at different levels of abstraction (or perspectives) is by no means a new proposition [80, 81]. While Grudin discusses the computer is ‘reaching out’, he identifies five different shifts in focus of interface development: the interface at the (1) hardware (2) programming task (3) terminal (4) interaction dialogue (5) and work setting [51]. Inspired by

Constructing a unifying terminology is a cross-disciplinary concern.

Common concepts should be based on a everyday description of human activity.

Prior work has categorized the interface along different levels of abstraction.

this, and in line with IS research, Kuutti and Bannon suggest that the following three levels suffice to describe the current state of affairs: a *work process*, a *conceptual*, and a *technical* level [80, 81]. The work process level considers the relation between the computing system and the context of its use; the conceptual level concerns those parts of the system which need to be understood in order to use it (including conceptual models); and finally the technical level determines *how* the user and computer communicate.

Different research disciplines need to be aware of all levels of abstraction.

Although different perspectives in HCI seem to correspond to “different design domains of an application, each containing also an interface as an integral part” [81], I do not deem it beneficial to emphasize this separation in a framework intended to unify differing points of view. In reality, even though the primary focus of a ‘technical’ level is to enable input and output modalities in a computing system, it can not do so effectively without concerning itself with the surrounding context in which interaction takes place (e.g., a spelling checker or autocomplete function needs to be aware about the language used¹). Conversely, beyond understanding the context of use, the ‘work process’ level needs to concern itself with how a system works (or can work) in order to meaningfully interpret its impact on the surrounding work context (e.g., miscommunication originating from a hastily approved autocompleted word). Different research perspectives are thus much more interrelated than made out to be in the earlier simplification presented in Figure 11. Although the primary focus of each perspective lies on a distinct level of abstraction, all rely heavily upon a basic understanding of the bigger picture.

Different levels of abstraction entail different types of objects in the world.

In line with Neurath’s view, the identified levels of abstraction in this dissertation are based on the observation of ‘physical’ *human activity*, as opposed to being based on a historical perspective [51] or different design domains recognized within HCI [81]. Nevertheless, such earlier demarcations are useful and, as will become clear later on, to some degree overlap with those found when using human activity as the main point of analysis. In the categorization that follows, each level of abstraction denotes a set of observable objects in the world (both actual and desired), related to human activity. The focus on objects is motivated by their essential role as part of user interaction, which forms the second dimension of the theoretical framework presented in this chapter. The purpose of this categorization is not to define intricate details of what each category entails (this is left open to the interpretation of individual perspectives), but to delineate one category from the other. Such a categorization allows to more easily relate findings from separate research disciplines to each other.

¹ To exemplify a failure in doing so: my Nokia Lumia 720 Windows phone retains a system-wide input language setting, requiring me to change it constantly depending on whom I am writing to.

What follows is a decomposition of human activity in terms of the objects it operates on with increasing levels of abstraction. Table 1 lists a few examples of such a decomposition based on different perspectives within HCI. These examples are not exhaustive nor definite, but merely serve to highlight the intended scope of each layer of abstraction. In addition, it is worth noting that not all objects of the same ‘kind’ are necessarily situated on the same layer (e. g., a text editor that allows grouping resources would be a workspace).

MATERIAL: As a fundamental building block, ‘material’ is designed to be indivisible; no interaction readily accessible to the user allows breaking it apart into further subcomponents. The material can only be manipulated as a whole. When manipulated as intended, the material maintains its original purpose or meaning (considering it in isolation). However, as part of a collection, *data material* can gain additional meaning based on its relation to other materials; *instrument material* can not, and serves solely to operate on other materials. This does not exclude data material from being able to operate on other materials, but in contrast to instrument material this is not its primary purpose.

Material traits:

1. *indivisible*
2. *manipulable*
3. *data or instrument*

ITEM: An item is a collection of data and related instrument materials to which direct or indirect access is provided. Each individual data material in the collection gains additional meaning based on its relation to at least one of the other data materials within the same collection. In case the resulting composition is designed to operate on material, it is called a *tool* (configurable instrument); otherwise, it is called a *resource*. Items can overlap with smaller contained items. Although material is designed to be indivisible, the whole can be converted into a collection of materials (generally of a different type than the original), effectively turning it into an item.

Item traits:

1. *collection of data*
2. *divisible*
3. *tool or resource*
4. *hierarchical*

WORKSPACE: From within a workspace, direct access to multiple resources is provided (not necessarily at the same time), and groups of resources can be formed. Manipulation of items is possible through the use of *instruments* (instrument materials or tools) directly or indirectly accessible from within the same workspace. This concept is not restricted to human labor. In other words, the intent behind manipulating items is not included in this definition.

Workspace traits:

1. *group resources*
2. *direct manipulation*

ACTIVITY: Within a workspace human activity takes place, directed towards a certain outcome, during which resources are constructed and instruments are used. Not all resources form part of the desired outcome. Some act as mere intermediate objects; losing them does not negate the outcome. Such ‘meta-resources’ can document the intent, context, and history of human activity.

Activity traits:

1. *uses workspace*
2. *outcome directed*
3. *meta-resources*

	PROGRAMMING	TEXT EDITING	DESKTOP COMPUTING	TASK MANAGEMENT
ACTIVITY	feature/bug report	review bookmarks	thesis writing calendar	project management
WORKSPACE	IDE debugging	library classification	desktop multitasking	calendar schedule
ITEM	statement/class function	paragraph/text text editor	file application	task alarm
MATERIAL	value/operator compiler	character formatting	icon window borders	date time

Table 1: Examples of objects that are part of human activity along different layers of abstraction.

Demarcations between layers are drawn based on the perspective adopted and the intended design.

Some of the terminology used in these dense definitions warrants elaboration. Multiple times it is stated that objects are ‘designed’ for some ‘purpose’, and entail an ‘intended’ use or ‘meaning’. This reflects the nature of the categorization, serving to unify differing perspectives on technology and design. Earlier I stated that any perspective relies on understanding the ‘bigger picture’, which I posit here can be broken down into four separate layers of abstraction. However, where borders between layers are drawn depends entirely on the perspective adopted, and therefore definitions on what delineates one layer from the other necessarily rely on references to how different perspectives make meaning of the world. Within system design this implies that a designer decides on the intended use of individual objects, but can not always anticipate breakdowns (bugs, wear and tear) or unforeseen usage of an object. This might lead to objects being categorized differently once the design ‘breaks down’.

Materials can be manipulated into items, and vice versa.

To further exemplify the distinction between materials and items it is worthwhile to relate the different layers of abstraction to techniques common within the arts. The ‘bigger picture’ for the artist entails both understanding the materials used, as well as envisioning the final piece. Only through practice can the artist gain an understanding of how to navigate the labyrinth in between. To this end, multiple techniques are at hand. For example, both sculpting and electronic music incorporate *additive* and *subtractive* techniques to manipulate objects into unique pieces of art (Figure 12). Using a subtractive technique, a sculptor carves away at stone looking at it as a divisible ‘item’, using a chisel as ‘instrument’ to ultimately reduce it to the ‘resource’ intended. The only limitation is the maximum amount of detail which can be obtained using the given ‘data material’. For a sculptor this is less restricting than for let’s say, a pixel artist², whose only available material is the pixel. On the other hand, using additive synthesis, a musician combines multiple sine waves (‘data material’) to create different timbres for electronic instruments (‘tools’) from which compositions are created (‘resources’).



² A pixel artist edits images on a pixel level. (art by Brian Gogol)

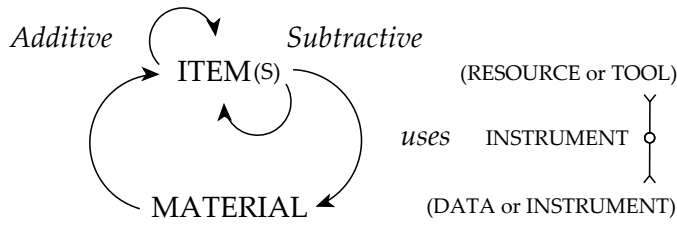


Figure 12: Using instruments, additive and subtractive techniques are used in order to shape objects.

The hierarchical nature of items also warrants further elaboration. Any combination of data and instrument materials is conceivable, but an item (resource or tool) needs to be composed of at least two materials. Similar to Gestalt psychology, the resulting composition “is other than the sum of its parts”³. For example, a multi-bit screwdriver is composed of exclusively data materials: the handle, and the individual bits. Each in separation is not ‘designed’ to be an instrument, but in combination they gain ‘purpose’ and form a ‘tool’. In contrast, a work of art gains ‘meaning’ based on the interrelationships of its containing data materials (e. g., one pixel in relation to another). The whole is a ‘resource’ (all pixels), but a subset of data materials can also be considered a resource in isolation (e. g., the pixels which make up the eyes of the fox in the image depicted earlier). In this sense items do not necessarily ‘contain’ other items, but can more generally be said to ‘overlap’ with others. This distinction becomes important when considering shared items, e. g., a function (‘tool’) in object-oriented programming which is called by multiple classes.

As a composition, materials either gain ‘purpose’ or ‘meaning’.

Other than additive and subtractive techniques, objects can also be reconstructed from the ground up using new materials, generally from a different kind. For example, in image editing software, text can be rasterized (‘converted’ from characters into pixels). The resulting object looks the same, but its makeup changes in the process. Using this technique a single material can also be converted into an item (Figure 13), or an item converted into material (e. g., optical character recognition).

Material and items can be converted, changing their makeup in the process.

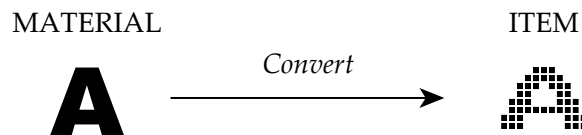


Figure 13: A character (‘data material’) is converted into pixels (‘resource’). For demonstration purposes the quality is reduced, but this is not a requirement.

³ The original statement by Kurt Koffka is often incorrectly translated to “the whole is greater than the sum of its parts”. This deviates from the original intended meaning which simply observes that the whole is *different* and *independent*, not greater.

Design requires continuous moving up and down the ladder of abstraction.

While an object might seem to contain other objects (is divisible), the determining factor on whether or not that makes it an item is its intended design. For example, a dialog box on a personal computer containing a button and a label is still considered ‘data material’ when its sole purpose is to inform the user. However, when in addition it contains an input field acted upon by the application (the input is taken into account), it becomes a ‘tool’. Such lines are harder to draw when considering the *process* of design; although a final design adheres to the four layers of abstraction outlined here, the process of design requires continuous moving “up and down the ladder of abstraction”⁴ [134]:

We stepped up a level of abstraction to see a high-level pattern, and then stepped down to discover the explanation for that pattern. I believe that this dance is where the deepest insights are born—not at any one level of abstraction, but in the transitions between them. . . . To understand a system, we must explore it.

Such exploration requires changing one’s perspective, and thus looking at objects in a different light. Once they become the focus of design, objects previously categorized as ‘material’ can become ‘items’ and vice versa, forcing the designer to start working with a whole new set of resources, instruments, and materials. For example, when the aforementioned dialog box needs to be modified, it can no longer be considered ‘material’. The focus shifts from using the application to user interface design, or possibly even software development. Each different perspective (abstraction level in the physical world) corresponds to a new set of relevant objects and associated classifications which need to be considered.

A workspace supports easy grouping of resources, whereas an item does not.

The distinction between items and workspaces is a challenging one. The key difference lies in being able to *practically* group related resources together. For example, some text editors support having multiple documents open simultaneously, but do not allow creating additional groups beyond the default collection of open documents. Therefore, a simple text editor is considered a ‘tool’ through which ‘resources’ are accessed. The desktop, however, supports having several instances of such a text editor open and is therefore considered a ‘workspace’. The emphasis on ‘practically’, albeit subjective, disambiguates user interfaces which are clearly designed to support resource grouping, from those that require a bit of imagination to achieve the same. For example, it could be said that a text editor supports grouping paragraphs. While this is true, each paragraph is part of the overall text, and the order they are presented in generally carries meaning. This is thus more in line with the definition of ‘item’ than that of ‘workspace’.

⁴ In his essay, Bret Victor refers to different layers of abstraction for understanding complex systems, but the same principles apply here.

Lastly, an activity *always* takes place within a workspace. This is necessarily true because the definition of ‘workspace’ is not bound to any individual interface nor physical constraints thereof. That is to say, a workspace within a computing system is always part of a bigger overarching work environment. For example, when working on a desktop computer the keyboard, desk, paper notes, whiteboard, or any nearby objects can all contribute to achieving the desired goal; the desktop interface thus forms only part of the entire office environment (‘workspace’). Within a single workspace multiple activities can take place, facilitated by the ability to group related resources together. Without resource grouping, activities would quickly become entangled, negatively impacting performance. ‘Meta-resources’ can further help organizing the workspace, clarifying which activity the available resources belong to, who is responsible for them, or any other information not directly forming part of the desired outcome of the activity.

Multiple activities can take place in a single workspace.

3.3 TYPES OF INTERACTION

Having outlined four different layers of abstraction, each building on top of one another, I will now turn to decomposing the second dimension of the interaction framework: *types of interaction*. Similar to the earlier decomposition, each constituting element serves as a unifying concept (enabling cooperation across disciplines), thus should be broad enough to allow for exploration from different points of view, yet specific enough to delineate one category from the other. In addition, as an orthogonal dimension to the first, each category within should be applicable to all four layers of abstraction.

Types of interaction need to be applicable to all four layers of abstraction.

In contrast to the activity abstraction dimension, there is no continuity or logical ordering which can be identified for different types of interaction. Earlier I stated *activity theory* is ill-suited to be used as a unifying framework for HCI, however, as an analytical tool it can be used to highlight the full context of human activity, thus also the full context of user interaction. There is no need to understand the specifics of activity theory at this point; I will merely refer to its central (yet self-explanatory) activity system model to introduce relevant categories of interaction.

Activity theory can be used as an analytical tool for user interaction.

The main unit of analysis in activity theory is the purposeful interaction of a subject with the world. This has also been the focus of the first dimension of the interaction framework. To a large extent the relation between ‘subject’, ‘object’, and ‘outcome’ are encompassed within the different categories part of the activity abstraction dimension (Figure 14). The remaining concepts of the activity system model and their relation to the former, then, can be used to inspire which categories make up the second dimension of the framework. These concepts are ‘tools’, ‘rules’, ‘community’, and ‘division of labor’.

Types of interaction should cover tools, rules, community, and division of labor.

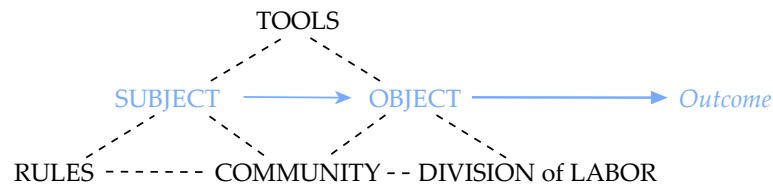


Figure 14: ‘Subject’, ‘object’, and ‘outcome’ in activity theory overlap to some degree with the activity abstraction dimension.

Tools are physical artifacts⁵ mediating human activity. Earlier ‘instrument material’ and ‘tools’ were identified as objects which allow for the manipulation of other objects (categorized based on their composition, collectively referred to as ‘instruments’). When using instruments, objects can be manipulated in two different ways:



CONSTRUCT: The act of bringing objects into existence. On the lowest level of abstraction this pertains to both data and instrument materials. Higher up this pertains to items, workspaces, and lastly activities which can be created through a single interaction, tailored to this specific purpose.



POSITION: Using additive or subtractive techniques, items can be composed from material. While doing so, material itself remains unchanged and is merely positioned to form part of a new or existing item. On higher levels of abstraction, items, workspaces, and activities are positioned (or retrieved) to be used and organized.

Human activity always takes place within a *community*. Even when objectives are not shared, community plays an essential role in achieving one’s personal goals. To this end, two interactions are available:



SEARCH: Searching for objects in a personal work environment is useful, but becomes essential when considering shared work environments. At different levels of abstraction different approaches to search become available, given that different types of information are specified and can be operated upon.



SHARE: Objects can be shared, providing others (or elsewhere) with the means to access and/or modify them. The technology needed and reasons to do so differ between layers of abstraction.

Rules govern the relationship between subjects within a community and the objects to which they have access and of which they make use. To specify rules (regulate), one type of interaction is specified:



CONTROL: Access to, guidelines for, and dependencies between objects can be controlled. Depending on the level of abstraction this ranges from specifying data types on the material layer to workflow specifications on the level of activity.

⁵ Activity theory also considers psychological tools, which are left out here.

Division of labor determines how the objects of activity are distributed and/or shared among multiple subjects within a community. In order to effectively collaborate and/or cooperate two interactions are needed:

NOTIFY: Status changes of objects can be communicated to other objects which might depend on them. In addition, links between objects can be established in order to relate them to each other.



EVOLVE: Understanding the evolution⁶ of how an object came to be can help in handing over and reporting on objects and activities, as well as easily reverting them to a prior state.



Having decomposed both dimensions, the resulting interaction framework is a ‘mosaic’ containing 28 *research topics* each related to a type of interaction at a certain level of abstraction. Based on the matching definitions of both dimensions each tile can be differentiated from one another and entails different questions for research (Table 2).








							
ACTIVITY							
WORKSPACE							
ITEM							
MATERIAL							

Table 2: Decomposition of the interaction framework into 28 different research topics. (Icons designed by Freepik.)

3.4 COMPLETING THE MOSAIC

All that remains is to fill out the framework. A full discussion of each individual cell is beyond the scope of this dissertation, but for each of the activity abstraction levels I will shortly highlight examples which are common to a wide variety of domains. A summary of these is presented in Table 3, providing a coarse overview of different research topics within HCI and how they are related to each other. This framework will form the basis for a discussion of related work in Part ii of this dissertation.

The interaction framework gives an overview of research topics in HCI.

⁶ The notion of ‘development’, understanding how human activity unfolds over time, is central to activity theory.








	 CONSTRUCT	 POSITION	 SEARCH	 SHARE	 CONTROL	 NOTIFY	 EVOLVE
ACTIVITY	context intent	task management	provenance	coordination	workflow	activity life cycle	logging reporting
WORKSPACE	grouping dedicated environment	multitasking suspend and resume	workspace overview view properties annotations	collaboration templating	accounts log in	interruptions	checkpoint replay
ITEM	documents	information curation (keep, manage, exploit)	information retrieval metadata	synchronization conflict handling	authorization	messaging	version control
MATERIAL	input/output devices modalities	selection manipulation	analysis	distribution serialization	protocols types dependencies	links data binding	history undo and redo

Table 3: Examples of relevant research topics for each of the categories within the interaction framework.

Materials are the fundamental building blocks with which users can interact. Different materials offer different *modalities*, each of which requires specific *input and output* technologies so that new materials can be created as well as viewed. By *selecting* one or more existing materials they can subsequently be moved, resized, rotated, or otherwise manipulated in any other way which treats the material as a whole. Searching among existing materials is simplified by using *analysis* techniques (e. g., image search to find images matching a given one). To transfer them (or share with others) they need to be converted into a suitable format to be carried across a transfer medium (e. g., *serialization*). There are different *types* of materials, each with their own capabilities and constraints. Some might *depend* on others in order to be put to good use (e. g., wheels of a car), or might follow certain conventions in order to be understood (e. g., *network protocols*). To enable this, *links* between different materials can be established so they can respond accordingly (e. g., *data binding* in programming). Lastly, a *history* of the construction of material can be maintained in order to allow retracing one's steps (e. g., *undo and redo*).

Items are compositions of material, possibly of different types. For example, text *documents* can be constructed to contain both characters and images. In order to support *information curation* processes (the keeping, managing, and exploitation of information), structuring mechanisms need to be put into place (e. g., folder hierarchies to organize documents). To further facilitate *information retrieval*, additional mechanisms can be made available to the user, like the ability to add *metadata* to documents. When items are shared across devices and/or people, a choice needs to be made on how to *synchronize* them: Can multiple people modify items simultaneously, or is there a need for a locking mechanism so that only one person can edit them at any given time? In case parallel manipulations are allowed, possible *conflicts* which arise need to be resolved. Additionally, *authorization* procedures might be desired to prevent unwanted access to sensitive documents. Similar to links between different materials, but on a higher level of abstraction, links between items can support the exchange of *messages* within a *network* of items. Lastly, through *version control* additional information regarding the history of an item can be specified by documenting particularly interesting points in time.

Workspaces provide access to multiple resources and instruments, allowing to *group* related items together. This supports the construction of *dedicated work environments* which can be *suspended and resumed*, thus supporting *multitasking*. While focusing on a task, *interruptions* external to the workspace need to be dealt with (e. g., through the use of a notification system). When multiple work environments are available, identifying and resuming the one needed can be enabled from a *workspace overview*. Once within a specific work environment, *view properties* can specify how the containing items should be visual-

ized. Additionally, *annotations* on items and the workspace itself can act as visual cues making it easier to find specific items and to document their purpose as part of the workspace. This is especially useful when sharing work environments, supporting *collaboration* both synchronously and asynchronously. Workspaces can also be set up and shared in order to serve as reusable *templates* tailored to a specific community, somewhat acting like ‘instruments’ but on a higher level of abstraction. Similar to items, access to entire workspaces can be restricted to certain *accounts*, requiring the user to *log in*. Lastly, building on top of version control for single items, *checkpoints* for entire workspaces can be specified and progress within the workspace can be recorded, allowing to *replay* work later.

Activities always take place within a workspace, but how work is divided among multiple workspaces is decided by the user; there are no strict imposed relationships between activities and available workspaces. Meta-resources documenting the *context* and *intent* of an activity can be used to formulate plans, which when outlined in time can form part of *task management*. Given this richer available information (the *provenance*⁷ of objects) search is no longer restricted to merely searching for objects based on what they contain, but can rely on who created them, for what purpose, and under which circumstances. By sharing such meta-information (including formulated plans for activities) work can be *coordinated* among colleagues, as well as friends and families. More extensive and possibly recurring plans can be made part of *workflows* of which the execution is controlled and closely followed up. Not all plans are strictly followed. Interruptions can give rise to formulating new goals, switching to different activities, or altogether abandoning ongoing work. This is part of the *activity life cycle* which work environments need to cater to. Lastly, overall progress can be *reported* by carefully *logging* the history of activities.

⁷ According to the Oxford dictionary provenance is the “place of origin or earliest known history of something” or a “record of ownership of a work of art or an antique, used as a guide to authenticity or quality”.

ACTIVITY MANAGEMENT

You know, one of the things that really hurt Apple was after I left, John Sculley got a very serious disease. And that disease, I've seen other people get it too, it's the disease of thinking that a really great idea is 90% of the work. And that if you just tell all these other people "here's this great idea," then of course they can go off and make it happen.

— *Steve Jobs: The Lost Interview* (2012)¹

So far, I have introduced activity-centric computing as a radically new computing paradigm, addressing some of the technological problems inherited from the past. The underlying idea (providing support for *the full context of human activity*) is arguably as old as the personal computer itself, but seemingly got lost somewhere over the course of history. Due to the widespread adoption of files and applications, introduced as part of the first commercially available personal computers, the path for subsequent innovation was all but set in stone; today we have bigger files and bigger applications, to the extent that some applications have now formed their own isolated ecosystems, taking over responsibilities traditionally assigned to the operating system. Most noticeably, window and file management are extended upon in a variety of—at times conflicting—ways by applications that need to manage large amounts of open resources: e. g., web browsers and IDEs. These are indications of a faltering underlying software architecture, which activity-centric computing sets out to address.

Unfortunately, the broader idea of activity-centric computing has remained largely that, an idea. Outside of research there is little incentive to invest in technology which breaks away from the status quo—incompatible with earlier products a company has invested in. Within research, addressing a research agenda which arguably covers the entirety of human-computer interaction (as I will demonstrate here) is outside of scope for short-running research projects with but a handful of researchers working on it. Therefore, most activity-centric computing research focuses on narrowly-scoped evaluations, addressing only part of the underlying vision, and ironically have to build on top of the computing paradigm they are arguing against. Like Steve Job's 'parable of the stones' ([Appendix A](#)), there is "a tremendous amount of craftsmanship in between a great idea and a great product", and we have only just started scratching the surface.

For historical reasons, activity-centric computing never took off.

Research on activity-centric computing is forced to focus on just a subset of the overall vision.

¹ This is the beginning of Steve Jobs's reply to the question, "What is important to you in the development of the product?", in an interview 10 years after he left Apple and two years before his return (1995). The full question and answer, including the inspiring 'parable of the stones' about teamwork can be found in [Appendix A](#).

A set of principles underpin most work on activity-centric computing.

That is not to say that no discernible progress has been made. As one of the most active researchers in this line of research, Jakob Bardram has formulated a set of key principles which have guided the design of several activity-centric computing systems over the course of the past decade. These were based on the design of a pervasive healthcare system accounting for “mobility, many interruptions, ad-hoc collaboration based on shared material, and [work which is] organized in terms of well-defined, recurring, work activities” [36]. In subsequent publications the principles were introduced [15, 17], elaborated upon [16], and more recently presented alongside a summary of 10 years of research on activity-based computing (ABC) [20]:

1. **ACTIVITY-CENTERED:** Work is organized into activities, which are higher-level computational constructs that encapsulate all resources, tools, and communication mechanisms into one goal-oriented interaction model. By moving away from classic application-oriented interfaces to multi-device activity-oriented workspaces, users are presented with logical units of work combined with the tools required to perform that work.
2. **ACTIVITY MULTIPLEXING:** By supporting activity suspension and resumption, users can easily switch between different activity contexts. Suspending an activity means its state is stored and removed from the active workspace, while resuming an activity restores it. This feature supports parallel activities (multitasking) and interruptions in work.
3. **ACTIVITY ROAMING:** Activities are stored in an infrastructure and hence can be accessed from multiple devices. This allows a user to suspend an activity on one device and resume it on another, thereby allowing the user to roam between devices. The context of an ongoing activity can also be spread across devices, allowing for multi-device interaction on one common activity context. Users are presented with awareness cues and overviews on the distributed state and accessibility of the activities.
4. **ACTIVITY ADAPTATION:** Activities adapt to the capabilities of the device(s) on which they are resumed. Hence, an activity might look quite different whether it is resumed on a wall-sized display or on a smartphone. A subset of an activity’s context can be displayed when it is spread across several devices.
5. **ACTIVITY SHARING:** Because activities are distributed, they can also be shared among users. Shared activities can be accessed and modified by all related participants. Accessing activities simultaneously allows for synchronous collaborative setups. Alternatively, asynchronous exchange of information is possible when separate users suspend and resume an activity. By attach-

ing messages or other objects to the activity, all related participants are notified of changes, thus providing users with awareness about what changed and on who is working on what.

6. **CONTEXT-AWARENESS:** Since activities are computational constructs that transcend a single user or device, they need to be aware of their usage context such as location, type of device, amount of users and other factors. The process of detecting, selecting and managing the activity and its resources is a semi-automatic process involving both the users as well as automated sensing and inferring of context information.


With the interaction framework in place ([Chapter 3](#)), I can now position Bardram's activity-centric computing principles within a wider research agenda for [HCI](#). But first, I will categorize different types of tools in support of *knowledge work*. By positioning these alongside the [ABC](#) principles, it will become clear how activity-centric computing is related to other lines of research, and more specifically, how its underlying vision tries to integrate different types of interaction which are generally supported by independent tools. Furthermore, a clear goal for activity-centric computing is outlined in regards to which additional types of interaction still need to be integrated in order to cover the full scope of knowledge work.

A goal for activity-centric computing can be formulated within the interaction framework.

4.1 INFORMATION FRAGMENTATION

Within the interaction framework, different categories of interactive computing systems supporting multiple types of interaction can be recognized ([Table 4](#)). By no means are these categories the sole ones imaginable, nor do they define strict boundaries in between computing systems, however, they do provide a coarse overview of different functionalities which are commonly bundled together within specific software systems. For example, a text editor (an *editing* tool) supports manipulating and searching for text, maintaining a history of edits (thus supporting undo and redo), and in some cases incorporates version control. However, more advanced operations such as data binding and restricting the possible values for data fields are generally only available in *programming* languages (although spreadsheets do support this to some extent). *File and window management* come prepackaged as separate subsystems in most operating systems, but no such support is provided for *task management* which requires adopting third party tools such as electronic calendars or productivity tools. To share and back up files *cloud storage* is commonly used nowadays. However, further cooperation requires employing *communication* tools, or *collaborative workspaces* which support editing shared documents simultaneously. Finally, some more advanced *workflow management* systems allow automating entire business processes.

Most software systems provide support for a set of different interactions.



ACTIVITY	Task management	Workflow management		
WORKSPACE	Window management	Collaborative workspace		
ITEM	File management	Cloud storage	Communication	Editing
MATERIAL	Editing	Programming		

Table 4: Different categories of computing systems positioned within the interaction framework.

Information fragmentation is caused by systems operating on just one level of abstraction.

As can be seen in Table 4, most software systems operate on just one layer of abstraction within the interaction framework. Although some more advanced computing systems that tailor to very specific practices do integrate different types of interaction from separate layers within the framework (e. g., IDEs and ERP systems), general purpose systems typically do not. These are the tools predominantly used by knowledge workers, involved in producing, transforming, consuming, and communicating large amounts of diverse information². While a wide range of specialized tools do allow users to select those most suited to their work, it also introduces *information fragmentation*: for each tool information items are stored, managed and viewed within separate application-specific collections [25]. Since studies have shown that information is often organized within hierarchies reflecting the users' projects or tasks [25, 28, 56], and a single project can make use of several tools, it is left to the user to associate the different related information items and maintain consistency across individual project hierarchies. These problems are exacerbated within distributed and collaborative environments, where information in addition can be fragmented across multiple devices and collaborators. Users need to decide where to store data, how to transfer it across devices, and how to share it with others [116].

Activity-centric computing integrates tools around the central notion of 'activity'.

What unifies different tools is the actual work they set out to support. Studies have shown that people organize their work in higher-level thematically connected units of work, often referred to in literature as tasks or activities [11, 37, 49]. By constructing tool integration around a computational representation of activities, the redundant work of managing them within individual computing systems can be offloaded to a centralized *activity management* system. This integrative aspect is a key characteristic of activity-centric computing which can be recognized in most of its principles (Table 5). In activity-centric computing an activity is "a computerized representation of a real-world human activity" [16]:

Computational Activity. An aggregation of services, resources, artifacts, and users that are relevant for a real world human activity.

² The following argumentation is largely adopted from my UIST publication [64].

4.2 COMPUTATIONAL ACTIVITIES








							
ACTIVITY							
WORKSPACE	1. Activity-centered	2. Multi-plexing	4. Adapt.	5. Sharing			
ITEM			6. Cont.-aw.	3. Roaming			
MATERIAL							

Table 5: The activity-centric computing principles represented within the interaction framework: (1) activity-centered; (2) activity multiplexing; (3) activity roaming; (4) activity adaptation; (5) activity sharing; (6) context-awareness.

A central feature of activity-centric computing is for users to be able to *assign intent to context*, i. e., to be able to construct computational activities. Therefore, the user first needs to be able to aggregate separate items into collections which make up said context, represented within the interaction framework as the construction of workspaces. The key distinction between ordinary workspaces and those that are activity-centered lies in how well the system supports aggregating the full context of human activity, thus including a near-exhaustive set of different types of ‘items’, e. g., documents, web pages, emails, contact information, and any other relevant information.



In case the user assigns one activity per workspace, switching between activities is just a matter of going from one workspace to the next; there is no longer a need to retrieve individual items. Moreover, when items are persisted within the workspace (and thus activity), file management becomes integrated with multitasking; in order to access files, all one needs to do is switch to the corresponding activity. Although not formulated as part of the original multiplexing principle, even further integration can be achieved by supporting rich organization of said activities, thus effectively integrating task management with file management as well.



The activity adaptation principle is not so much about integrating separate tools, as it is a desirable property for any workspace to have, whether it is displaying items related to one particular activity or any other information. Allowing users to switch between different views on the same information provides them with the opportunity to tailor their work environments to specific tasks or circumstances. Part of this process can be automated by making computing systems context-aware. However, context-awareness is not restricted to activity adaptation and can be applied to any of the interactions that are specified on the workspace layer (through analysis of objects on the items layer): e. g., automatically assigning relevant items to a workspace, automated log in, and filtering irrelevant interruptions.





Activity roaming
and sharing

Activity roaming and activity sharing are highly interlinked, the former enabling the latter. In unison these two principles support working on synchronized files within collaborative workspaces. This integration is not unique to activity-centric computing; distributed workspaces can just as well be used to work on contexts other than one individual activity and rely on the same technologies in order to do so. However, by associating one activity per workspace and sharing additional information in regards to these—who scheduled them, for how long, and who else is working on them—shared workspaces could further be integrated with the coordination mechanisms needed to manage the activities they represent. So far, however, this has not been researched within activity-centric computing.

Activity-centric
computing needs to
integrate workspaces
and items into one
activity context.

By outlining the ABC principles within the interaction framework (Table 5) it becomes clear that activity-centric computing has thus far focused primarily on setting up workspaces, generic enough to encompass the full context (all ‘items’) of human activity, and to support interactions on them *as a whole* (e. g., sharing, and suspend and resume). This work can be particularly challenging, given that it needs to build on top of a technological stack essentially designed to do the opposite: each workspace is designed to manage just a specific subset of items. Therefore, activity-centric computing needs to integrate different existing workspaces into one overarching ‘activity context’ workspace, covering the full range of items and available interactions (Table 6). By doing so any material can be accessed from within the full context of the activities it belongs to, addressing the problem of information fragmentation. The ‘control’, ‘notify’, and ‘evolve’ interactions, however, have thus far been overlooked (Table 5).

Activity
management needs
to be supported.

More importantly, insufficient support for *activity management* (the full range of interactions represented on the top ‘activity’ layer) is likely to have resulted in some of the open issues encountered within activity-centric computing [20]: organizing activities is error-prone since their contexts easily intertwine, and a large number of activities need to be managed once all interactions occur within the context of activity. What is needed more than mere labeling of activities (ascribing intent to a workspace), is providing support for the full set of interactions incorporated within tools currently supporting *task* and *workflow* management (Table 6).

ACTIVITY	Activity management (task & workflow management)						
WORKSPACE	Activity context						
ITEM							
MATERIAL							

Table 6: The full range of interactions activity-centric computing should address.

Prior activity-centric computing systems seem to assume that commencing work on an activity precedes or coincides with defining it. In reality, however, users often define loose goals only to start working on them somewhere in the near future. The temporal aspects of when and how activities are defined and how they evolve over time warrant further exploration. Furthermore, any long-term deployment of an activity-centric computing system (even for a single user) would need to consider how to store the large number of activities that users accumulate over time. This is reflected in the open issues listed as part of a review on activity-centric computing, clearly indicating a need to provide better support for the organization of activities [20]. Therefore, the full set of interactions which currently make up task management need to be incorporated into activity-centric computing.

Because of the emphasis of activity-centric computing on supporting users to construct computational activities—as opposed to *prescribing* them—workflow management systems have generally been considered a separate class of systems [36]:

In our view, a human activity precedes the computational activity that mirrors it, whereas in a workflow system the computational activity precedes and dictates the human activity.

However, some overlap with workflow management becomes inevitable as part of sharing activities with other users. This implies an activity context is already set up, thereby necessarily prescribing human activity to some extent. Therefore, a logical continuation of the overarching vision behind activity-centric computing is to incorporate features from workflow management. This, however, requires improving support for *task management* first, which will be the focus of this dissertation.

Scaling up activity-centric computing relies on task management.

Sharing activities relies on workflow management.

Part II

DESIGN AND TECHNOLOGY

The so-called ‘desktop metaphor’ of today’s workstation is instead an ‘airplane-seat’ metaphor. Anyone who has shuffled a lapful of papers while seated in coach between two portly passengers will recognize the difference—one can see only a very few things at once. The true desktop provides overview of and random access to a score of pages.

— *No Silver Bullet*, Frederick P. Brooks, Jr. [31]

There is a noticeable overhead associated with managing many different parallel activities, which can easily lead to information overload [95]. By contextualizing information—hiding irrelevant data while pushing important information to the foreground—this problem can be alleviated. Different areas of research are trying to provide the user with contextualized information, each approaching the problem from a slightly different angle. PIM research tries to empower users by providing extensive tool support by which to manage and access their information. HCI follows a more user-oriented approach, where the user is usually placed central during system design. Lastly, artificial intelligence (AI) research focuses on making systems ‘context-aware’ through the use of activity recognition, allowing the system to respond to ongoing activities. In short, PIM, HCI, and AI emphasize *information*, *users*, and *automation* respectively, but all address a common problem—information overload.

The topic of this dissertation is *activity-centric computing*, an approach within the field of HCI to contextualizing information around the central concept of ‘activity’. In this line of research, what defines a concrete activity is user-specific, since it relies on which intention is expressed by or is meaningful to the user. The underlying goal is to provide a computing platform for users which allows them to focus on the high-level collaborative activities they engage in, rather than being forced to deal with low-level details such as deciding on specific software applications or retrieving necessary resources when resuming activities. In this section, I will provide an overview of various systems in this line of research, which have been introduced over the course of the past 10 years¹. First, however, given the lack of support for *task management* in earlier described work, I will investigate how task management is typically supported in various personal information management tools².

Contextualizing information is a common approach to addressing information overload.

Activity-centric computing supports user-defined computational activities.


¹ This overview of related work is largely based on an earlier publication [20].

² The discussion of task, window, and file management is adopted from an earlier publication [64].

5.1 TASK, WINDOW, AND FILE MANAGEMENT

Task, window, and file management are an integral part of knowledge work.

This dissertation focuses on how *task management* can be integrated into activity-centric computing. Therefore, in order to analyze how task management is interrelated with underlying tools in support of knowledge work, I will review prior literature on PIM tools supporting the management of tasks, windows, and files (Table 7). These are used at different points in time throughout the life cycle of an activity, thereby providing a better understanding of how activities evolve over time. In addition, this can highlight conflicts where functionalities of tools overlap with one another. Such conflicts need to be addressed within a conceptual model serving as the basis for design of systems in support of *integrated knowledge work*.



ACTIVITY	Task management					
WORKSPACE	Window management					
ITEM	File management					
MATERIAL						

Table 7: Task, window, and file management are an integral part of knowledge work, generally supported by independent tools.



Task management

Tasks in *task management* are best defined as a set of actions which need to be performed in order to achieve a certain goal. Within a desktop environment, a single task usually employs several different application windows and documents. *Window management* helps to organize these during multitasking, but once a task is discontinued and the associated applications are shut down the window configurations set up by the user are lost. To reconstruct the work environment users need to reopen all related documents and reposition application windows for optimal use. Therefore, users often aggregate documents related to one particular task in task-specific folders [25, 28, 56]. This prevents them from having to browse to several different folder locations within the *file system* when resuming a task. Similarly, some systems support creating *task-centric resource collections* within existing applications. Taskmaster [24], e.g., recasts email as a task management tool by allowing users to aggregate email threads within task-centric collections called ‘thrasks’. Task Gallery [112] allows users to organize windows and files within pieces of artwork hung on the walls of a 3D virtual art gallery. Mylyn [76] is an extension for the Eclipse programming environment which allows users to switch between programming tasks and automatically builds up their context based on source files users interacted with. Most of these approaches, however, are application-specific and only support aggregating a limited set of document types. TAGtivity [106], on the other hand, allows

tagging any resource on a personal computer. While tags don't necessarily have to represent tasks, they were shown to be used as such, allowing the user to easily retrieve files of different work contexts. TaskTracer [40] monitors the user's interaction with a personal computer and automatically builds up multiple task contexts, of which an overview is subsequently shown to the user. What many of these tools neglect, however, is that *planning* future events plays a crucial role in knowledge work as well. The user needs to be able to prepare, structure and reflect on future work. The widespread adoption of electronic calendars, to-do lists, and other productivity tools indicates their value as part of task management. These tools, however, are intrinsically disconnected from the actual resources needed to start work on future tasks [132].

Window managers are tools that allow users to handle an increasing number of windows by grouping them together spatially, leveraging human spatial memory when they need to be retrieved. By allowing users to switch between separate window groups, *multitasking* is supported. Many modern window managers (such as those available in Windows XP up to Windows 8) automatically group windows from the same application together under one item on the taskbar. Although this reduces taskbar clutter, a task might rely on windows from several different applications. Therefore, GroupBar [124] extends on this idea by providing the user with more control; windows belonging to different applications can be grouped together arbitrarily on the taskbar. Overall, existing window managers focus on similar lightweight support for *task management*. Rooms [55] was one of the earliest systems to do so and popularized *virtual desktops*. Using virtual desktops work can be distributed across several different dedicated workspaces, each showing only those windows opened from within one specific desktop environment. A similar feature is now available in most contemporary operating systems and has most recently been introduced in Windows 10 as the 'task view'. Other window managers explore alternate visualizations for window grouping. One such example is Scalable Fabric [113]; windows shrink in size when moving them into the periphery and can be organized within labeled groups. WindowScape [128] offers implicit task management by storing snapshots of short-lived window groups (used in unison) which can be returned to at a later moment in time. Elastic Windows [70] allows organizing windows hierarchically and can perform operations on window groups, e. g., hiding an entire window hierarchy. Spatial arrangement, however, does not need to be limited to 2D space, e. g., BumpTop [5] allows dragging and tossing objects around in 3D space, extending even further on the physical attributes of the desktop metaphor. However, long term task management (like planning events ahead of time), requires users to install separate task managements tools.



Window
management



File management

Whittaker describes three *information curation processes*: the keeping, management and exploitation of information. Based on how information is stored, there are four main strategies of accessing information: navigation, search, orienteering (hybrid of navigation and search), and tagging [138]. People preserve large quantities of emails, bookmarks, and personal files and organize them in ways that will promote future retrieval. However, information items are stored, managed, and viewed from within application-specific collections, causing *information fragmentation* [25]. Not all information items are purely informative. Some are *action-oriented*, like emails. They require the recipient to do something, possibly before a certain date, and are generally kept around as visual reminders. In order to alleviate the overhead of organizing files, time-based organization has been brought forward in previous research. Lifestreams [46] was the first to do so, providing a simple stream of documents over time. TimeScape [110] extends on this by integrating with the desktop interface. Time travel allows going to past and future states of the desktop. Only those documents used or placed at a certain point in time show up. Other approaches, like Haystack [1], offer the ability to construct personalized information collections by creating relations between separate information items. To simplify file retrieval, Leyline [47] offers information about the creation, use, and context of documents, including whom documents have been shared with—their *provenance*. Although such tools do provide improved access to the underlying file system, in essence files are a remnant of the original desktop metaphor. Users are forced to mentally connect window representations to the files they represent. Therefore, when resuming a task users first need to find all related files and restore their window configurations prior to commencing any actual work.

Task, window, and
file management are
intrinsically
interconnected.

It is clear that task, window and file management, each a fundamental part of knowledge work, are three practices intrinsically linked to each other. As depicted in Figure 15, conflicts arise where their supported functionalities overlap. To sum up and add to some of the conflicts highlighted earlier: window managers support lightweight *multitasking* but are unaware of the tasks specified in traditional task management tools; *interruptions* often lead to commencing new tasks or revisiting old ones [67], which requires (re)opening the relevant application windows; *task-centric resources* are often grouped together in the file system with folder hierarchies acting as kind of project plans [69]; *action-oriented* items (like emails or temporary files) are left behind as reminders [138]; email has been shown to be the primary source of new events entered in a calendar, and calendars themselves are not only used for planning, but also to report on finished projects and to record memorable events [132]; lastly and most prominently, *information curation* is handled differently per application, causing *information fragmentation*.

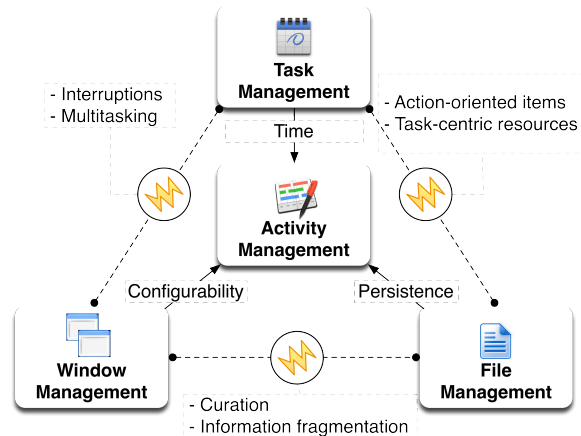


Figure 15: Overview of task, window, and file management, highlighting conflicts which arise between them [64].

What unifies these different practices is the actual work they set out to support. Studies have shown people organize their work in higher-level thematically connected units of work, often referred to in literature as *tasks* or *activities* [11, 37, 49]. By constructing tool integration around computational representations of activities, the redundant work of managing them within individual computing systems can be offloaded to a centralized *activity management* system. Such a system needs to integrate with the core aspects of task, window and file management; activities evolve over time, should be easily configurable, and should be persisted so their context does not get lost once the system is shut down (Figure 15). Within prior activity-centric computing systems, files, applications, communication, and collaboration can be structured within computational representations of activities. However, with the strong emphasis on *activity context* in this line of research, activity management has thus far received little attention.



Activity management

5.2 DESKTOP SYSTEMS

Since the seminal work on Rooms [55] many desktop systems supporting activities have been described. UMEA [72] is a first example of a system that monitors the user's behavior within self-defined 'projects'. Activity context is built up automatically within the currently selected activity, providing the user with an overview of their interaction history. TaskTracer [40] (Figure 16) and CAAD [108] use advanced data collection frameworks to monitor the user's interactions, from which activity representations are automatically created. Although the work of defining activities in TaskTracer is automated, manual construction of activities is supported as well. Activity Explorer [96], the Activity Bar [17], and Giornata [136] are examples of systems that fully rely on users to define meaningful activities.

Context can be constructed in various ways.

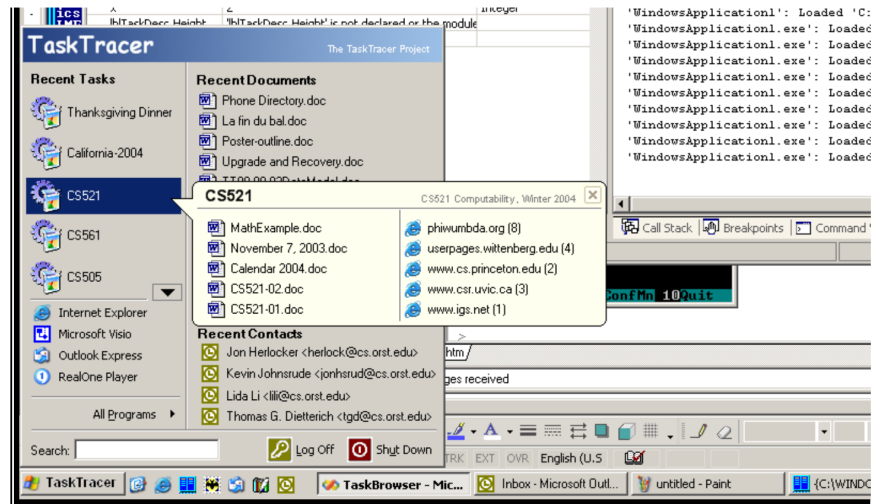


Figure 16: TaskTracer for Windows XP [40].

Different degrees of integration with traditional computing are possible.

Giornata (Figure 17) and the Activity Bar re-frame the desktop interface for personal computers to be activity-centric, for OS X and Windows XP respectively. They provide activity-centric management and sharing of context like windows, files and contacts by integrating with the traditional desktop operating system, adding or modifying existing components from the desktop user interface. The user can switch between different ongoing activities by suspending and resuming them, thus facilitating multitasking. In contrast, Activity Explorer [96] resembles an email client, less integrated with the operating system, but was the first system to show that you can meaningfully structure communication and collaboration processes within shared representations of human activity.

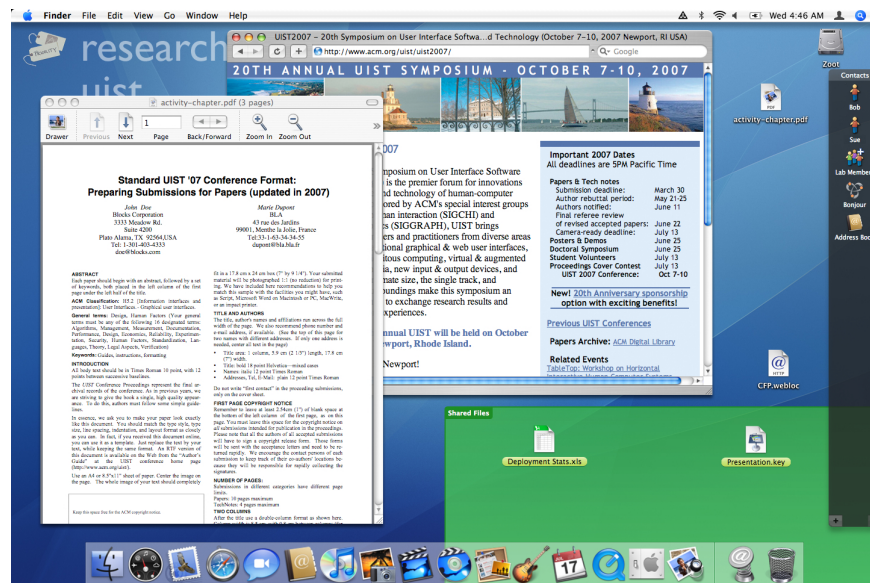


Figure 17: Giornata for OS X [136].

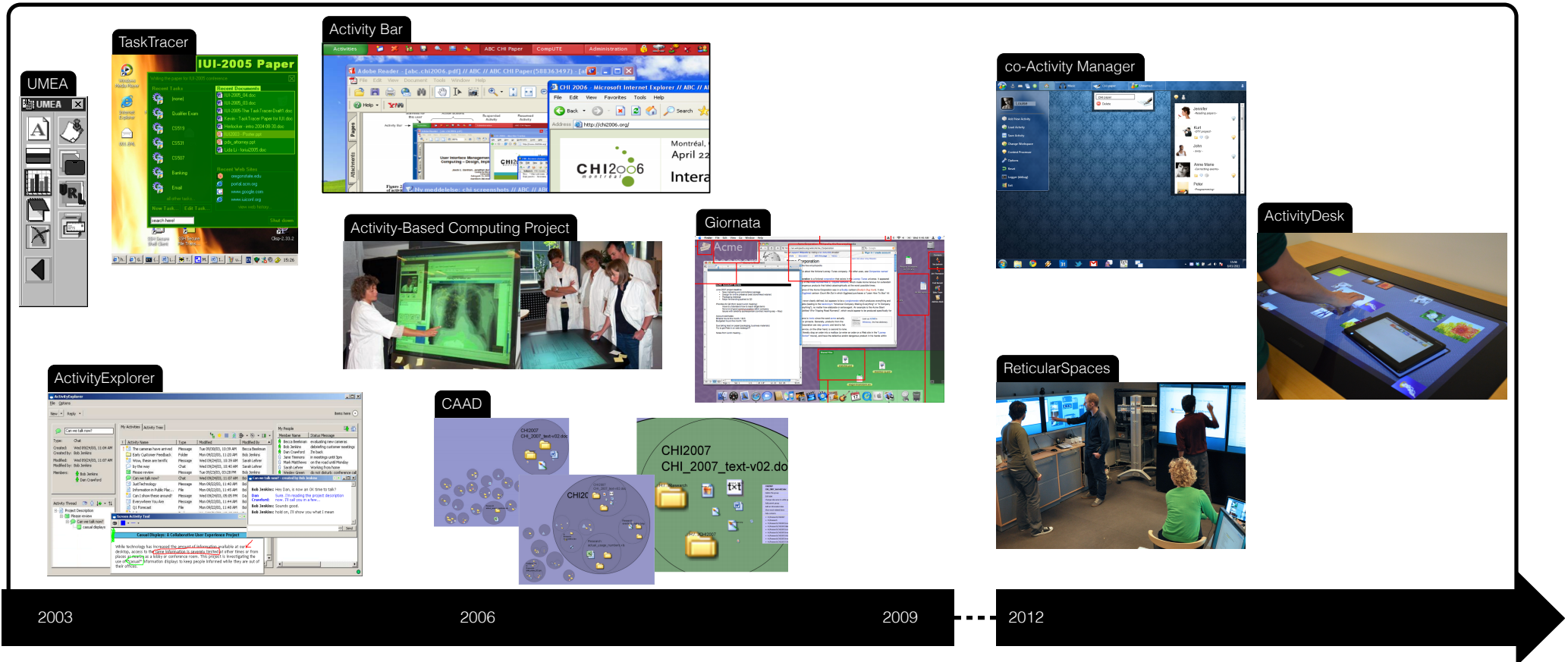


Figure 18: A historical overview of activity-centric computing since 2003 [20].

Supporting the activity life cycle remains an open issue in activity-centric computing.

The main technological contributions introduced in this dissertation—Laevo and co-Laevo (Chapter 6)—can be positioned within this category of activity-centric *desktop computing systems*. The most recent work in this area is co-ActivityManager (cAM) [58], which extends on the concept of the Activity Bar. cAM supports three main features: (1) dedicated activity workspaces; (2) collaboration through activity (and resource) sharing, status updates, and online communication; and lastly, (3) activity management through an activity task bar (Figure 19). The goal of cAM is to integrate communication and collaboration channels more explicitly into activity-centric computing. The system was deployed for a period of 14 days in a multidisciplinary software development team. An analysis of the activities created by users showed different levels of granularity of goals and time spans used. Participants organized activities in three categories: (1) ad hoc activities (e.g., a to-do), (2) short-term activities (e.g., day to day work), and (3) long-term activities (e.g., ongoing collaborative projects). Despite these insights into how users appropriate activities of varying scope, it is still unclear how these different activities relate to each other and how they fit into the entire shared *activity life cycle* of daily work. As reflected in the research question of this dissertation (Section 1.1), this thus remains an open issue within activity-centric computing, even in most recent work. To this end, Laevo and co-Laevo provide explicit support for activities of varying levels of granularity.

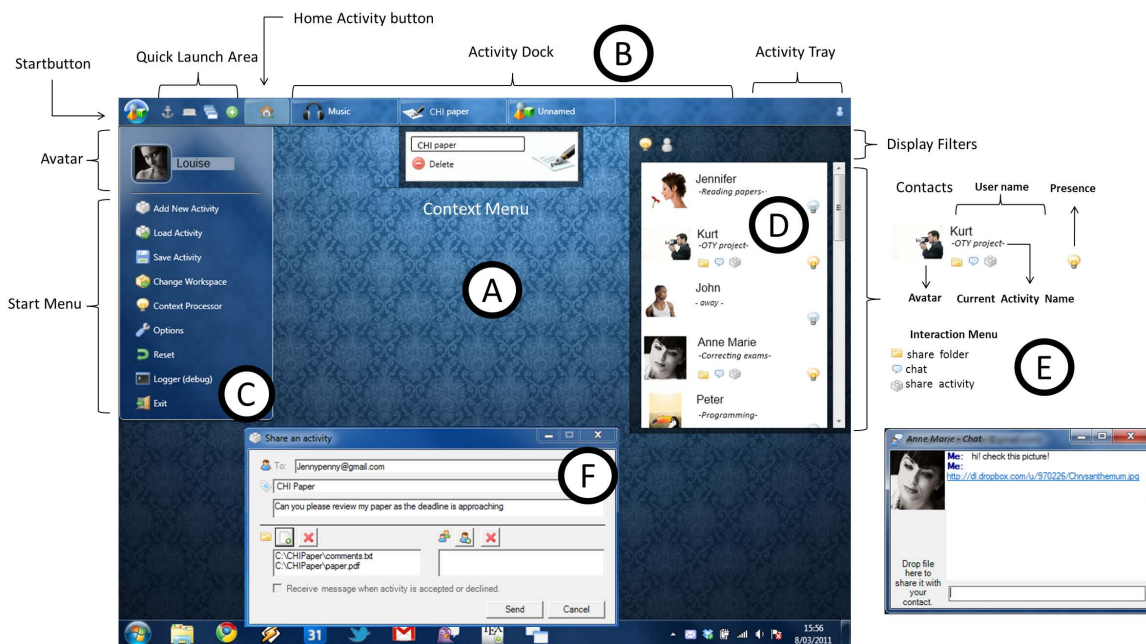


Figure 19: co-ActivityManager [58] is an activity-centric desktop computing system incorporating: (A) activity workspaces, (B) task bar to visualize activities, (C) start menu to manage activities, (D) a collaboration manager through which (E) communication can be initiated and (F) activities can be shared.

5.3 UBIQUITOUS COMPUTING SYSTEMS

Although not the focus of this dissertation, it should be noted that activity-centric computing has been applied to other domains than desktop computing. A major focus has been on ubiquitous computing and distributed user interface systems, demonstrating the merit of activities as a context model in multi-device environments (Figure 20). When moving away from a single user/device scenario to a more pervasive environment many of the multitasking and interruption problems are greatly amplified.

A major body of work on activity-centric computing was performed under the name of activity-based computing (ABC), and took its outset in the design of a ubiquitous computing infrastructure to support the nomadic, collaborative, and time-critical work in hospitals [16, 36]. This research introduced an activity-based infrastructure consisting of distributed middleware and specialized user interfaces optimized for medical work. All services, applications, and resources related to patient care are bundled in distributed activities. For example, medical records, information on the medicine administered, and medical images for a patient are linked together in an activity, which is shared across clinicians involved in the patient's treatment and care. This infrastructure was extended to incorporate large interactive displays [18] and automatic activity detection [38], resulting in increased activity awareness.

Problems are exacerbated when multiple devices and users are involved.

Ubiquitous activity-centric computing has been applied to hospital work.

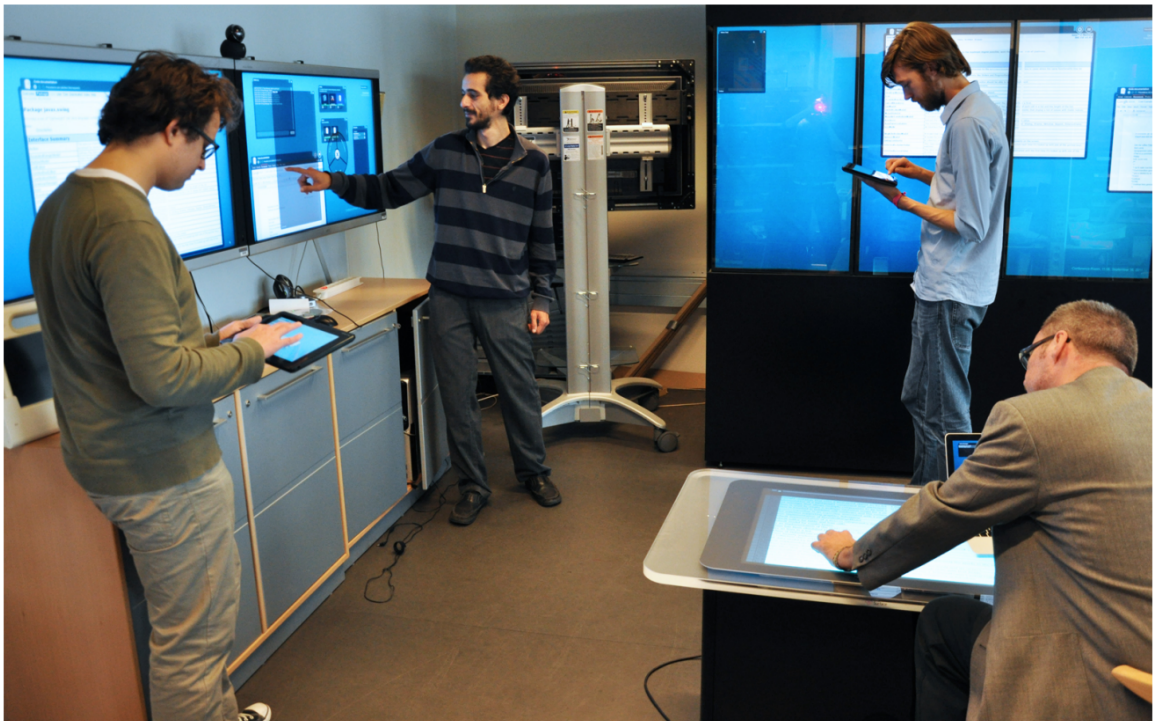


Figure 20: ReticularSpaces is an activity-centric smart space environment [19].

Activity representations should be tailored to the specific form factor of devices.

More recently, this research was extended upon to create support for distributed activities across multiple devices in the ReticularSpaces infrastructure [19]. ReticularSpaces is an activity-based smart space system designed to support: (1) unified interaction with applications and documents through ReticUI, a novel design for distributed user interfaces; (2) management of the complexity of tasks between users and displays; (3) mobile users in a local, remote, or nomadic settings; and (4) collaboration among local and remote users (Figure 20). ReticularSpaces was deployed in a smart space environment and was exposed to end-users through a scenario-based evaluation. The study showed that users found the use of activities intuitive in a multi-device context as it allowed them to move tasks between devices and collaborating users. However, it also highlighted a number of open issues with device-specific visualization of activities and cross-device interaction. While the ReticUI interface duplicated the same user interface on all devices with only small adaptations, users argued that this adaptation should go much further and interfaces as well as information density should be much more tailored to the type of device.

Working on an activity using multiple devices simultaneously is cumbersome.

MULTI-DEVICE INTERACTION Mobile devices, such as smart phones and tablets, have become an intrinsic part of people's everyday life. Alongside laptops and desktop computers, these devices have become part of a device ecology in which each device acts as a specialized portal into the overarching information space of users. There is no longer a one-to-one relation between users and devices, but rather a one-to-many, or even many-to-many relationship. This introduces a configuration overhead when users have to access the same activity from multiple devices [57]. ActivitySpace [60] is an early prototype providing support for the management of multiple devices in an office environment (Figure 21). The main goal of ActivitySpace is to reduce the configuration work required to use multiple devices simultaneously by using an interactive desk as a medium to move and share activities across devices.

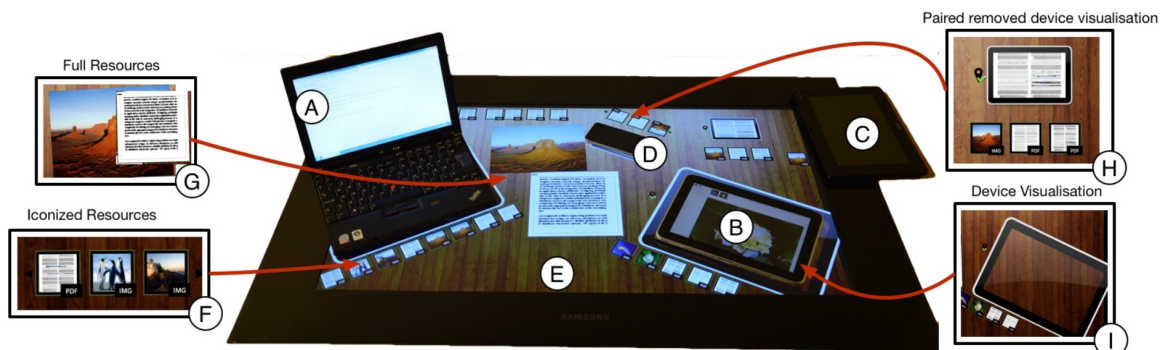


Figure 21: ActivitySpace supports moving and sharing (F, G) resources across (A-E) multiple devices. Placeholders are shown (H, I) when devices are removed. [60]

6

LAEVO AND CO-LAEVO

*levo lēvo (laevo), āvi, ātum,
... to make smooth, to smooth, polish
levo, lēvo, āvi, ātum,
... to lift up, raise, elevate*

— Lewis & Short, *A Latin Dictionary* (1879)¹

In this chapter I will introduce *Laevo* and *co-Laevo*²: activity-centric computing systems which integrate workspaces with task and workflow management respectively (Table 8). *Laevo* is the result of an elaborate design process and has been evaluated during a two-week in situ field study by various knowledge workers [64]. The system is based on an analysis of three fundamental knowledge work practices (task, window, and file management), presented earlier in Section 6.1, giving rise to a conceptual model for design depicting the different states and transitions of human activity over time—the *activity life cycle*. *Laevo* is primarily designed to function as a personal information management (PIM) system, but is consistent with the broader scope of activity management. To demonstrate, *co-Laevo* is a conceptually coherent superset of *Laevo*, extending on the design to incorporate support for cooperation [65]. However, *co-Laevo* is incomplete in regards to providing support for the full set of interactions which make up workflow management. As part of this dissertation, *co-Laevo* mainly serves to indicate how the conceptual model from *Laevo* can easily be extended on in order to provide support for the full scope of human activity within a general purpose computing system.



Laevo and co-Laevo integrate dedicated workspaces with support for activity management.








	  			   			
ACTIVITY	Task management			Workflow management			
WORKSPACE	(LAEVO)			(CO-LAEVO)			
ITEM							
MATERIAL							

Table 8: *Laevo* and *co-Laevo* integrate workspaces with task and workflow management respectively, which as a whole constitutes activity management.

¹ I found this old latin translation for the verb ‘relieve’ through Google Translate, which I personally feel aptly describes the intent behind activity-centric computing: to relieve the user from the complexities of managing multiple activities by ironing out issues of interoperability between independent tools.

² These systems will largely be presented as they were in earlier publications [64, 65].

6.1 ACTIVITY LIFE CYCLE

The activity life cycle is a conceptual model for activity management.

As I conclude in [Chapter 4](#), prior research on activity-centric computing has mainly focused on which information and which services the *activity context* is comprised of, based on theories of cognition and observations of real-world practice. However, the previous literature review clearly indicates that investigating the *transience of information* is equally important; where does information come from, how is it passed from one tool to another, and when is it no longer needed. Within the context of activities this means investigating how the activity context is created, used, and how it changes over time. As concluded in the evaluation of Giornata “[this] raises further research questions about how individuals think about the stages in an activity’s lifecycle” [135]. Here I introduce the *activity life cycle*: a conceptual model encompassing key concepts for *activity management* ([Figure 22](#)). This model represents the main interactions that influence the state of activities over time and relates them to three fundamental processes of knowledge work: *multitasking*, *archiving*, and *planning*.

Multitasking, archiving, and planning correspond to window, file, and task management respectively.

Multitasking is a common process which involves switching between and managing a large number of open windows, files, and other resources that are associated with different activity contexts [37, 49, 67]. Within activity-centric computing this is supported by allowing users to aggregate related resources into meaningful structures reflecting their ongoing parallel activities and providing support to easily switch between them. Restoring the entire work context is automated, minimizing reconfiguration work users would otherwise have to do manually. Multitasking is traditionally supported by window managers. File management, on the other hand, primarily supports *archiving*. Malone identified two important practices which describe how office workers physically organize their information on their desks and in their offices: filing and piling [87]. Following this categorization, structuring work within labeled activities allows for the systematic ordering of information—the equivalent of filing. In con-

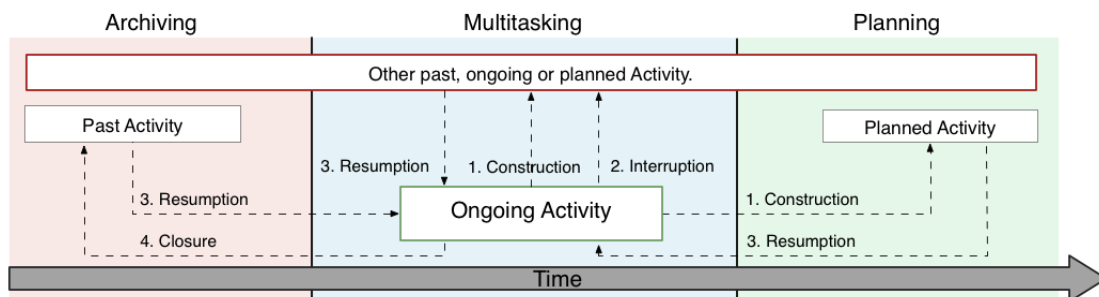


Figure 22: Modern knowledge work consists of archiving, multitasking and planning. Four fundamental practices, related to these processes, determine how an activity evolves over time [64].

trast, while working on an activity, documents can easily be piled within the current open activity context, automatically associating the two. Following ‘overview first, zoom and filter, then details-on-demand’ [122], users can search through filed activities and subsequently retrieve the associated piles (activity contexts). Lastly, task management constitutes short- and long-term *planning*, an essential part of knowledge work which shapes future activities. While planning activities, part of the context can already be built up in order to be exploited once work on the activity finally commences.

Time ties archiving, multitasking, and planning together. It is an integral part of planning and a useful reference when retrieving archived activities. Furthermore, given that users already have a mental model of how activities occur in time, the temporal dimension is an extremely useful construct to be made part of a conceptual model for activity management. Considering time, activities can be in three different states: *ongoing* when part of the current multitasking session, *past* when work has been discontinued, and *planned* when intending to work on them at a later point in time. Four practices influence the state of activities over time (construction, interruption, resumption, and closure), which to some degree correspond to the interactions in the interaction framework (Table 9).

Activities are either ongoing, past, or planned.

1. **CONSTRUCTION:** Construction is the practice of defining the context of an ongoing or planned activity. During this practice, users gradually build up and modify the containing items, thus refining the scope of the activity. Because there are no clear borders that fully define activities up front, the overhead of constructing, updating, and moving items in between activities should be minimized, supporting ad hoc and post hoc activity creation. Improving general activity awareness can reduce occasions where activities become intertwined.

2. **INTERRUPTION:** External and self-interruptions are events which often carry context and elicit the user to reconstruct or resume another past, ongoing, or planned activity. By providing a central interruption mechanism, the context of incoming








							
ACTIVITY	1.	3.	4.			2.	
WORKSPACE	Construction	Resumption	Closure			Interruption	Time
ITEM							
MATERIAL							

Table 9: Practices which influence the activity life cycle, positioned within the interaction framework. Task management is fully covered, whereas workflow management will be elaborated upon as part of co-Laevo.

interruptions can easily be incorporated into new or existing activities, supporting *construction*. Additionally, coupling interruption handling to activity construction also improves activity awareness, as users are encouraged to actively pair incoming interruptions to their corresponding activities.

3. **RESUMPTION:** Resumption is the practice of restoring the full context (all ‘items’) of a previously *constructed* past, ongoing, or planned activity. When resuming an activity, ideally its context reflects the user’s mental model before the activity was interrupted. This facilitates quicker resumption, essential for efficient multitasking. Although activities can become irrelevant, they might need to be revisited shortly, or even restored at a later point in time.
4. **CLOSURE:** When an ongoing activity loses its temporal relevance during *multitasking*, its context is removed from the set of parallel activities. Rather than deconstructing the activity, the entire constructed context (all ‘items’) should be preserved in such a way to allow for easy resumption at a later time. Archiving should be instant and reversible.

6.2 PERSONAL INFORMATION MANAGEMENT

*Laevo supports
dedicated
workspaces
organized in time.*

Laevo is a *temporal* activity-centric desktop interface for Windows 7 and 8. It is designed for personal use on a single computer in support of office work. Going back to the seminal work of Rooms [55], Laevo allows users to structure work (applications and files) within *dedicated workspaces*. Similar to other activity-centric computing systems [16, 17, 58, 135], each workspace can be used to reflect a separate activity constructed by the user. Where previous activity-centric computing systems focus primarily on providing functionality *within* such workspaces, Laevo focuses on all practices which influence activities over time. The entire *activity life cycle* is supported by introducing workspaces for to-do items, planned activities, and providing support for interruptions and easy transitions between different activity states. When working within a workspace, the user interface of Laevo is kept to a minimum; no new interface elements are introduced except for a system tray icon through which the user is notified of incoming *interruptions*. Only when opening the activity overview, are users presented with a touch-enabled full-screen user interface from where activities and interruptions can be accessed, opened, and organized (Figure 23).

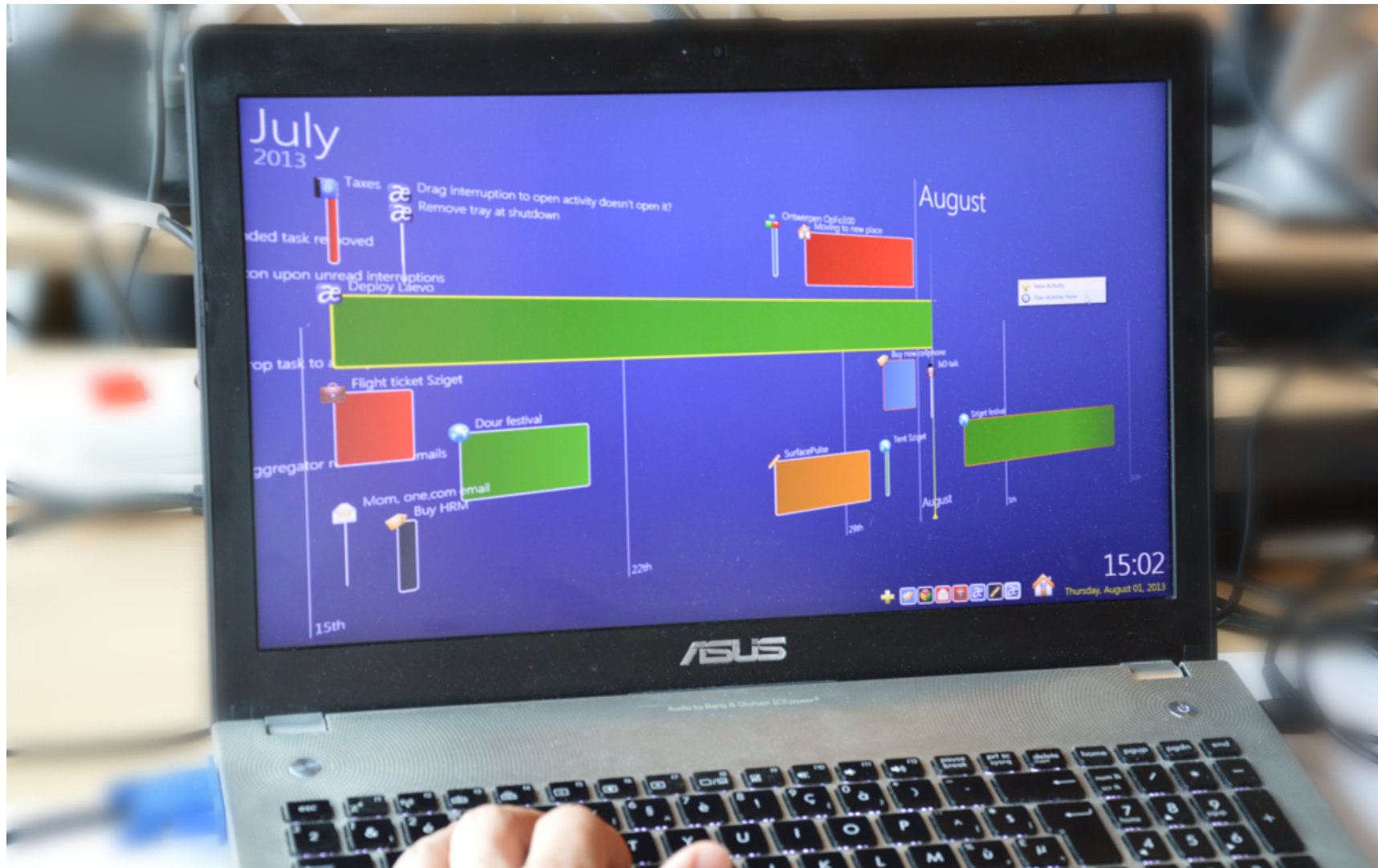


Figure 23: Laevo is an *activity-centric* desktop interface that supports: (1) persisting, restoring and managing window configurations and files within *dedicated workspaces* which are managed on a time line; and (2) handling *interruptions* and to-do items through a centralized notification system, in place or by redirecting them to new or existing workspaces. [64].

Dedicated workspaces are implemented as virtual desktops; interruptions as tray icon notifications.

When working on a particular activity, only the relevant activity context is visible. This helps reducing information overload and thus allows the user to maintain focus. This is achieved by setting up a virtual desktop per activity, which still supports all the existing tools the user is accustomed to (Figure 24). The context of the activity automatically arises from the user working within the currently open desktop environment; windows which are opened within the workspace are assigned to the activity it represents (activity *construction*), but the underlying resources (e. g., open files within applications) remain unmodified. When the user switches to a different workspace, the currently open windows are hidden and all windows of the new activity context are restored (activity *resumption*). Users can always see at a glance whether something requires their attention; Laevo introduces a system tray icon which lights up yellow when new *interruptions* arrive (Figure 24 B). The visualization is intentionally kept to a bare minimum in order not to disrupt the flow of the user. Users decide for themselves when they want to address interruptions, which they can do by opening the activity overview.

An activity context library supports file management.

In order to organize *files* within the context of activities, each activity has access to an *activity context library* (Figure 24 A). This is a standard Windows library, similar to “Pictures” or “My Documents”, directly accessible from the side menu in Windows Explorer. Libraries allow the user to aggregate files from a specified set of paths. The content of the library is contextualized by Laevo so it only shows paths assigned from within the currently active activity. One folder is created by default and maintained for each activity, reflecting its date and name, e.g., “3-11-2014 UIST 2014 paper”. This supports the common file management practice of structuring files within folders representing tasks [56]. At any time, users can freely add new folder locations from their existing folder hierarchies to the library, ensuring interoperability with previously established archiving practices.

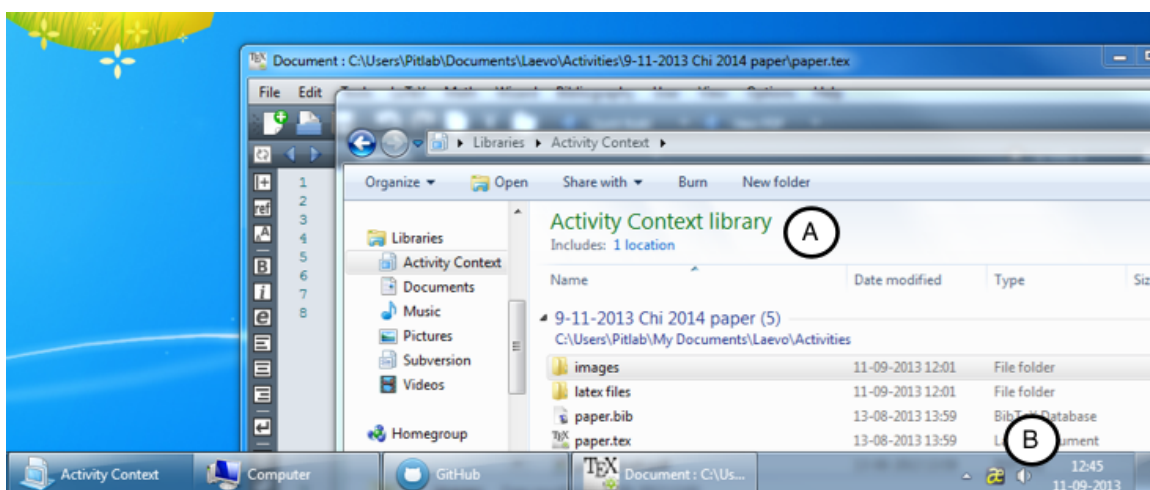


Figure 24: The (A) activity context library and (B) tray icon within a dedicated activity workspace.

There are no clear-cut borders in between activities: while working on one activity the user might have started work on another. Therefore the system provides a flexible mechanism by which windows can be moved in between different workspaces. Similarly to how files are cut and pasted, shortcut keys can perform these operations on application windows once they have focus. Windows that are cut disappear immediately. Multiple cut operations can be performed in a row, and a subsequent paste operation reassigns all windows to the currently active workspace. Other shortcut keys allow quick access to the activity context library, creating new empty activities, closing the current activity, and switching between the last two accessed activity workspaces.

Transitioning between activities is optimized to support the fluid nature of activities.

6.3 THE ACTIVITY TIME LINE

Laevo provides an overview of activities in *time* and space (Figure 25), highlighting the conceptual model on which the design is based—the *activity life cycle* (Section 6.1). At a glance users can see when activities were open or are *planned*. Spatial organization, colors, and icons can be used to identify activities or indicate relationships between them. This can be a powerful cue for recall by invoking episodic memory [83], thus facilitating the *resumption* of old activities. A full-screen representation hides the low-level details of the work users were working on and is intended to help users reason on a higher level of abstraction. The overview contrasts the ongoing activity with the full surrounding work context, including parallel ongoing activities (supporting *multitasking*). The time line is the ‘heart’ of the application, and can be accessed at any point in time using the re-appropriated keyboard button, CAPS LOCK³, which when pressed provides immediate access to all other activities. It also acts as a modifier key to access all other shortcut keys. Alternatively the user can double-click the tray icon to access the overview. The granularity of activities can vary drastically; some lasting weeks, others mere minutes. Therefore, the time line supports continuous touch-enabled panning and zooming in order to show any arbitrary time interval⁴. Depending on the zoom level, the semantically most meaningful labels are shown (e.g. months or days). A one-dimensional representation of time, as opposed to a more traditional calendar layout, emphasizes the fragmented and parallel execution of activities over time [49]. Lastly, applying a perspective transformation has two advantages. It saves screen estate without hampering readability too much [84] and it strengthens the presentation of time following the common conceptual metaphor of expressing time as a physical path in space. For example, “approaching the end of the year” or “leaving days behind”.

Activities can be managed from a full-screen time line overview.

³ This idea is borrowed from the now defunct Humanized Enso launcher application.

⁴ The time line flexibly scales to *any* resolution, thus supporting different form factors.

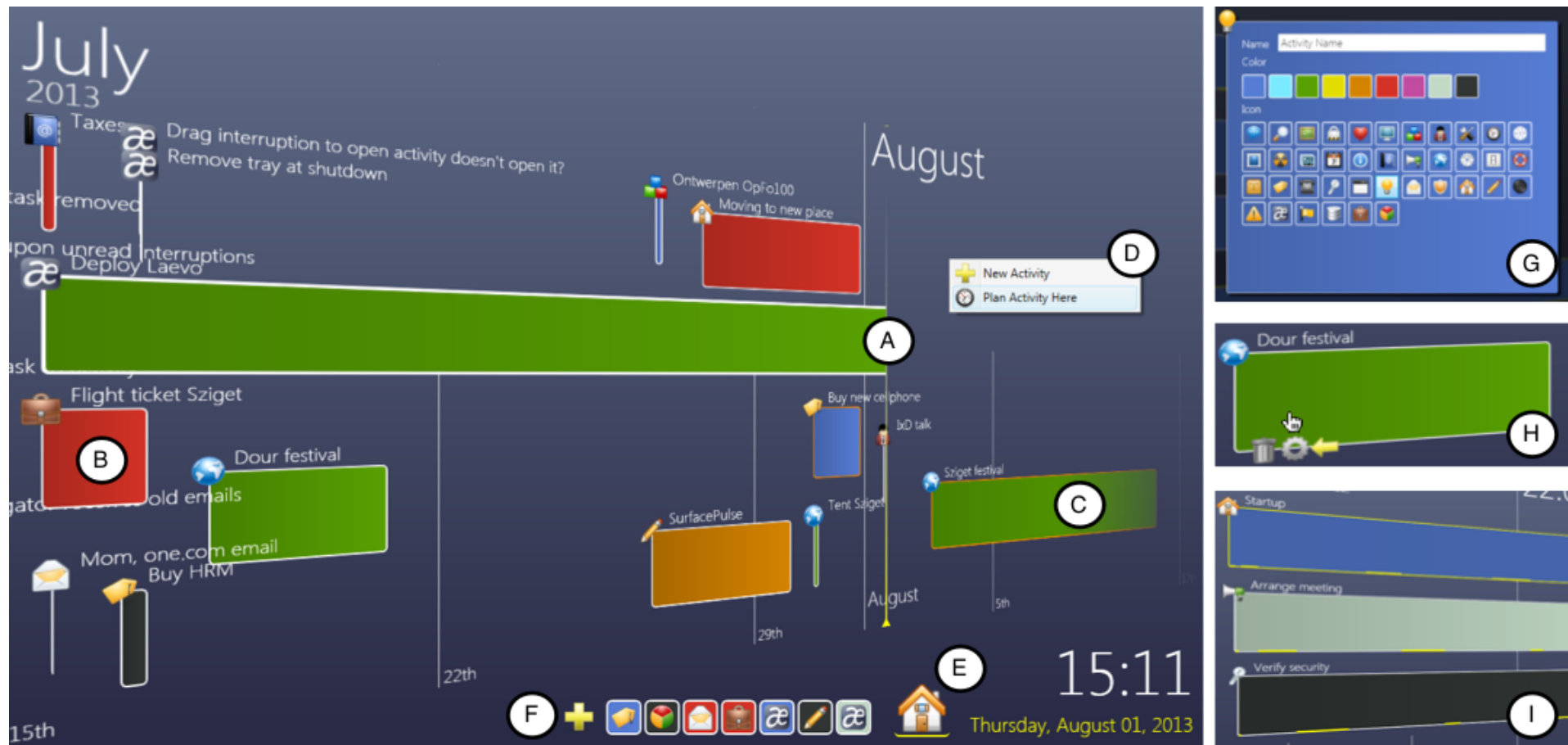


Figure 25: The activity overview through which activities are accessed and managed in Laevo. It displays (A) open ongoing activities, (B) archived activities, (C) planned activities, (D) a context menu through which new activities can be created, (E) the home activity, and (F) to-do list. The three detailed images show the (G) popup menu to edit activities, (H) action buttons when hovering over an activity, and (I) attention span lines.

Figure 25 highlights the user interface elements accessible from the overview screen, as well as the different *states* activities can be in. An activity can either be: *open* to represent ongoing work (Figure 25 A); *closed* when finished in the past and now *archived* in time (Figure 25 B); or *planned* when a time slot has been assigned for it in the future (Figure 25 C). When starting Laevo, all windows previously not associated with an activity are assigned to a permanent *home activity* (Figure 25 E). It can be used as a workspace for smaller, less important tasks which aren't immediately recognized as an activity, or aren't activity-specific. Activities are represented by rectangular areas which stretch along the time line horizontally, indicating when the associated work context was open (or is planned to be). Only one workspace can be *active* at any given time (highlighted with a yellow border on the time line), but several can be open simultaneously, allowing for *multitasking*. An activity is made active simply by clicking on it, after which the overview is hidden and its dedicated workspace shows up. Yellow lines along the bottom of a workspace indicate when they were active (Figure 25 I), but this visualization can be disabled when deemed too distracting.

It is up to the user to change the state of activities as they unfold. This encourages users to reflect on their ongoing work, increasing activity awareness. The state of closed and planned activities can be changed to open through a hover menu, which is always positioned near the mouse (Figure 25 H). Doing so brings the activity into the current multitasking session, visually represented by stretching the rectangle up to a line which indicates the current time. Activating an activity does not open it, so at any time it can be inspected without changing the state of the activity. The same hover menu is used to edit and remove activities, however, removing activities is discouraged. Instead, the system encourages *archival* over time by simply *closing* activities. To reinforce this, an ongoing activity can only be removed after it has been closed and when it does not contain any open application windows. Closed and planned activities that do have windows open in them are displayed with an orange border.

The user can edit the name, icon, and color of each activity in a pop-up window which can be accessed from the hover menu (Figure 25 G). As the activity is edited, changes are applied immediately to reflect the final representation on the time line, i. e., the background color and icon in the top left corner changes. For easy access, activity names can also be edited directly from the time line; the labels act like ordinary input boxes. Activities can be dragged up and down, allowing the user to organize them vertically. This combined with the *temporal* dimension in which they are displayed gives plenty of suitable visual cues to identify activities, following the redundancy gain principle.

Activities are either open, closed, or planned. Only one activity can be active at a time.

Users need to change the state of activities in order to facilitate multitasking.

Activities have a name, icon, color, and can be positioned vertically.

6.4 TO-DO LIST AND INTERRUPTIONS

To-do items are activities which are not yet positioned in time.

Although future activities can be planned at specific time intervals on the time line, there is also a need for less structured task management. Laevo allows users to create *to-do items* which behave the same as activities, except that they are always visible on the overview screen irrespective of the currently visible time interval (Figure 25 F). Only their icons and colors are shown in order to save up space. However, hovering over to-do items shows their name, as well as a menu to edit or remove them (Figure 26 A). Simple drag operations allow users to rearrange to-do items to reflect their priority. If a user decides to start working on, or specify a suitable time interval for a to-do item, it can be dragged to the time line. Dropping it in front of the current time opens it, while dropping it behind plans it at that particular position (Figure 26 B). Alternatively to-do items can be dragged to existing activities, in which case their contents are merged (the associated windows and activity context library paths).

Interruptions are introduced as to-do items.

Laevo features a centralized notification system for *interruptions*, well integrated with the rest of the system. It does so by equating interruptions with to-do items. They are added to the to-do list but can be distinguished by a flashing yellow border (Figure 26 A). When within a workspace, the tray icon lights up as long as there are unattended interruptions. Opening interruptions for the first time opens up their context with the appropriate application. For example, when an email is received it shows up in the to-do list with the subject as name; clicking on it opens up the email message. Using the normal to-do list functionality, this allows users to either handle interruptions in place (from within the workspace), merge them with existing activities, open, remove, or plan them.



Figure 26: (A) Interruptions, e.g., emails, arrive in the to-do list and are highlighted. (B) To-do items can be dragged to the time line to start work on them or plan them.

This design is based on early psychological considerations for interruptions formulated by Miyata and Norman [93]:

Think of the user buried in the task, unwilling to be interrupted, but every so often finishing a cycle and “coming up for air,” quickly breaking from the task and taking a quick look around. The reminder should only be noticeable during that “breathing spell”.

When switching between activities, part of multitasking, users are confronted with any pending interruptions in the to-do list, but within a workspace (while working) users are only notified of interruptions through the tray icon.

Interruptions are only shown when switching between activities.

6.5 THE COOPERATIVE ACTIVITY LIFE CYCLE

co-Laevo extends on the design of *Laevo* to incorporate support for cooperating teams [65]. ‘Cooperation’, as conceptualized within the field of *CSCW*, refers to *interdependence in work*: “multiple individuals working together in a conscious way in the same production process or in different but connected production processes” [118]. In addition to the actual cooperative work, *articulation work* needs to occur to enable effective cooperation: the coordinating, scheduling, meshing, and integrating of interdependent activities. To this end, *coordination mechanisms* are often put in place [120]. *co-Laevo* provides computational support for such coordination mechanisms in a desktop work environment for knowledge workers.

co-Laevo extends on Laevo to support coordination mechanisms.

The design of *co-Laevo* was inspired by considering the *activity life cycle* (Section 6.1) within a cooperative work environment, in which activities are shared. This gives rise to several design implications for *cooperative activity life cycle* management, which will be presented from three different perspectives, summarized in Figure 27: (1) the users, (2) their data, and (3) the different views on that data.

Design implications for cooperative activity life cycle management are introduced.

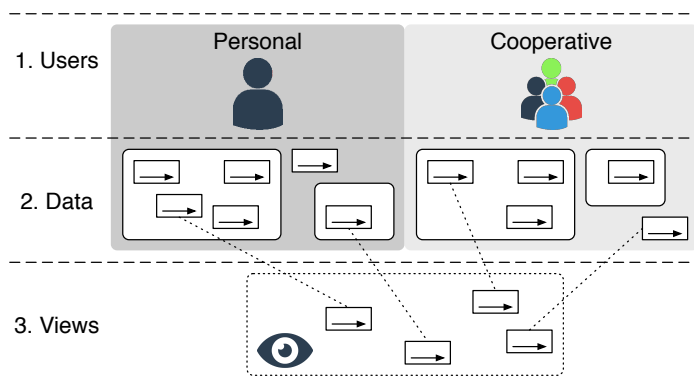


Figure 27: Implications for design in cooperative activity life cycle management. Activities can be both personal and cooperative, requiring a seamless transition between a personal and cooperative view [65].



Activity signifiers tie a personal workspace to a shared workspace.

Collective management of the state of activities supports team awareness.

USERS (*shared activities*) In addition to personal activities, users need to have access to the shared activities of users they cooperate with. This implies that for each activity there is a local, as well as possibly shared context associated to it. Common to both, however, is the *activity signifier*: a necessary description used to refer to and discuss the activity [57]. To this end, each participant needs to be able to have a personal workspace associated to a shared activity signifier. Although the workspace is local, it can be used to access shared resources and initiate collaboration with participants.

Activity *construction* is “the practice of defining the context of an ongoing or planned activity. During this practice, users gradually build up and modify the content, thus refining the scope of the activity” [64]. Different from PIM, activity construction is no longer restricted to one individual user. As posited by Schmidt and Bannon [118], within cooperating teams there is a need to “support the ongoing dynamic articulation of distributed activities and the cooperative management of the mechanisms of interaction themselves”. Activity coordination can be supported by collectively managing the state of activities in a shared work environment (e.g., an activity that changes from a planned state to open, indicating work has started on it). However, in order to keep users that depend on an activity up to date, there is a need to inform everyone within the shared work environment of modifications (these might be possible *interruptions* during ongoing work), including life cycle state changes (e.g., an activity which is completed by another user). In addition to facilitating coordination, this supports team *awareness* by regularly confronting users with the activity descriptions and states of other users when switching between their own activities as part of everyday knowledge work (*multitasking*).



Activities should be groupable.

DATA (*organizing activities*) Considering that the number of activities that need to be managed in a cooperative environment is the same as during personal information management, but multiplied (approximately) by the number of users one cooperates with, there is a scalability issue that needs to be addressed. Even more so than in PIM, there is a need to be able to group related activities together and relate them to other groups of activities. Merely representing activities along a temporal dimension no longer suffices (even for small activity sets) since the user needs to be able to share groups of activities with other users. These should be separable from one’s own private activities. Additionally, groups of activities might be part of an overarching higher-level goal. For example, programming activities can be shared within a software development team, but furthermore are part of higher-level project management of the overseeing company. It should thus be possible to fluently define, relate, and share, collections of activities.

A permanent internet connection, and thus a continuous up-to-date collection of activities, cannot be guaranteed. Therefore, activity state conflicts can arise when users modify the state of a shared activity while offline. Since the personal planning and associated local workspace of users might depend on the state or existence of the activity, the concerned users need to be notified and presented with a resolution mechanism once activities are synchronized. For example, two separate users might have opened (and thus indicate having started work on) an activity. A choice needs to be made whether one of the users abandons ongoing work, or whether work up to that point should be aggregated. Alternatively, one of the user continues work on a copy of the activity.

Resolution mechanisms are needed to synchronize divergent activities.

VIEWS (*personal and cooperative workspaces*) Laevo was designed with PIM in mind, providing support to easily *suspend and resume* the activities of just one user. Although cooperative activity life cycle management additionally requires access to the activities of users one cooperates with, not all shared activities should be made part of the personal workspace. This would rapidly lead to information overload. Users thus need to be able to claim ownership (which can be shared with other users) over the activities they are interested in. To further support coordination, it should also be possible to suggest ownership to others. A similar view to that of Laevo (a personalized overview) can show all activities one has *claimed ownership* over, distinct from a view which provides access to the full set of activities one *has access* to. A seamless transition between the two supports frequent switching between them as part of creating and selecting new activities to work on.



Activity ownership should be distinct from activity access.

6.6 SHARED ACTIVITY HIERARCHIES

The original design of Laevo has two distinct work environments, one *activity management* environment (the activity time line), and several *dedicated workspaces* (one per activity). On the activity time line, Laevo employs the inherent state changes of activities over time as a way of organizing and accessing them. co-Laevo extends on this design by introducing *activity hierarchies* and *shared activity time lines*. During the design process a strong emphasis was placed on finding a satisfactory compromise between the original requirements derived from PIM literature (the personal overview of activities as presented in Laevo) and incorporating access to even larger numbers of activities part of a cooperative work environment. The newly introduced features do not conflict with the earlier design of Laevo. In fact, co-Laevo can be seen as a conceptually coherent superset of Laevo which merely introduces an additional layer of abstraction to scale up the original design to a cooperative context.

co-Laevo introduces activity hierarchies and shared activity time lines.



Activity hierarchies support the hierarchical nature of goals.

ACTIVITY HIERARCHIES As part of an evaluation of Laevo (which will be presented in [Chapter 8](#)), insufficient support for high-level activities that need to be revisited only sporadically was observed, like long-term projects representing particular clients. Users expressed problems in managing them: stopping and reopening projects did not feel adequate as the project was not discontinued, but rather, did not require any attention at the time. This indicates a need for the user to be able to define more granular activities within the context of higher-level activities. For example, a meeting activity could be considered part of an overarching project activity. This observation is reflected in activity theory [73], in which activities constitute *actions* directed at specific *goals* which in turn are part of attaining an underlying *motive*. Goals can be decomposed into sub-goals, sub-sub-goals, and so forth. For example, a PhD student might be motivated to graduate (the activity), for which he needs to write a thesis (the action), containing several different lower-level actions targeted at sub-goals, like reading up on related research. Although activity theory distinguishes between activities and actions, there is no such need to make a distinction in the user interface. It is the user who mentally assigns an intent to an activity upon its creation. However, the hierarchical nature of goals should be reflected in the user interface in order to support richer management of both high-level and low-level goals.

Each activity has a dedicated workspace and an independent time line.

To this end, co-Laevo introduces a separate time line for each activity within the system. Each activity represents thus not only a point of access to a dedicated workspace, but also to a time line from where its sub-activities can be managed. For intelligibility reasons (as not to confuse the user) a choice was made not to make the state of activities depend on that of parent activities, and vice versa. Activities can thus contain sub-activities of any state, e. g., open activities can contain planned activities, and closed activities can contain to-do items. Not much is to be gained from automating possible dependencies between the two, in contrast to the added complexity this would introduce. The ‘Home’ activity of Laevo is used as the root for the activity tree which represents a top level time line. An example activity hierarchy for a PhD student is shown in [Figure 28](#).

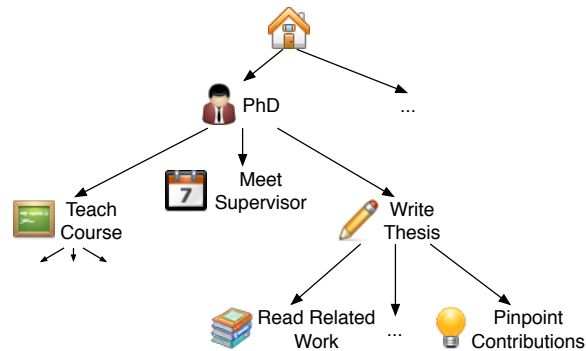


Figure 28: An example activity tree for a PhD student.

The original time line of Laevo provides a complete overview of all the user's personal activities. In contrast, the *personal view* in co-Laevo shows only a subset of all accessible activities. In a newly introduced *hierarchy view* (Figure 29) all activities in the hierarchy can be accessed and users can decide which activities to show on their personal time line by claiming ownership over them (Figure 29, 4). The personal time line thus provides an aggregated overview of a set of selected activities from different levels in the hierarchy view. Browsing hierarchies is similar to how folders are navigated in an ordinary file system, with breadcrumbs indicating the current position within the activity hierarchy (Figure 29, 1). The hierarchy time line shows the containing activities of the currently selected activity (functionally identical to the old personal view from Laevo since there were no hierarchies). Pressing a button allows seamlessly transitioning between both views (Figure 29, 3): the background color changes, the time line is repopulated with the requested activities, but the current visible time interval remains the same. In both the personal and hierarchy view, an activity log keeps track of changes made to the activities of the currently visible time line (Figure 29, 7).

A personal view shows a subset of activities selected from a hierarchy view.

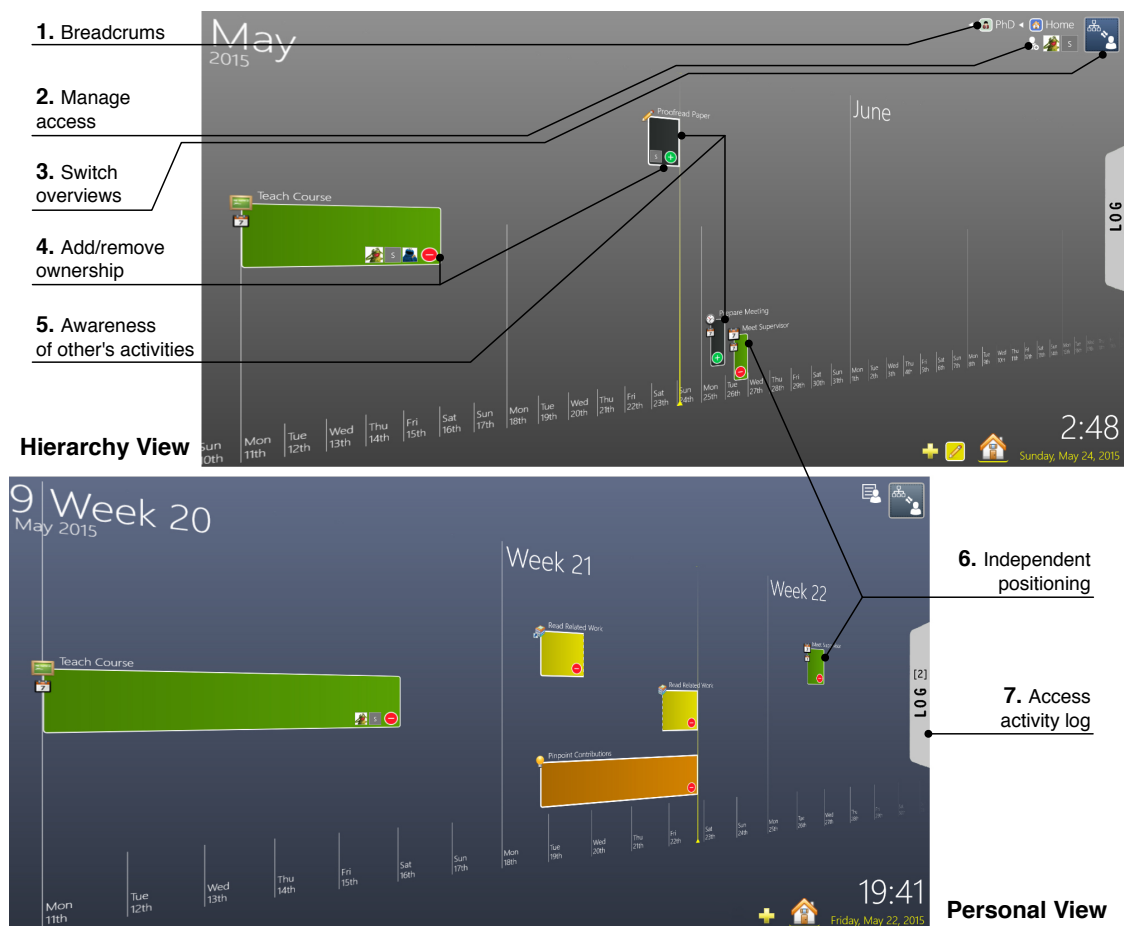


Figure 29: An overview of the newly introduced cooperative features of co-Laevo in both the new hierarchy view, and preexisting personal view.

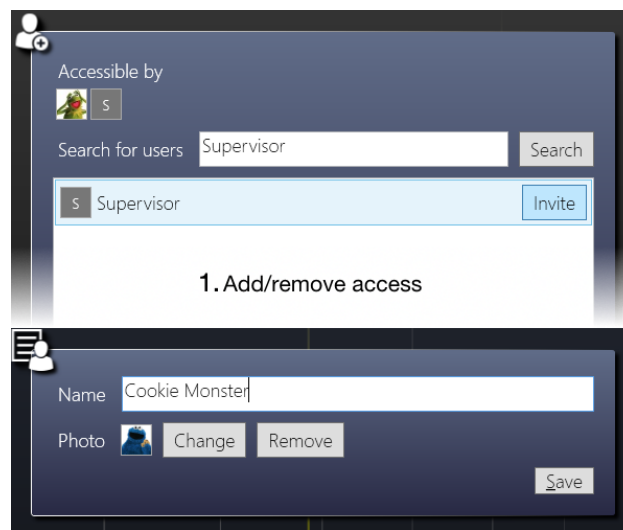
The personal view is conceptually identical to Laevo.

Compared to Laevo, the personal time line remains largely unchanged, except for newly added cooperative features including a user profile (Figure 30, 2), a list of owners per activity, and the ability to remove ownership over an activity (same representation as in the hierarchy view). Removing ownership removes the activity from the personal time line but keeps it in the hierarchy view. Activity ownership is not shown for the active user on the personal time line as this would be a redundant visualization; it is, however, shown in the hierarchy view. The vertical position of activities on the personal view can be modified independently from the vertical position on the hierarchy view (Figure 29, 6). This is necessary to maintain a personalized organization of activities in the personal view as opposed to the hierarchy view which can be organized collaboratively.



Articulation work is an important part of cooperation.

SHARED TIME LINE Prior activity-centric computing systems provide the means to share activities and their work context with other users and devices [58, 135]. However, support for coordination is limited to sharing *individual* activities. No explicit support is provided to *articulate* (divide, allocate, coordinate, schedule, mesh, interrelate, etc.) activities within cooperative work arrangements [118]. Although some strict interpretations of *situated action* favor highlighting the ad hoc (in situ) nature of knowledge work over elaborate support for planning, the two are not mutually exclusive [121]. For example, activity theory argues that plans are achieved, but undergo continual modifications in the course of action. There is thus a need to support *situated planning*: the plan should be made a malleable part of the activity [12].



2. Profile

Figure 30: The window used to (1) add and remove access for users, and (2) modify the user profile. Initials are shown when no profile picture is set.

co-Laevo supports situated planning by allowing users to access shared activity time lines (Figure 29, hierarchy view). The notion of *activity access* is distinct from *activity ownership*. Activity access implies having access to an activity and all of its sub-activities (and their sub-sub-activities, etc.), but does not imply activity ownership. Activity ownership means users claim ownership over an activity, at which point it will be displayed on their personal time line. The user will be notified of any changes made to the activity through the activity log accessible from their personal view (Figure 29, 7). Users who have access to an activity can see who has claimed ownership over it⁵ (Figure 29, 4). In short, activity access can thus be used to share and coordinate plans with participating users, and activity ownership can be used to set up a personal work environment. Ownership can also be suggested to other users, but remains in a pending state until approved, which can be done from the personal activity log. In case an owned activity is removed by someone else, the activity is automatically moved to the home time line, thus removing it from the shared work context. Similar to other owned activity state changes, the user is notified of removal through the personal activity log.

From the hierarchy view, access can be given to other users to the currently shown activity time line (Figure 29, 2 and Figure 30, 1). This will trigger an interruption which is added to the to-do list of the recipient, representing the activity they were just invited to. This is distinct from activity notifications (introduced in co-Laevo as the activity log), which refer to existing activities and do not carry any new context. Although the initial activity representation (icon, color, and name) corresponds to that of the invited activity, invited users are free to change the representation locally. In essence, only the containing activity time line is shared, allowing users to freely mount the shared activity anywhere within their own personal activity hierarchy using the original activity manipulations available in Laevo. They can thus also choose to represent it as an open, closed or planned activity.

Ownership over activities can be claimed or suggested from shared activity time lines.

Users can be given access to a time line, of which they are notified through the to-do list.

⁵ User who have ownership might be hidden when insufficient space is available on the time line. This depends on the current zoom level.

Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer. Much of the complexity that he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform. These differ from interface to interface, and from time to time, not because of necessity but only because they were designed by different people, rather than by God.

— *No Silver Bullet*, Frederick P. Brooks, Jr. [31]

In order to indicate how much the design of user interfaces relies on, and is influenced by underlying architecture [42], I presented some key points of interest within the history of interactive computing systems at the start of this dissertation (Chapter 1). Although new computing systems have a lot to gain from building on top of prior technologies, their design is also largely restricted by it. This has led to a historical *bottom-up* approach to system design, resulting in contemporary computing systems which in many ways still resemble some of the earliest systems introduced¹. In contrast, activity-centric computing is a *top-down* approach to system design which envisions a radically new computing paradigm. Regardless of the aversion most researchers in this line of research feel towards the current technological stack, they still need to build on top of current technologies in order to be able to evaluate their newly introduced systems within a real-world work environment. Therefore, a major focus of prior work has been to integrate existing applications into activity-centric computing systems so that their content can be made part of computational activities. This is a labor-intensive, error-prone process, which in addition is often invalidated by future releases of the targeted applications. As discussed in Chapter 4, the majority of work in activity-centric computing has therefore operated on the workspace layer within the interaction framework; multiple applications need to be aggregated into one overarching ‘activity context’ workspace. This has left little time to focus on some of the higher-level implications of activity-centric computing, i. e., activity management.

Implementing an activity-centric computing system is labor-intensive.


¹ Reading up on early HCI literature is a truly humbling experience in that it reveals a vision for future technologies which in many ways resembles and even transcends present-day technologies. With a focus on technology and a short time to market only a handful of features are adopted and the underlying vision often gets lost along the way (e. g., collaboration in NLS [43], virtual desktops in Rooms [55], and the Xerox Star which was much more document-oriented than applications are today [29, 68]).



The *DW* toolkit supports creating research prototypes for activity management.

DW TOOLKIT To this end, I introduce a toolkit² which centralizes the redundant work of integrating separate applications into overarching *dedicated workspaces* (*DW*). Although these can be used to represent activities, this is not a prerequisite—activities are not part of the supported abstraction. As an intermediate approach, the *DW* toolkit “sit[s] atop of a layer of more fundamental infrastructure” [42], namely the Windows operating system and existing applications (Table 10). The main focus is to support the *construction* and *positioning* (suspend and resume) of arbitrary workspaces, not tied to any one specific application. Less focus is placed on *search*, although this is a logical point for extension. This frees up time for researchers to focus on *activity management*, essential to advancing the research agenda for activity-centric computing (Table 10). It must be noted that this toolkit is merely intended to be part of a *transitional phase*, supporting the construction of prototypes which “serve as ‘proxies’ for how real applications built on the final infrastructure might work” [42]:

Thus, after identifying user-facing needs, general capabilities can be pushed down into infrastructure in a top-down manner.



ACTIVITY	Activity management (Laevo [64], co-Laevo [65], and ActivitySpace [60])					
WORKSPACE	<i>DW</i> toolkit		NooSphere [59]			
ITEM						
MATERIAL	Operating system & applications					

Table 10: The dedicated workspaces (*DW*) toolkit is an intermediate approach to supporting the construction of arbitrary workspaces.

Prior middleware solely focuses on activity distribution.

Such a toolkit which can act as ‘middleware’ for the construction of dedicated workspaces did not yet exist. Although the underlying infrastructure of activity-centric computing systems is at times described, no independent reusable implementation is made available to the public. For example, the *ABC* framework for medical work in hospitals does shortly describe key components of the implemented architecture [16], including the integration of different applications with one central ‘state manager’, but does not provide the source code to enable this functionality as an isolated package. In contrast, NooSphere supports “the prototyping of distributed interaction systems” [59]. As an activity-centric toolkit it provides support for the distribution of activities (enabling activity *sharing*), resolving many of the complexities surrounding the implementation of networked applications. However, it does not support the integration of existing applications within the context of dedicated workspaces.

² Source code is available on github: <https://github.com/Whatthecode/ABC>

7.1 ARCHITECTURE

The DW toolkit (Figure 31) is built on top of the .NET framework for the Windows operating system. It exposes an application programming interface (API) implemented in the C# programming language, designed for researchers or developers interested in creating an interactive computing system supporting dedicated workspaces. Within a *workspace manager*, default implementations for common workspaces (e. g., a virtual desktop manager) are available. New types of workspaces can be introduced by extending on the provided abstract classes. In addition, a *plug-in management system* integrates with the prepackaged workspace managers in order to allow developers to integrate with or specify custom behavior for native applications. These plug-ins are loaded at run time using the Managed Extensibility Framework (MEF) or through simple configuration files. Plug-ins can communicate with extensions for native applications using, e. g., named pipes. Under the covers the workspace managers use Platform Invoke (P/Invoke) to communicate with the native Win32 API³. Two less developed components, part of ongoing research, enable centralized *interruption management* (e. g., incoming emails) and the invocation of *services* in an application-agnostic way (e. g., opening a webpage). Since the framework acts as middleware, it necessarily tailors to just one specific operating system. However, it might inspire the design of similar architectures on other operating systems.

The toolkit is designed for Windows and can be extended on to support more applications.

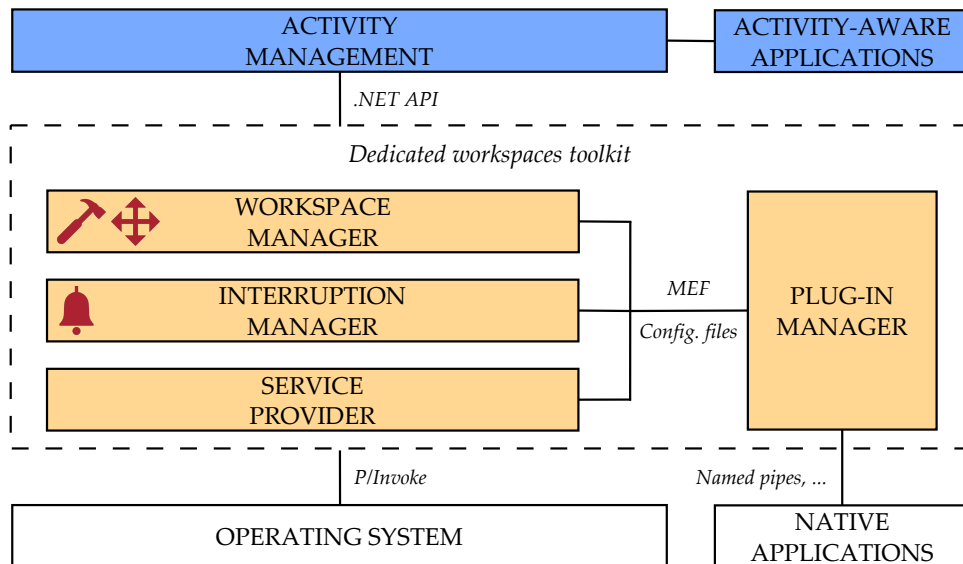


Figure 31: The dedicated workspaces toolkit is comprised of four components: a (1) workspace manager, (2) interruption manager, (3) service provider, and (4) plugin manager. It sits on top of the operating system and supports the creation of interactive computing systems which support *integrated* activity management.

³ A managed wrapper for P/Invoke is used, part of a separate library: <https://github.com/Whathecode/Framework-Class-Library-Extension>

7.2 WORKSPACE MANAGER

A central workspace manager manages and supports switching between dedicated workspaces.

The central component of the DW toolkit is the WorkspaceManager. It aggregates multiple instances of AbstractWorkspaceManager and enables performing operations on them as a whole. From this one central location, dedicated workspaces can be *constructed*, *switched between*, and *merged*. In line with the composite pattern, the WorkspaceManager itself is an AbstractWorkspaceManager, thus (in theory⁴) supporting workspace hierarchies (Figure 32). Depending on the specific managers passed during the construction of the aggregate manager (Listing 1), different items are incorporated into the context of each individual workspace. For example: a VirtualDesktopManager manages application windows; a LibraryManager manages the paths of a specific Windows Library (as employed in Laevo, see Section 6.2); and a DesktopIconsManager manages desktop icons (as employed in an updated version of co-Activity Manager [58] using the DW toolkit). Concrete implementations of AbstractWorkspaceManager operate on one specific AbstractWorkspace implementation. For example, the VirtualDesktopManager manages VirtualDesktop's (Figure 33). The WorkspaceManager thus needs to handle different concrete implementations of AbstractWorkspaceManager and AbstractWorkspace. Therefore it accesses them through non-generic interfaces⁵, which for the sake of simplicity are not displayed in Figure 32. When switching between workspaces using the WorkspaceManager, all corresponding items from the aggregated workspaces are swapped out.

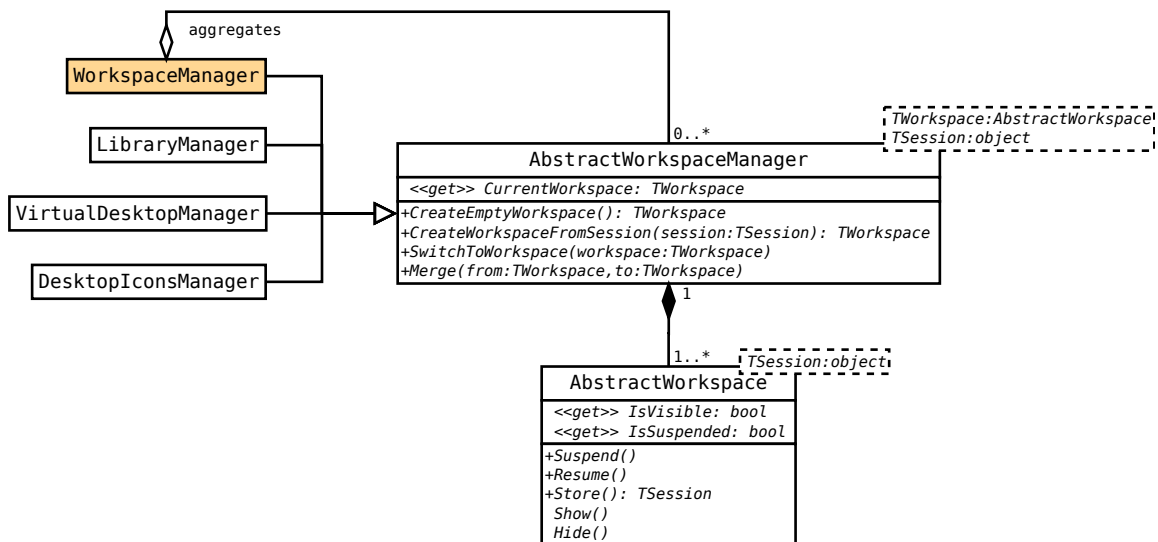


Figure 32: Class diagram for the WorkspaceManager. Extensibility is supported by extending from AbstractWorkspaceManager, allowing to incorporate additional items in a workspace.

⁴ Specific implementations of AbstractWorkspaceManager need to account for this.

⁵ The abstract classes contain a NonGeneric wrapper (following the adapter pattern) initialized from within the constructor: <https://whatthecode.wordpress.com/2015/12/15/non-generic-wrapper-instead-of-base-class-or-interface/>

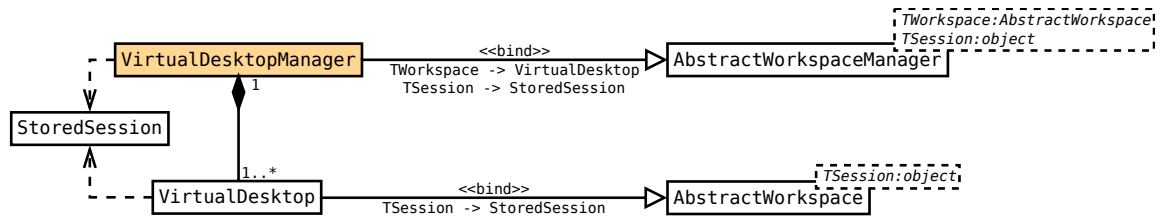


Figure 33: Example of one concrete `AbstractWorkspaceManager`, the `VirtualDesktopManager`.

```

// Create individual workspace managers.
var vdmManager = new VirtualDesktopManager( vdmSettings,
    persistenceProvider );
var libraryManager = new LibraryManager( "Activity Context" );

// Initialize aggregate workspace manager.
WorkspaceManager = new WorkspaceManager( new[] { vdmManager.
    NonGeneric, libraryManager.NonGeneric } );

```

Listing 1: Setting up a `WorkspaceManager`.

Internally, switching between workspaces is achieved by *showing* and *hiding* them. This implies that even though items from a certain workspace are no longer visible, they might still take up computational resources. Since creating too many workspaces might slow down the system, it should be possible to *suspend* a workspace, thereby releasing the resources it contains (e.g., the `VirtualDesktopManager` should close down applications). This is an *interjected abstraction*: a situation “in which low-level infrastructural concepts become part of the conceptual model of the interface” [42]. To make managing a plethora of activities within activity-centric computing a possibility, suspending workspaces should be made part of the user interface⁶. Using the `DW` toolkit, a recent version of Laevo [64] incorporates said functionality (Figure 34). Workspaces can be suspended from *within* the workspace. The `DW` toolkit closes as many resources as possible (i.e., those for which plug-ins are available), and subsequently asks the user to clean up any remaining ones. Alternatively, remaining resources can be merged with the home workspace. When *resuming* the workspace, the resources suspended by the `DW` toolkit are reopened.

Workspaces can be suspended and resumed to free up computational resources.

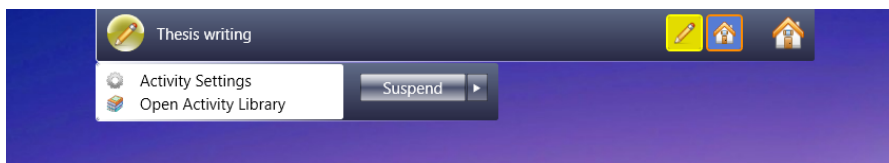


Figure 34: Suspending a workspace in a more recent version of Laevo.

⁶ I do so reluctantly, since this goes counter the vision of activity-centric computing in which activities can simply be suspended by switching to another activity.

Only one workspace can be visible at a time.

Another inevitable interjected abstraction, due to not being able to close the traditional desktop environment without shutting down the system, is that one workspace needs to be visible at any given point in time (they cannot all be hidden)⁷. In addition, only one workspace can be visible (or more specifically, be manipulated) simultaneously. This is part of the conceptual model of a ‘workspace’, which is deliberately enforced by the [API](#). It is by performing work ‘within’ a workspace that the containing context is built up. Therefore, allowing users to work on multiple workspaces simultaneously would inevitably lead to dissolving the borders in between them. At least, without clear demarcations (which are not part of the desktop metaphor), the system, and possibly even the user, would not know which workspace new or repositioned items belong to. However, workspaces can be merged with one another, and concrete implementations of `AbstractWorkspaceManager` can implement mechanisms to move (or share) individual items in between workspaces. For example, the window cut and paste operations supported by the `VirtualDesktopManager`.

Workspaces can be stored so they can exist beyond the lifetime of the application.

An `AbstractWorkspace` can be stored, which saves the current state of the workspace in an object serializable by the .NET framework (using `DataContractSerializer`). The caller still needs to decide when and where to store this ‘session’. Sessions can be passed to a workspace manager to reinitialize the state and containing items of the corresponding workspaces. This is distinct from ‘suspend’, e. g., a workspace can be stored without suspending it. For example, `Laevo` can be closed down, at which point all workspaces are stored and previously hidden items reappear (using a `Close` method). Upon restarting, workspaces are recreated from the stored sessions and only a `StartupDesktop`⁸ remains, showing all resources that could not be allocated to prior workspaces.

7.3 PLUG-IN MANAGER

Plug-ins are required to integrate with preexisting applications.

Any middleware which intends to support a broad set of preexisting practices would out of necessity have to rely on a plug-in management system in order to integrate with the near endless amount of independent applications which are available on the market. A plug-in manager can support the *loading* of independently developed integrations, and furthermore facilitate their *distribution*. Without a plug-in manager in place, each release of a new or updated application necessitates evaluating and possibly redeploying the system under construction in order to guarantee that it functions properly.

⁷ `Laevo` somewhat works around this by showing a full screen user interface (the time line) which can not be closed when a workspace is suspended. Underneath, however, the workspace is still visible.

⁸ For simplicity a `Close` method and `StartupWorkspace` are not displayed in [Figure 32](#).

For example, the `VirtualDesktopManager` relies on two points of extension: *window specifications* and *persistence providers*. These are loaded from components passed during the construction of `VirtualDesktopManager` (e.g., Listing 1): `IWindowSpecifications` and `AbstractPersistenceProvider` (Figure 35).

Workspace managers can be extended by loading plug-ins.

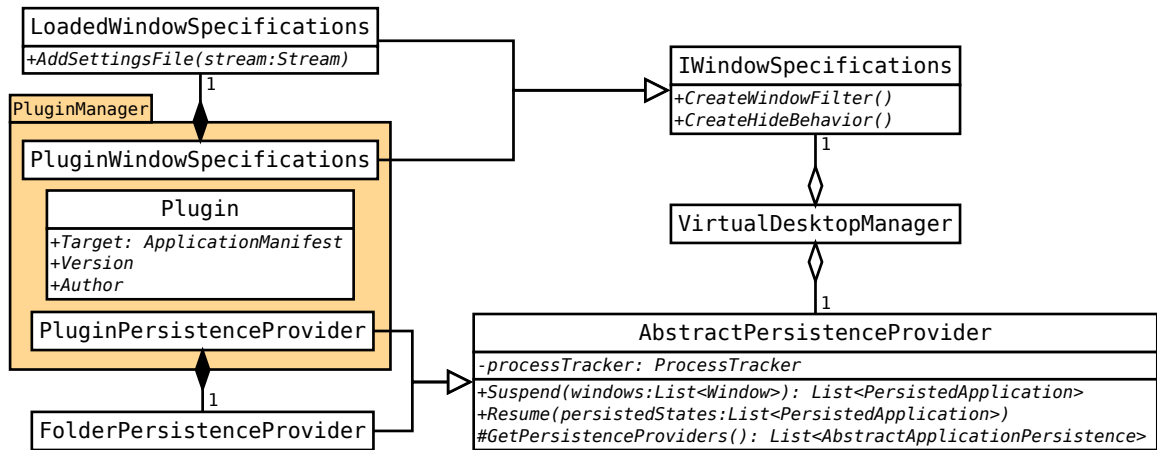


Figure 35: `VirtualDesktopManager` plug-ins which are managed by `PluginManager`.

Window specifications indicate which windows need to be managed (or not) and can specify custom behaviors for when application windows need to be hidden, i.e., during window cut and paste operations and when switching between workspaces⁹. Window specifications can be loaded through settings files using `LoadedWindowSpecifications` (example configuration file in Listing 2).

Window specifications are loaded through settings files.

```

<Process Name="chrome" CompanyName="Google Inc.">
  <IgnoreWindows>
    <Window ClassName="Base_PowerMessageWindow" />
    <Window ClassName="SWFlash_PlaceholderX" />
    <Window ClassName="CSpNotify Notify Window" />
  </IgnoreWindows>
  <HideBehavior>
    <Default Hide="SelectedWindow" />
    <!-- Visible status bar also needs to be hidden. -->
    <Include ConsiderWindows="AllWindows">
      <Window ClassName="Chrome_WidgetWin_0"
        Visible="True" Title="" />
      <Window ClassName="Chrome_WidgetWin_1"
        Visible="True" Title="" />
    </Include>
  </HideBehavior>
</Process>
  
```

Listing 2: Settings for Chrome, which has a status bar as separate window.

⁹ Each process in Microsoft Windows can host several application windows, some of which are never visible, and some of which are only made visible when the process decides to. Since processes handle application windows differently, it is not possible to rely on a single universal implementation to handle all application windows.

Persistence providers are loaded through highly specific plug-ins.

Persistence providers are used when *suspending* a workspace to save the state of specific applications and subsequently to shut them down, and to reinitialize state once they are resumed. There are no default ways of suspending applications, which is why configuration files do not work in the case of `AbstractPersistenceProvider`. Instead, plug-ins are loaded through `FolderPersistenceProvider`, which uses [MEF](#) to retrieve assemblies conforming to the interface of `AbstractApplicationPersistence` from a folder on the hard drive. These implementations vary wildly, and require a certain degree of creativity to access native applications. For example, restoring Windows Explorer windows (file system explorer) relies on accessing the Windows [API](#); restoring Notepad (a simple text editor) relies on monitoring passed command line parameters to see which file was open ([Listing 3](#)); and restoring open browser tabs from Chrome requires implementing a custom Chrome extension which uses inter-process communication ([IPC](#)) to talk to the plug-in loaded by the [DW](#) toolkit.

```
[Export( typeof( AbstractApplicationPersistence ) )]
public class NotepadPersistence : AbstractApplicationPersistence
{
    public NotepadPersistence()
        : base( "notepad", "Microsoft Corporation" ) { }

    public override object Suspend( SuspendInformation toSuspend )
    {
        // Is data to know which file was open passed?
        if ( toSuspend.CommandLine == null ) { return null; }

        ProcessHelper.Setup( @"C:\Windows\System32\taskkill.exe", "/"
            pid " + toSuspend.Process.Id, "", true ).Run();

        // Extract file name from commandLine.
        Match split = Regex.Match( toSuspend.CommandLine, "\"(.*)\" (.*)" );
        return split.Groups[ 2 ].Value;
    }

    public override void Resume( string applicationPath, object
        persistedData )
    {
        if ( persistedData == null ) { return; }
        var filePath = (string)persistedData;
        ProcessHelper.Setup( applicationPath, filePath ).Run();
    }

    public override Type GetPersistedDataType()
    {
        return typeof( string );
    }
}
```

Listing 3: The persistence provider for Notepad.

Similarly, other `AbstractWorkspaceManager`'s might have to rely on plug-ins to integrate with preexisting applications. As per the quote at the start of this chapter, it is thus necessary to master "arbitrary complexity" [31], conforming to the many different interfaces which applications that need to be integrated with expose. This is a difficult and time-consuming task, and therefore we would do well by making this a collaborative research effort. Ongoing work is to set up a visual `PluginManager` (Figure 35), part of the `DW` toolkit, through which plug-ins can easily be installed, shared, and distributed. This would support the long-term deployment of activity-centric computing systems which can be updated externally to satisfy unanticipated needs from end users. A `Plugin` targets a specific application and lists the versions it supports, thereby allowing to compare applications installed on the user's computer to available plug-ins. Following the strategy pattern, plug-ins loaded by the `PluginManager` are managed by separate concrete implementations (e.g., `PluginPersistenceProvider`), but can nonetheless rely on the earlier described mechanisms internally (Figure 35).

A plug-in manager is needed for a longitudinal deployment.

Part III

EMPIRICAL STUDIES

I regarded [the virtual desktop manager (VDM) of Ubuntu] as several screens between which you can switch, [in Laevo] I saw it more as a distribution of my activities.

— *Laevo participant in first study*

After identifying early usability and stability problems in a one day in situ pilot study (including 12 participants), Laevo (Chapter 6) was deployed during a two-week field study¹. The goal of this second study was to assess the short- and mid-term feasibility of using the system and explore the perceived impact of Laevo on the work practices of the user. Six participants (age ranging from 28 to 59, one female) were recruited to participate in the experiment. Participants came from a broad range of backgrounds, including consulting, engineering, and software development, to represent different types of knowledge work. Participants were required to have experience with either Windows 7 or 8. Similar to the pilot study, the experiment was conducted in situ, meaning that the system was deployed on their personal computer, used in their own work environment. Two out of 12 users from the pilot study participated in this second long-term study. The other 4 participants used Laevo for the first time.

Laevo was evaluated during a two-week field study including six participants.



6

METHOD Users were sent a link to a blog post² containing an installer and a complete manual of Laevo (Section B.1). Participants were requested to use the system over a period of 14 days during their day-to-day work. During this period, they were asked to keep a diary, in which they had to add one entry each day. The entry was based on a number of predefined questions on how and if they used the system that day, with a particular focus on any special events that had occurred (Section B.2). Additionally, usage data from the system was automatically collected throughout the experiment. At the end of the experiment participants were requested to anonymize any sensitive information before submitting their data and diary. From this data, a local representation of their timeline could be reconstructed for analysis. Finally, after the experiment was completed, participants were invited for a semi-structured interview in which their experiences with the system were discussed. Initial questions were individually outlined beforehand based on their diaries.



Participants filled out a daily diary and a semi-structured interview followed.

¹ Only Laevo was evaluated during a field study, co-Laevo was not.

² <https://whatthecode.wordpress.com/2013/08/04/start-of-laevo-user-studies/>



Participants were more focused and more aware about their activities.

RESULTS All participants experienced benefits by structuring their work within the context of activities, confirming prior findings of in situ field studies which have evaluated activity-centric computing systems. Once activities are set up, people like having only those things for the task at hand visible as it helps them in keeping *focus*. Participants experienced losing less time when switching between parallel tasks since the right folders and files were still open. Without Laevo, extreme *filers* (P₁, P₂, P₄) ordinarily closed windows in order to coordinate many parallel tasks.

“I’m getting used to not having a cluttered desktop. I made an activity for something that only took about one hour, but this allowed me to focus on the task at hand.” – P₁

When asked to elaborate on where this focus came from it became clear that Laevo made participants more *aware* about their activities. The activity overview played an important role in this. Participants argued that each time they switched between activities they were exposed ‘at a glance’ to all ongoing and planned work, which P₅ stated as being an advantage over a to-do list on paper which can easily be ignored. This allowed them to make more conscious decisions about which activities to prioritize. As mentioned by one user:

“By explicitly defining your activity, you are [...] forced to reflect on what is your current active ‘task’, [which] seems to make you less distracted. When you do switch to something else, it becomes a very conscious choice.” – P₆

Analyzing the participants’ time lines indicated that for all users except one (P₃) the majority of activities were concrete instantiations aimed towards a certain goal, as opposed to ‘types’ of activities like ‘programming’. Noteworthy, two users (P₂, P₃) who were using Outlook instead of Gmail, and thus were not receiving email interruptions, created long-running ‘check email’ activities.

“Using other [VDMs] I structured my work according to [type of work]. What distinguishes Laevo from other [VDMs] is it is inviting to organize [work] as concrete activities.” – P₆

Activity scopes varied and changed over the course of the experiment.

The activity model provided by Laevo was appropriated by users to support different activity *scopes*, ranging from only having short hour-based tasks (P₂, P₆), to long-term projects that lasted several weeks (P₃, P₄), to a combination of both (P₁, P₅) (Figure 36). One user (P₅) created a number of short tasks at the start of each day, meticulously planning which work needed to be done. As the day progressed he used the activity time line to check whether he was still on schedule, reprioritizing activities where needed. Another user (P₄), however, argued that activities only made sense to him for long-term projects. As part of his work, he created activities for each of the clients he was

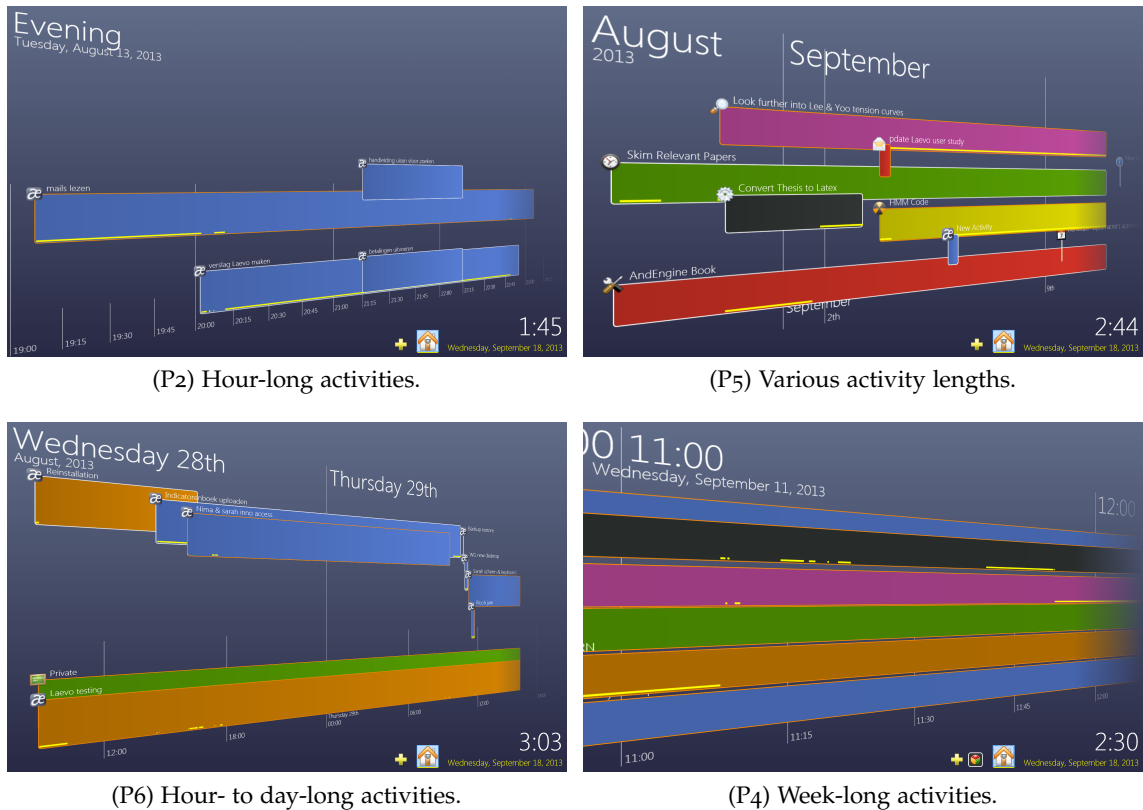


Figure 36: Different activity scopes, depicted by the time lines of participants P2, P4, P5, and P6.

working with. For all smaller tasks, he used the home activity or simply handled them inside the currently open activity. In general, all participants saw the home activity as a catch-all environment, often used for quick and dirty work or private activities, thus keeping actual work separated from ‘daily clutter’. All participants that eventually created short-term activities mentioned a *‘learning curve’* they had to go through when using Laevo. After a while, P1 and P2 adopted a *‘nothing ventured, nothing gained’* attitude towards creating activities. The up-front configuration work initially seems like an overhead, but over time users start to see several advantages when doing so: *less clutter, increased focus, productivity and efficiency*.

“[...] the more separate activities I start in Laevo, the less I’m tempted to quickly do another activity within an existing one, the more fluently and efficient I can work. You have to ‘learn to use’ Laevo – as is the case with everything.” – P2

Participants incorporated the system to varying degrees into their existing work practices, depending on how much overlap there was with the other tools they used. Before using Laevo, P2 used to write to-do items on a piece of paper. She first experimented with planning activities in the future in order to remind her of them, but afterwards started using the built-in to-do list instead. P1 opened up

Appropriation of Laevo depended on the overlap with existing practice.

entire projects within to-do items as placeholders for side projects he wanted to start working on when he had some spare time. Another participant (P5) preferred the task list from Google since he could easily access it from his mobile phone. However, at the start of each day he manually transferred the to-do items with the highest priority to the time line from Laevo, planning his day. He preferred the overview in Laevo over the Google task list. Lastly, P6 did not use the to-do list nor the incoming email interruptions since he kept an elaborate to-do list on paper, which he divided in different zones by priority. Similarly, participants used different approaches to construct activities. P5 had set up emails to arrive in his to-do list. A request to update his daily report of Laevo lead to opening it up as a new activity, while a reminder email by Google was dragged to the corresponding time on the time line. Another participant (P4) who had not set up email interruptions manually cut and pasted email windows to the corresponding activities, effectively using them as reminders within his long-running projects. P2 manually introduced to-do items, which lead to the creation of activities the next day. Analyzing the time lines of all participants (e. g., Figure 36), it was clear that users applied a wide variety of color schemes and physical layouts.

The study highlighted new features for co-Laevo.

Laevo provides limited support for the revisitation of past activities. Previously closed activities that are reopened are shown on the time line as a continuous visualization, which most participants found to be an incorrect representation. The original design rationale behind this feature was to create a connection between the two points in time the activity was used. The yellow lines indicate when the activity was active. The users, however, argued that it should not be represented as a continuum but rather as *multiple instances* of that particular activity. This feedback was incorporated in the design of co-Laevo (Figure 37). Stopping and reopening activities splits their representation.



Figure 37: Multiple instances of one activity in time in co-Laevo.

The concept was proven effective, but more applications need to be integrated.

Three out of six participants (P2, P5, P6) continued using the system after the study. Two other users mentioned they would continue using Laevo in case stability issues would be resolved (P1 and P4) and in case integration with Outlook would be provided (P4).

TASK SWITCHING IN SEQUENTIAL MULTITASKING

The heart of what information overload really is may very well lie between tasks rather than within. — Mulder et al. [95]

A major part of this dissertation focuses on how new technologies can help out users during intensive knowledge work. Activity-centric computing posits that providing computational support for human activities allows users to more easily switch between independent ongoing work, thus facilitating *multitasking*. Although this basic premise is based on theories of cognition and has largely been confirmed by the evaluation of several different activity-centric computing systems, the resulting findings are predominantly qualitative. They do not provide deeper insights into the *degree to which* supporting multitasking through the use of *dedicated workspaces* improves knowledge work, nor how multiple workspaces compare to a ‘single workspace’ work environment. Therefore, in this section I introduce two studies¹ which evaluate computational support for *sequential multitasking*—the interleaving of several tasks which are executed one at a time. These studies investigate two of the fundamental principles of activity-centric computing: *activity-centered* and *activity multiplexing* (Table 11).

Few studies investigate how dedicated workspaces support sequential multitasking.








								
ACTIVITY	1. Activity-centered							
WORKSPACE	(dedicated workspaces)	2. Multiplexing						
ITEM		(multitasking)						
MATERIAL								

Table 11: The ‘activity-centered’ and ‘multiplexing’ principles of activity-centric computing cover support for sequential multitasking through the use of dedicated workspaces.

The first study is “an experimental study incorporating 16 participants in which a traditional Windows 7 environment is compared to one augmented by virtual desktops” [62]. The second study “investigat[es] the window manager of Windows 7, unraveling the processes and strategies used when switching from one task to another” [66]. Both studies share a similar methodology: participants follow the same procedure and work on the same task sets. However, the data analysis differs. Therefore, related work and part of the methodology for both studies will be presented as a whole.

Two studies are presented which share a similar methodology.

¹ These studies will largely be presented as they were in earlier publications [62, 66].

9.1 MULTITASKING CONTINUUM

*There are describes
different types of
multitasking.*

As I will discuss here, several studies in both cognitive sciences and HCI study multitasking, as well as related concepts such as interruptions and task switching. However, some ambiguity exists in the terminology used. Where some theories, models, and studies investigate short-term subconscious processes, others describe activities which can last up to several hours. According to Salvucci et al. [115], “multitasking can be represented along a continuum in terms of time spent on one task before switching to another.” Along this continuum, research related to multitasking can be divided into (at least) four different topics: studies investigating (1) concurrent multitasking, (2) task switching, (3) interruptions, and (4) sequential multitasking. Although unifying theories of the full spectrum—ranging from concurrent to sequential multitasking—have been proposed [115], the individual empirical studies do not report on the same phenomena and are hence hard to interpret as a whole [61]. The studies presented in this chapter focus on investigating sequential multitasking, but to situate it within this wider research, and to prevent misinterpreting study results, I will provide a short overview and report on important differences and similarities.



*Working on tasks
simultaneously.*

CONCURRENT MULTITASKING During concurrent multitasking cognitive resources have to be divided across several competing parallel tasks, such as driving while talking on the phone. Many ‘dual-task’ studies investigate dual-task interference and reduction of performance while performing two simultaneous tasks. For example, driving while on the phone has been shown to result in slower responses to traffic signals [125]. Other studies show that unfulfilled goals (like finishing a paper) interfere with tasks that require executive function [88, 92]. Executive function includes working memory, reasoning, and problem solving, and can only pursue one goal at a time. Since executive function is in high demand during knowledge work, interleaving several long-term tasks (as studied in this chapter) might decrease overall productivity when mental processes remain focused on prior goals. However, by consciously formulating plans for unfulfilled goals, such problems may be avoided [92].



*Switching between
different types of
tasks.*

TASK SWITCHING Task switching studies within cognitive sciences explore switching costs between tasks at a microscopic level [78], like the action of pressing a key. Such cognitive tasks require an appropriate configuration of mental resources, which task switching studies refer to as a ‘task-set’. This is not to be confused with the task sets participants have worked on in the studies reported on in this chapter, which cover higher-level real-world knowledge work. In a typical task switching experiment, effects of switching between two different

task-sets are observed, like classifying a digit as odd or even, and classifying a letter as consonant or vowel. Switching costs include longer responses on the switched-to task immediately following the task switch. It is tempting to equate such a task switch with higher-level knowledge work where complex task goals need to be substituted. However, this is a fundamentally different operation [94].

INTERRUPTIONS Interruption studies measure the effects of short-lived secondary tasks interrupting a primary task, on both the primary and secondary task. A primary task here refers to higher-level goal-oriented work, like solving a Sudoku. Interruptions are short-term tasks which require suspension of the primary task, like answering a short question posed by a colleague. Interruptions can lead to annoyance and anxiety [8], and feelings of stress and frustration [90, 91]. As summarized by Monk et al. [94], some studies show people perform post-interruption tasks more slowly, and that more errors are made compared to pre-interruption performance. Characteristics determining the disruptiveness of interruptions include task similarity to the primary task, interruption complexity, control over interruption onset, availability of primary task retrieval cues, and duration [94]. Not all interruptions are alike or as disruptive, depending on when they occur. Interruptions at task boundaries cause less anxiety and induce less errors [7], but this might depend on interruption relevance to the primary task [50]. Other findings suggest resuming a primary task slowly can reduce the amount of errors made [32]. Lastly, interruptions can also disrupt task management, i. e., cause the resumption of unintended tasks [41]. Based on such insights, interruption management systems attempt to alleviate the disruptiveness of interruptions by predicting interruptibility of the user [133].

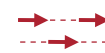
When a secondary task interrupts a primary task two relevant time intervals become important to study: time taken (or allowed) to disengage from the primary task before starting work on the secondary task, and time taken to resume the primary task after completion of the secondary task [6, 30, 131]. Although the studies reported on in this chapter do not include the traditional notion of a secondary task, the *disengagement and resumption stage* are still useful concepts, which can also be recognized within sequential multitasking studies while switching between two tasks.

SEQUENTIAL MULTITASKING In contrast to concurrent multitasking, sequential multitasking denotes the interleaving of several primary tasks which are executed one at a time. Compared to interruption studies there are no secondary tasks, as all tasks are long-lived and of equal importance. This is representative of common everyday knowledge work [11, 37, 49]. Switching from one task to the next involves retrieving all the necessary resources (e. g., multiple appli-



A secondary task interrupting a primary task.

Interruptions give rise to a disengagement and resumption stage.



Executing tasks one at a time.

cation windows) needed to resume the previously suspended task. Studies investigating sequential multitasking are among other things interested in measuring the effects of task interleaving on productivity and accuracy. The control condition typically consists of performing tasks in sequence, as opposed to an experimental condition where the same tasks need to be, or are voluntarily interleaved. Results show an inverted U-relationship between multitasking and productivity; there is thus an optimal amount of task switching which leads to the highest productivity. However, increased levels of multitasking lead to a significant loss in accuracy, indicating a trade-off between productivity and accuracy [2]. This trade-off is further influenced by task difficulty: easy tasks benefit from multitasking due to the increase in stimulation, but task performance for hard tasks can decrease due to an overload in mental workload [4].

There are different reasons for switching tasks.

Interruptions leading to task switches can either be internal (self-initiated) or external. Although most interruption studies focus on external interruptions, internal interruptions are as common in knowledge work, and there are different reasons for users to decide to switch tasks [53, 67]. Based on flow theory, these can be broadly categorized as either originating from negative (e. g., frustration, exhaustion) or positive (e. g., exploration, reorganization) feelings associated with the task [3]. For example, studies show that users have a tendency to continue working on more rewarding tasks (with a continuous rate of return), and have a tendency to switch tasks after the completion of subgoals [41, 107].

The impact of specific window managers has not been studied.

So far, however, no studies have investigated how different support for task switching in a desktop environment influences sequential multitasking. Prior studies generally employ a custom application in which participants can switch between trivial tasks by the press of a button, e. g., solving a Sudoku, unscrambling letters to form words (Scrabble) and finding the “Odd One Out” between a set of shapes [2, 4, 107]. Although these task sets make it straightforward to measure productivity and accuracy, they are not ecologically valid representations of real-world knowledge work within a desktop environment. Using a window manager, users can structure (resize and position), hide, and retrieve application windows, thus supporting sequential multitasking. Therefore, this chapter introduces two studies which investigate the very nature of task switches, as supported by window managers. This differs from prior sequential multitasking studies which have focused on the impact of the amount of task switching on task performance. The first presented study focuses on how task switching as supported by dedicated workspaces influences task resumption and overall knowledge work, compared to a traditional desktop environment. The second study investigates in more detail what comprises a task switch in a traditional window manager, still widely in use to this day.

9.2 TASKS AND TASK SEQUENCE

Similar to other sequential multitasking studies, participants work on a set of different tasks [2, 4]. Since the focus lies on the nature of task switches rather than degree of multitasking, both studies presented in this chapter control for task switches by instructing participants to switch between given tasks at predetermined intervals (mandatory task interleaving), mimicking a heavy multitasking scenario with concurrent deadlines. The concepts of a *disengagement stage* and a *resumption stage* (which make up a task switch) are introduced, illustrated in Figure 38. These concepts build on terminology used in interruption studies [6, 30, 131]. However, in the studies presented here, users do not return to a primary task, but rather switch between two separate tasks as part of an ongoing multitasking session. *Task disengagement time* is defined as the time between the initial *interruption* and the point in time where task resumption commences. In the studies presented here, all interruptions are external alerts given by the experimenter. Real-world interruptions may also be initiated by users themselves (internal interruptions). *Task resumption time* is the time between commencing retrieval of the first needed resource and the first work performed on the next task. This includes retrieving and opening documents required for the next task, finding files, launching applications, cleaning up the desktop, and taking a break.

The presented studies follow mandatory task interleaving to control for task switches.

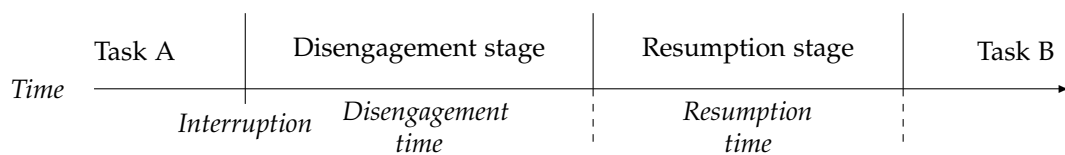


Figure 38: During the disengagement stage, users wrap up a task after having received an interruption. During the resumption stage, users retrieve and prepare the next task to continue working on it [62].

Participants had to switch between four tasks, each requiring several application windows. This supports the main intent of *simulating heavy multitasking where several application windows are open simultaneously*. Tasks were designed according to criteria similar to earlier studies [107]: performance should be measurable, and tasks should be linear in progression (should increase continuously and monotonically with time spent on the task). In contrast to earlier studies [2, 107], each unit of work over time (subtask) required approximately the same amount of *effort*. Participants were instructed to work on subtasks in the order listed in the assignment. All tasks were designed to be long enough so they could not be completed within one hour. This allows for the measuring of productivity as a percentage of completed subtasks, and accuracy as a percentage of correctly performed work. These task requirements eliminate the possibility of using more complex tasks like solving a Sudoku. However, as opposed to prior se-

Participants worked on four separate tasks within an unmodified desktop environment.

quential multitasking studies, none of the applications used throughout the experiment were modified or simulated. Participants worked in a completely unmodified desktop environment (Figure 39). The documents required for the tasks were placed in the default “Documents” folder of Windows. No other documents needed to be accessed during the study. For each task a text file with a short assignment description was provided, referring to the required files and folder location needed for that task.

WRITING (W): Participants type text found within a PDF file into a new text document (copy is disabled). No formatting needs to be applied. At regular intervals the text includes assignments asking participants to either substitute the assignment with text displayed in an image pointed to on the hard drive, or with the translation of a word using Google Translate. Image thumbnails are available.

This implies *four* applications need to be used: a file explorer, a browser, PDF reader, and a text processor.

SEARCHING (S): Participants perform calculations based on searching the Internet. For example, “Height of the Eiffel Tower in meters + year when it was completed =”. Both the intermediate and final results need to be written in a text file.

This implies *three* applications need to be used: a browser, a text processor, and a calculator.

COMPARING (C): Participants highlight differences between an original and modified text. Modifications include synonyms, left out words, or additional words, but the modified text is still grammatically correct.

This implies *two* applications need to be used: two text processors.

ORGANIZING (O): To mimic folder navigation, a folder hierarchy contains images organized by type of object (e. g., bridges, islands ...). A task folder contains the same images but disorganized. Participants need to identify what is displayed in the images from the task folder, find the folder of the corresponding type, within it find the image, and subsequently copy its filename into a text document.

This implies *two* applications need to be used: a file explorer and a text processor. However, experienced users generally use several file explorer windows. In addition, the image viewer can be used when image thumbnails are unclear.

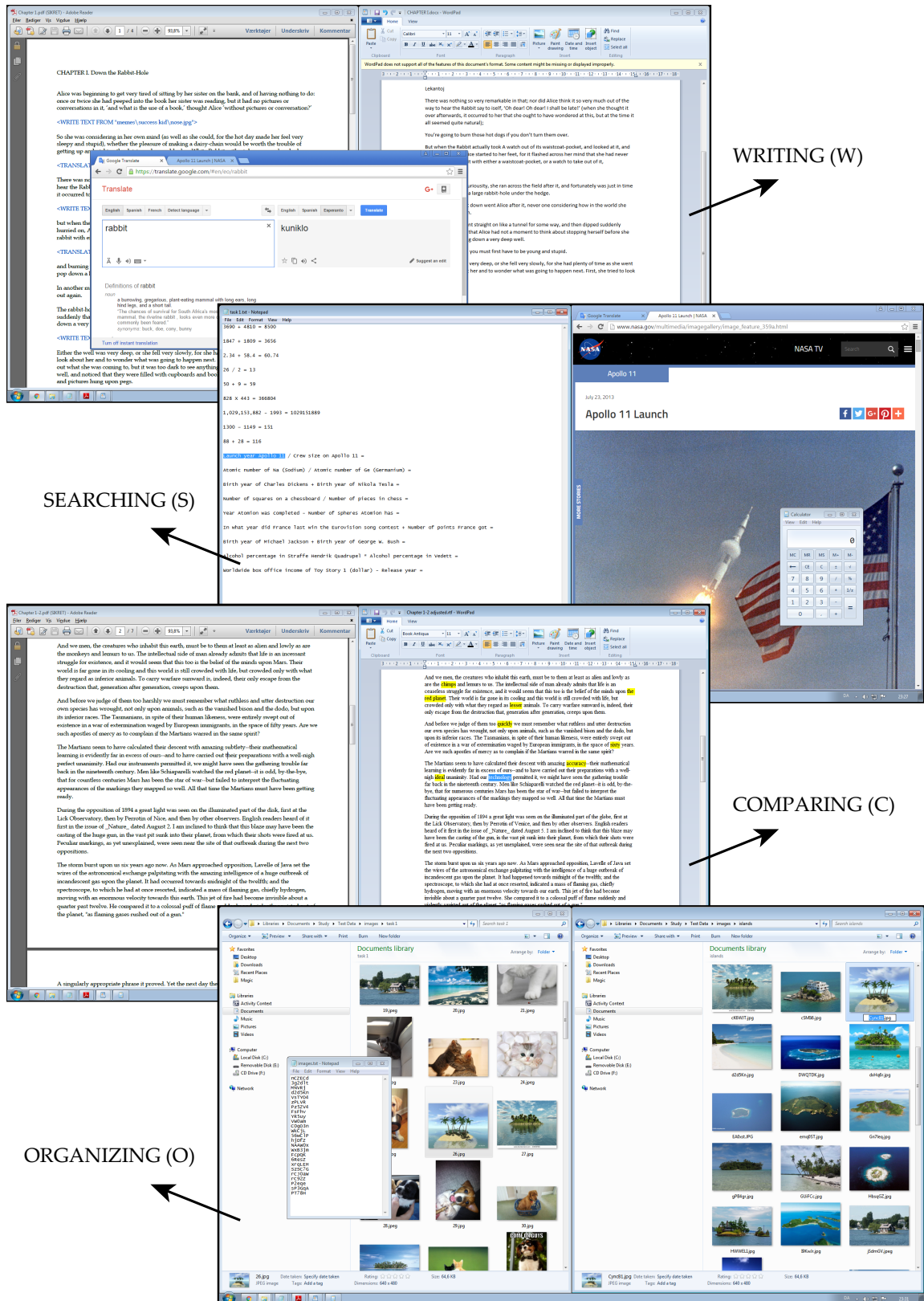


Figure 39: For each of the four tasks used during the studies, a screenshot is depicted showing one possible window configuration while working on them [66].

Over the course of 50 minutes, participants switch tasks 12 times.

During a 10 minute *training session*, participants familiarized themselves with a training task set, similar to the evaluation tasks, and were asked to work on them until they understood what each task entailed. During the *main evaluation*, a researcher notified the user 12 times when to switch between tasks over the course of 50 minutes. These notifications were done at predetermined intervals of 2, 4.5, and 6 minutes, totaling in 12.5 minutes of work per task (Figure 40). This is representative of real-world knowledge work, which involves frequent switching between different tasks [49]. The researcher announced task switches by stating the required task number and a short description: e. g., “Now please switch to task A, which is the writing task.” Participants were instructed that they were allowed to wrap up ongoing work by finishing the current subtask, e. g., a copy/paste operation, or finish writing a sentence, but were not allowed to commence work on a new subtask. Any exceptionally late responses were addressed by restating that the participant had to switch to the next task. Finishing subtasks was allowed to control for disruptive effects due to differences within the *disengagement stage*, since interruptions at task boundaries are known to be less disruptive [7]. The studies presented here are only interested in effects due to differences within the *resumption stage* (Figure 38) [30]. After 50 minutes of working on the tasks, participants completed the NASA-TLX test to assess overall cognitive load during the experiment. A modified version of the NASA-TLX test was used, Raw TLX [54], which eliminates the weighting process of the separate subscales.

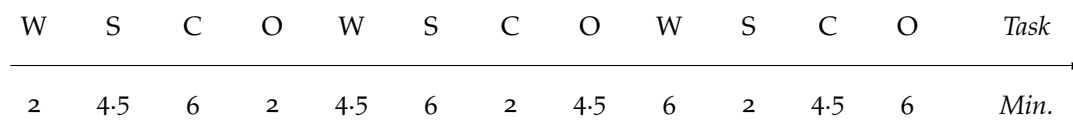


Figure 40: Task sequence of tasks W, S, C, and O, which is followed in both studies [62, 66].

9.3 STUDY 1: COMPARATIVE STUDY

Dedicated workspaces are compared to a traditional desktop environment.

Although the widespread use of virtual desktops clearly shows their importance, no prior studies have assessed their impact on knowledge work compared to a traditional desktop environment. The goal of this study is to *quantify the difference in time taken to switch between tasks under both environments, as well as measure possible effects on the performed tasks and experienced task load.*

H1: *task resumption*

Adopting virtual desktops as dedicated workspaces allows instantaneous switching between parallel ongoing tasks, which in a traditional environment requires several operations. Therefore, it is hypothesized that *it takes longer to resume a previously suspended task under a traditional Windows 7 environment than when using dedicated workspaces.*

The necessary work to switch to the next task could be considered the equivalent of a secondary task within interruption studies. Unlike interruption studies, however, participants do not return to the same primary task but to a previously suspended goal. Since interruptions can lead to annoyance, anxiety, and stress [8, 90, 91], it is hypothesized that *dedicated workspaces will reduce annoyance and anxiety compared to a traditional Windows 7 environment*.

Errors during task management have been shown to affect task performance [41]. Furthermore, in line with interruption duration and complexity influencing disruptiveness [94], the nature of a task switch might determine its disruptiveness. Therefore, it is hypothesized that *simplifying task switching by means of dedicated workspaces will improve task performance*.

PARTICIPANTS Sixteen users (age 27–58, 12 male, 4 female) were recruited to participate in this study. Their backgrounds included program manager, ICT manager, pharmacist, clerk, student in social work, and software developer, representing a broad spectrum of knowledge workers. Only six participants worked within IT, ensuring not only expert computer users were recruited. All but one of the participants had extensive (> 1 year) experience with the Windows operating system, of which 12 specifically with Windows 7. The single inexperienced participant used OS X, and was not as familiar with the Windows 7 window manager, but did work within IT. Only four participants had used virtual desktops before. To incentivize participants in performing the given tasks, they were informed that the person ranking first in the study would win a cinema ticket. It was made clear that overall task score was calculated based on task progress, as well as accuracy. No other compensation was given for participating in the study.

EXPERIMENTAL DESIGN The experiment was run as a within-subjects design with one independent variable—*User Interface: Dedicated Workspaces Vs. Windows*. Under each of the two conditions participants worked on the four separate tasks (Section 9.2) between which had to be switched at predetermined intervals. Two separate but similar task sets were created, one for each condition. Average *Resumption Time* was measured while switching between tasks, as well as average *Construction Time* needed to set up a task the first time. Other dependent variables measured were overall *Cognitive Load* throughout the experiment, and *Productivity* and *Accuracy* for each of the four tasks. The order in which the conditions were completed, and the order of the task sets used, were fully counterbalanced across participants (thus four different groups with four participants each, as shown in Table 12). In addition to counterbalancing for learning effects, this also accounts for differences in between the task sets.

H2:
annoyance and anxiety

H3:
task performance



16



Within-subjects

1. *resumption time*
2. *construction time*
3. *cognitive load*
4. *productivity*
5. *accuracy*

	TASK A - B	TASK B - A
Windows - DW	4	4
DW - Windows	4	4

Table 12: Counterbalancing of condition and task set order: Windows or Dedicated Workspaces (DW) first, and task set A or B first.



Dedicated workspaces from Laevo were used.

MATERIALS It was preferred for users to work in their own work environment to minimize the impact on the measures due to unfamiliarity with the workspace. However, due to practical limitations (e.g., work hours), six of the participants performed the study at a different desk, on a notebook with a 15.6 inch screen provided by the experimenter. Although this introduced variability between work environments, in all cases the same setup was used under both *User Interface* conditions. No multiple monitors were used, but screen sizes did vary. The computer ran either an unmodified Windows 7 environment or was augmented using Laevo [64]. Within the context of this study, and in line with many virtual desktop implementations, a full screen overview simply displays different ‘buttons’ each representing an individual dedicated workspace (as shown in Figure 41), which is opened when clicked. Workspace representations (including a name, color, and icon) could be modified by the participants to help in identifying them.



Figure 41: Example of the full screen overview used for task switching when using Laevo (near the end of the experiment), visualizing the different dedicated workspaces, represented as rectangular ‘buttons’ on a time line.

Using the traditional desktop environment while running the experiment, this results in a workspace as shown in Figure 42, representative of heavy multitasking. In contrast, Figure 39 shows what individual tasks look like when using dedicated workspaces, whereas Figure 41 shows the interface used to switch between them. The main difference lies in how many windows are accessible from the Windows task bar.

PROCEDURE Six users had installed and used Laevo for at least a full day prior to the experiment, and thus did not need an introduction to the system. The other 10 users were given a one-hour introduction on how to use Laevo, focusing primarily on hands-on experience with the virtual desktop functionality. Given that this is the only functionality of Laevo which was used as part of the experiment, this provided them with sufficient training to take part in the study. Before starting the experiment, participants received a 10 minute introduction of the setup and tasks, and were asked about their background, experience with Windows 7, and whether they had used virtual desktops before. Tasks were demonstrated using an example task set, after which participants were asked to work on them shortly. In both conditions, participants worked on the tasks for 50 minutes, after which they completed the NASA-TLX test to assess overall cognitive load during the experiment. In between both conditions, 10 minutes were reserved to recuperate from the heavy workload. In total, the duration of the experiment was two hours for participants that had used Laevo before, and three hours for those that still needed an introduction.



1. *introduction laevo and tasks*
2. *first condition*
3. *10 minute break*
4. *second condition*

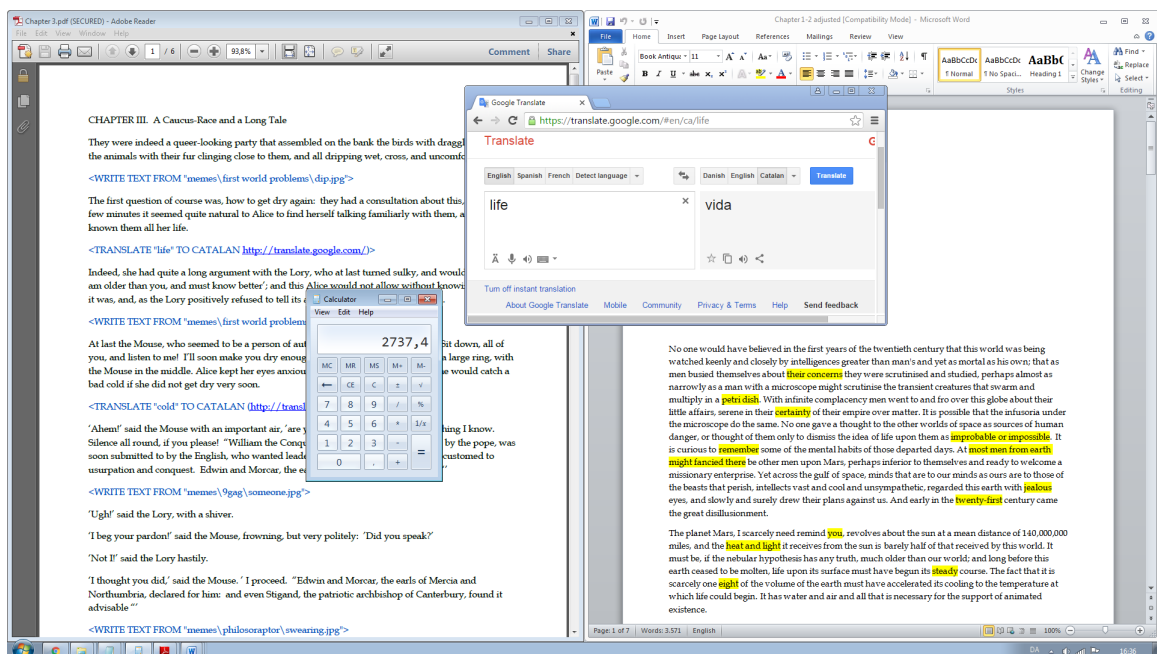


Figure 42: Possible window configuration while working on the tasks under a traditional desktop environment. Windows required for all four tasks are represented on the task bar.

Disengagement and resumption time were measured using a stopwatch.

A stopwatch ran throughout the experiment, used to keep track of task switches. In rare occasions when a problem occurred, either with the dedicated workspaces environment, or participants raising questions about the task set, the timer was paused until the issue was resolved. Both time taken during the disengagement and resumption stage were measured by the observing experimenter for all but three of the participants (due to three experiments which ran with two participants in parallel) by marking the time when the participant stopped working on the previous task, and the time when the participant performed the first operation on the switched-to task. Once all windows required to continue work on the task were retrieved and readily accessible, any mouse or keyboard input on a window of the switched-to task was considered to be the first operation. The resulting time intervals correspond to measured time intervals within interruption studies [6, 94, 131]. The first time participants were asked to work on a task (and are thus not resuming it), construction time was measured in the same way: the time taken to open the required documents and to set up the workspace.



RESUMPTION TIME
significant difference

Three outliers were
identified.

RESULTS Figure 43 shows an overview of average resumption times and standard deviation per task switch throughout the experiment for both conditions. A paired one-tailed t -test was used to analyze overall average *Resumption Time* in both conditions. A significant difference was found ($t(12) = -2.95, p < 0.01$) between *Dedicated Workspaces* ($\mu = 7.52, SD = 2.93$) and *Windows* ($\mu = 26.78, SD = 25.61$). The effect size² using Glass's $\Delta = 0.75$.

Since a high variance under the *Windows* condition was observed (traditional desktop interface), individual resumption times of participants were investigated in more detail. Figure 44 shows an overview of average resumption times and standard deviation per participant for both conditions, ordered by average resumption time under the *Windows* condition. Three outliers can be recognized with resumption times under the *Windows* condition which vary considerably more than the other participants; all three outliers (which includes the OS X user) encountered a task switch which took them over two minutes to complete (250, 140, and 203 s respectively). Removing these outliers and rerunning the t -test, a more significant difference is measured ($t(9) = -4.43, p < 0.001$) between *Dedicated Workspaces* ($\mu = 6.26, SD = 1.92$) and *Windows* ($\mu = 15.79, SD = 7.00$). The effect size using Glass's $\Delta = 1.36$. On average *Resumption Time* is 9.53 s shorter when using *Dedicated Workspaces*.

CONSTRUCTION
TIME
significant difference

A paired one-tailed t -test was used to analyze average *Construction Time* in both conditions, expecting additional time needed to set up *Dedicated Workspaces*. A significant difference was found ($t(12) =$

² Rather than the common Cohen's d , Glass's Δ is used since standard deviations differ substantially between conditions [82].

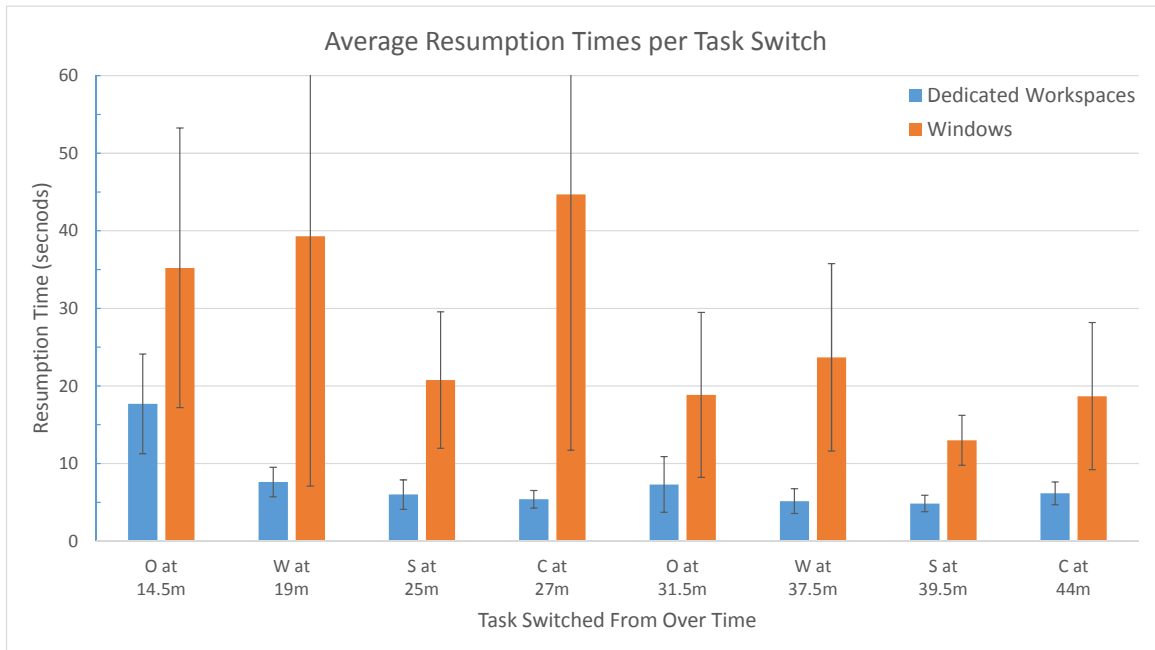


Figure 43: Overview of average resumption time per individual task switch under both conditions.

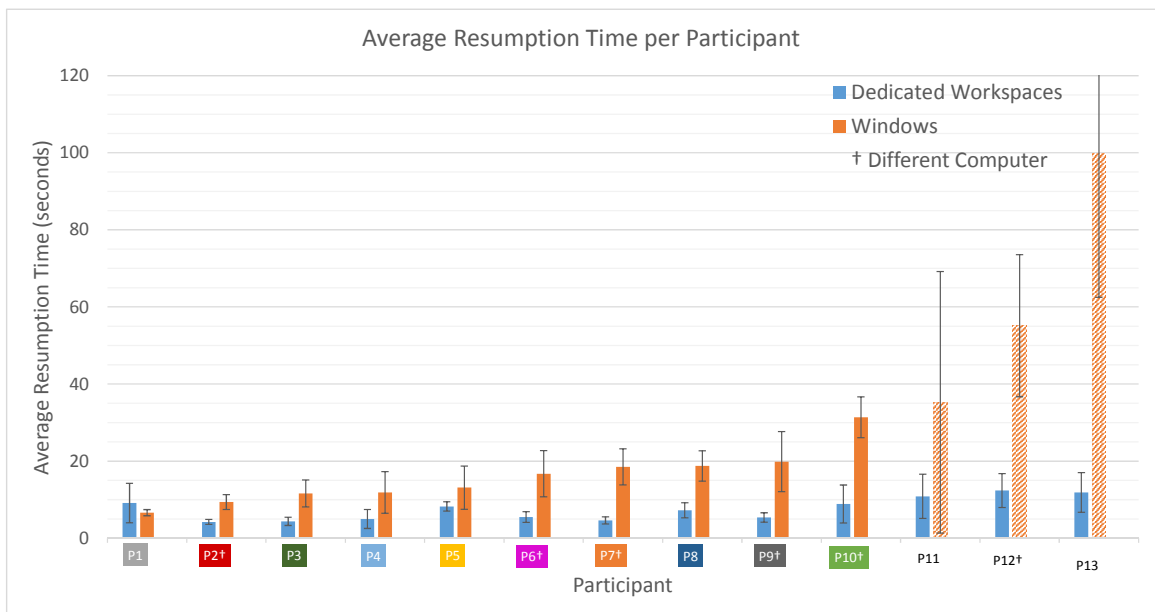


Figure 44: Overview of average resumption times per participant under both conditions. P11, P12, and P13 were identified as outliers due to proportionally high variance in the Windows condition compared to the other participants. Participants marked with ‘†’ did not perform the study in their own work environment, and did not use their own computer.

3.15, $p < 0.005$) between *Dedicated Workspaces* ($\mu = 27.21, SD = 17.05$) and *Windows* ($\mu = 14.05, SD = 5.43$). The effect size using Glass’s $\Delta = 2.43$. On average *Construction Time* is 13.16 s longer when using *Dedicated Workspaces*.

Participants spent less time overall on task switching using dedicated workspaces.

Figure 45 shows an overview of all task switches throughout the experiment for participants one through ten, corresponding to the participants listed in Figure 44. The first three task switches represent construction times where the workspace still needs to be set up. The remaining task switches represent resumption times. The y-axis shows total (cumulative) time spent on task switches over time when using dedicated workspaces, minus time spent on the same task switches under the traditional Windows environment. The graphs thus show whether, and when, using dedicated workspaces resulted in spending more, or less time on task switching. The increase in *Construction Time* when using *Dedicated Workspaces* can be seen during the three initial task switches. However, during subsequent task switches, *Resumption Time* is considerably lower. Therefore, for all but two participants, *Dedicated Workspaces* resulted in spending less time overall on task switching at the end of the experiment. Seven out of ten participants already made up for additional construction time when using *Dedicated Workspaces* after just three subsequent task resumptions.

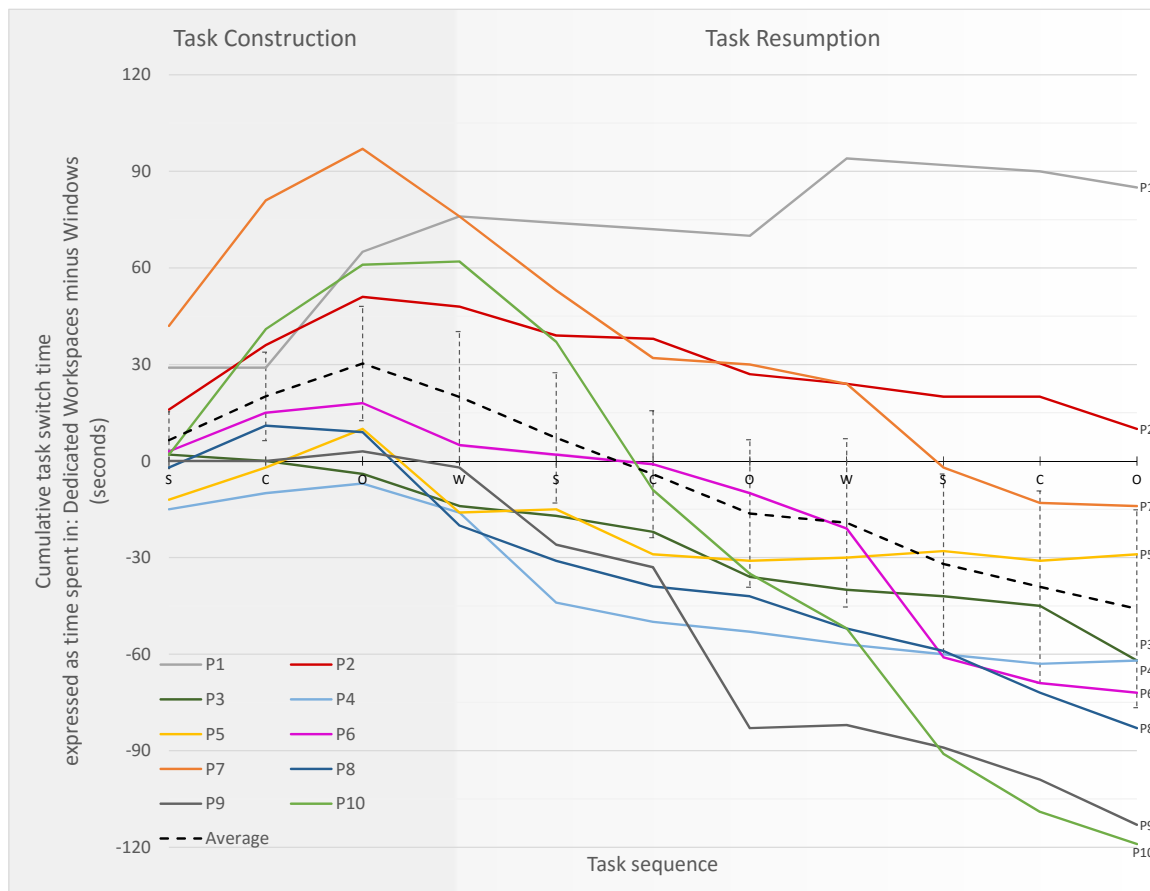


Figure 45: An overview of task switches throughout the experiment per participant. The first three task switches represent construction times where the workspace still needs to be set up. The remaining task switches represent resumption times. The y-axis shows total (cumulative) time spent on task switches over time when using dedicated workspaces, minus time spent on the same task switches under the traditional Windows environment.

Cognitive load was measured using Raw TLX [54], which assesses work load on six separate scales with 21 gradations. The overall measured task load index is the average of the six scales. An overview of averages and standard deviations of the separate TLX scales is shown in Figure 46. A paired one-tailed *t*-test was used to analyze overall *Cognitive Load* in both conditions, expecting a reduced cognitive load when using *Dedicated Workspaces*. A significant difference was found ($t(15) = -2.45, p < 0.05$) between *Dedicated Workspaces* ($\mu = 9.66, SD = 2.48$) and *Windows* ($\mu = 11.25, SD = 2.91$). The effect size using Hedges's $g_{av} = 0.56$. On average *Cognitive Load* is rated 14.1% lower when using dedicated workspaces.

COGNITIVE LOAD
significant difference

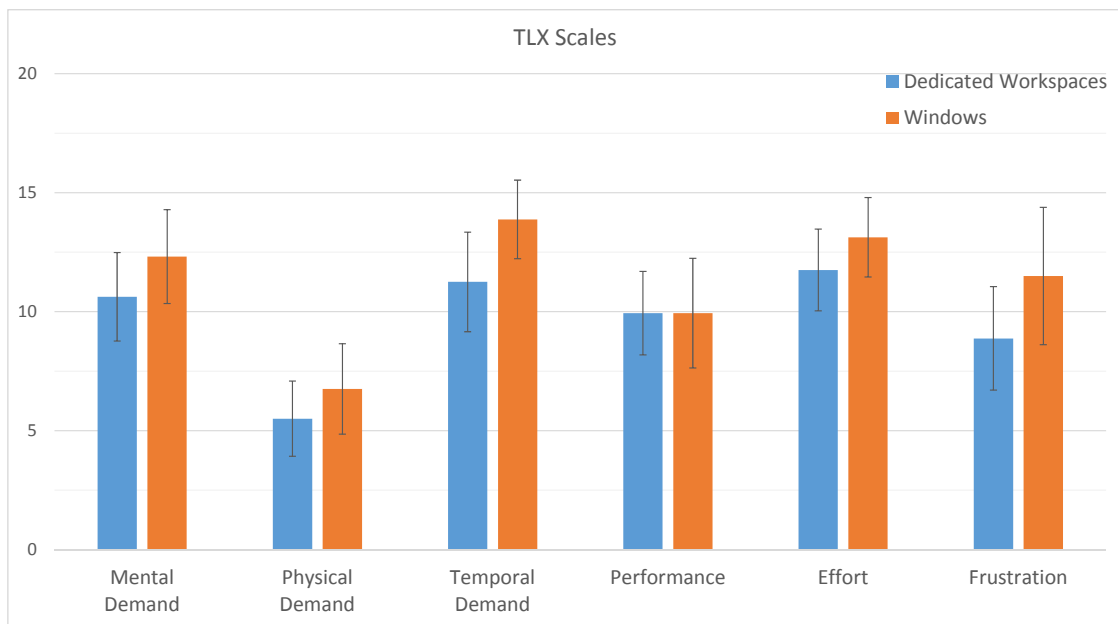


Figure 46: Breakdown of averages for each scale used as part of the Raw TLX test in the comparative sequential multitasking study.

Running two-tailed *t*-tests on the separate scales results in only finding a significant effect ($p < 0.05$) for *Temporal Demand* ($t(15) = -2.41, p < 0.05$) and *Effort* ($t(15) = -3.30, p < 0.01$). For *Temporal Demand* the mean of *Dedicated Workspaces* ($\mu = 11.25, SD = 4.19$) was lower than *Windows* ($\mu = 13.88, SD = 3.30$), Hedges's $g_{av} = 0.66$. On average *Temporal Demand* is rated 23.4% lower when using dedicated workspaces. For *Effort* the mean of *Dedicated Workspaces* ($\mu = 11.75, SD = 3.44$) was lower than *Windows* ($\mu = 13.10, SD = 3.34$), Hedges's $g_{av} = 0.38$. On average *Effort* is rated 11.5% lower when using dedicated workspaces.

Mainly Temporal Demand and Effort differed.

Paired two-tailed *t*-tests were used to analyze task *Productivity* and *Accuracy* in both conditions for all four tasks ($N = 16$). There are no significant effects ($p < 0.05$) between *Dedicated Workspaces* and *Windows* for any of the tasks on *Productivity* or *Accuracy*, of which an overview is provided in Table 13.

PRODUCTIVITY AND ACCURACY
no difference

Significant learning effects were observed.

To inspect the severity of learning effects, paired one-tailed t -tests were used to analyze task *Productivity* and *Accuracy* of tasks between *Session 1* and *Session 2*: the first, and second condition participants were exposed to respectively. A significant difference was found between *Session 1* and *Session 2* for *Productivity* on all four tasks, as summarized in Table 14. On average participants performed 20.8% more work overall in *Session 2* than in *Session 1*.

	WRITING		SEARCHING		COMPARING		ORGANIZING	
	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>
p	0.51	0.63	0.63	0.24	0.66	0.80	0.39	0.29
t	0.67	0.49	0.50	1.21	0.44	0.26	0.88	1.09
DW μ	0.48	0.99	0.42	0.83	0.45	0.84	0.46	0.92
Win μ	0.49	0.99	0.40	0.79	0.43	0.83	0.43	0.89
DW SD	0.21	0.01	0.20	0.13	0.21	0.11	0.26	0.13
Win SD	0.23	0.01	0.20	0.09	0.17	0.12	0.24	0.20

Table 13: Overview of t -tests comparing productivity and accuracy of all tasks between dedicated workspaces (DW) and a traditional desktop environment (Win).

	WRITING		SEARCHING		COMPARING		ORGANIZING	
	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>	<i>prod.</i>	<i>acc.</i>
p	< 0.001	0.44	< 0.001	0.21	< 0.001	0.13	< 0.001	0.19
t	4.25	0.79	4.91	1.32	4.26	1.62	4.28	1.36
g_{av}	0.30	-	0.41	-	0.42	-	0.41	-
S1 μ	0.45	0.99	0.37	0.79	0.40	0.82	0.39	0.89
S2 μ	0.52	0.99	0.45	0.83	0.48	0.85	0.50	0.92
S1 SD	0.20	0.01	0.19	0.12	0.17	0.12	0.23	0.20
S2 SD	0.23	0.01	0.20	0.10	0.20	0.11	0.25	0.13

Table 14: Overview of t -tests comparing productivity and accuracy of all task between the first session (S1) and the second session (S2) participants took part in.

9.4 STUDY 2: IN-DEPTH ANALYSIS

An in-depth analysis of task switching was conducted.

Current window managers have evolved considerably since their original inception to allow having more and more simultaneous windows open and providing different mechanisms of switching between them. Although some studies have observed different strategies by which users organize application windows when using virtual desktops [111] and when using multiple monitors and computers [22, 52], these studies have only provided high-level insights into multitasking and have not considered the detailed processes and strategies which take place *while* switching between tasks. As argued by Mulder et al.

[95]: “the heart of what information overload really is, may very well lie *between* tasks rather than *within*.” For example, in the first study (Section 9.3) it was shown that switching between tasks when using dedicated workspaces is not only faster, but also reduces cognitive load during multitasking [62]. Additionally, a high variance in task resumption time when using the traditional desktop environment was observed among participants (Figure 44). Therefore, to analyze what constitutes these differences, the goal of this second study is to investigate *how window managers support the user in switching between separate tasks during sequential multitasking*.

PARTICIPANTS Seven participants (age 22 to 33, 6 male) with varying backgrounds (research, marketing, game design, student) were recruited for the study. Participants with sufficient, but differing, levels of experience with Microsoft Windows were selected to investigate how experience affects task switching. On a scale from Novice to Expert, no participants selected Novice, three selected Average (P₂, P₅, P_x³), one selected Advanced (P₁), and three selected Expert (P₃, P₄, P₆). All participants are considered knowledge workers since they reported using a computer the majority of the day for both work and leisure ($\mu = 10.9$ hours, $SD = 4.4$ hours). All participants indicated engaging in multitasking on a regular basis, working on several activities in parallel ($\mu = 2.8$ activities, $SD = 0.4$ activities). No compensation was given for participating in the study.

MATERIALS A dedicated Windows 7 notebook with a 15.6 inch screen and external mouse was used for the study. Participants were allowed to configure the notebook to their liking (e. g., change the keyboard layout and default browser). Both recently accessed documents and browsing history were cleared prior to each test. A video camera was set up so that the screen, the participant’s hands, and posture were in view. Additionally, the screen, webcam (facial expressions), and important key strokes (e. g., ALT-TAB) were recorded using TechSmith Camtasia 8.

Windows 7 includes a number of features designed to support multitasking (Figure 47). All open application windows are depicted as icons on a taskbar. Hovering over them shows a window preview. Windows of the same application are grouped together, in which case the preview shows multiple windows. Clicking the taskbar icon, window preview, or the window itself brings it to the foreground (on top of all other windows). Prolonged hovering over thumbnails causes all open windows except the highlighted window to be hidden. Windows can be positioned freely, resized, minimized to the taskbar, maximized full screen, or docked side-by-side (Figure 39).



7



An unmodified Windows 7 environment was recorded.

Windows supports various features for task switching.

³ This participant is excluded from data analysis due to complications during the study.

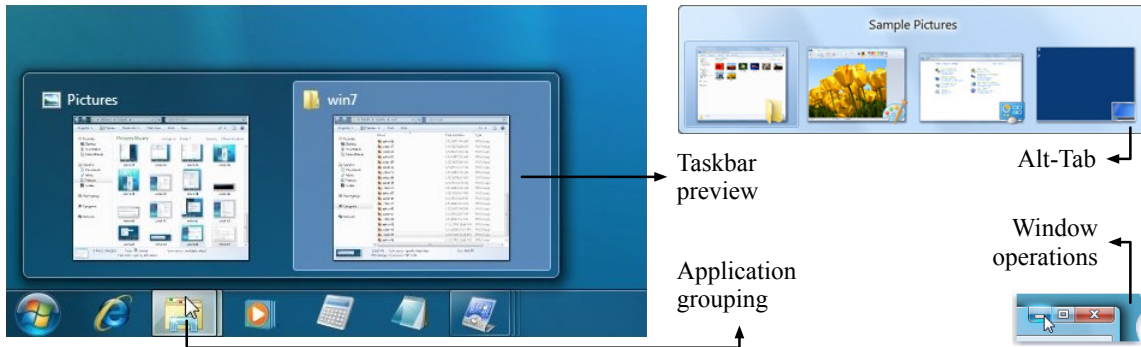


Figure 47: Window manager features of Windows 7 (source: <http://windows.microsoft.com>).

These functions are also available using shortcut keys. An additional shortcut key hides all windows, thus revealing the desktop. Lastly, pressing ALT-TAB brings up an overview screen which displays all open application windows. From here, the user can navigate between windows by repeatedly using ALT-TAB until the desired window is selected. Upon release the latest selected window is brought to the foreground. Alternatively, the mouse can be used to select a window directly from the overview.



1. *briefing*
2. *training*
3. *evaluation*

PROCEDURE The study comprised three distinct phases: briefing, training, and evaluation. During the briefing participants filled out a questionnaire assessing their overall computer literacy and the degree of multitasking they engage in on average (Table 15). Meanwhile, the notebook was set up to reflect the users' preferences, including their preferred web browser. The training phase (10 minutes) and evaluation phase (50 minutes, 12 task switches) followed the earlier described procedure (Section 9.2).

Age				
Job title				
Gender	m	f		
OS X familiarity	nov	avg	adv	exp
Linux familiarity	nov	avg	adv	exp
MS Windows familiarity	nov	avg	adv	exp
Type of activities performed.				
On average, number of activities performed throughout the day.				
On average, number of concurrent activities.				
On average, number of hours a day using a computer.				

Table 15: The questionnaire assessing computer literacy and degree of multitasking users engage in on average.

DATA ANALYSIS Two researchers collaboratively reviewed the video recordings on a large wall display. ChronoViz [45] was used to synchronize, play back, and annotate the data streams. This setup allowed simultaneous observation of posture, verbalizations, facial expressions, and interactions of the participants during task switching (Figure 48). Other segments were ignored. Throughout several iterations, a coding framework emerged, providing a full breakdown of task switches based on discrete events (Table 16). All task switches were reinspected until no further changes to the framework needed to be made to account for all observed interactions. Using the resulting coding framework, a third researcher independently validated all annotated data, leading to revisions (after discussion) where annotations were unclear, or considered inconsistent with the framework.

Annotation consisted of two main phases. First, for each task switch three points in time were identified: (1) *the beginning of the alert*, identified by the voice of the experimenter; (2) *the start of the task switch*, identified by the first interaction aimed at preparing the environment for the next task (e.g., minimizing a window, or moving the cursor towards the taskbar to open a new resource); and (3) *the start of work on the next task*, traditionally associated with the first recordable input users perform on the task set (e.g., a tower of Hanoi move [94])—in this study referred to as *objective resumption*. However, observations showed that objective resumption does not always coincide with the time at which users noticeably start working on the next task. Therefore, a more subjective measure was included—*subjective resumption*—also taking observable mental readiness into account. Example indications for this are: users' verbal utterances (e.g., "oh yes, here I was!"), physical movements (e.g., fingers used as pointers, relaxing before starting a task, facial expressions), and cursor movements (e.g., hovering over a relevant point in the task set).

Second, the identified task switch intervals were fully annotated by playing back the recording at $\frac{1}{4}$ of the original speed. During a first pass, task switch intervals were broken down into intervals denoting interactions with *resources* of the system ('Action' categories in Table 16). Resources are defined as containers of content relevant to the task which can be shown or hidden separately: e.g., files, folders, and tabs. During a second pass, an intent and feature of the user interface used was assigned to each of these interactions ('Intent' and 'Interface' categories respectively in Table 16). Lastly, an additional attribute allowed highlighting whether or not an action could be considered erroneous, given the intent ('Modifier' category in Table 16). Five examples of fully coded task switches are visualized in Figure 52.



Recorded data was coded, resulting in a breakdown of task switches.

Identified in first phase:

1. beginning alert
2. start of task switch
3. objective resumption
4. subjective resumption

Identified in second phase:

1. interactions with resources
2. intent assigned to interactions

<u>ACTION</u>	Time intervals further breaking down task switches.
<i>Interaction</i>	When users provide input to the task set through either the keyboard (e. g., writing) or the mouse (e. g., selecting text, navigating menus to access operations like saving).
<i>Navigation within</i>	When users navigate within an active task set while the majority of the content of the task set is already visible (e. g., the use of arrow keys to move the cursor within a text; scrolling using either the mouse wheel or scrollbar; moving, minimizing, or resizing a window; reading; concentrated eye movements within the active task). This does not require an application window of the task set to have focus (i. e., be active). The same resource remains visible during the annotated interval. As opposed to the ' <i>Interaction</i> ' category, the content (or selection) which is part of the task set does not change.
<i>Navigation to opened</i>	When users restore an open resource, while the required content of the task set is not yet visible. This action can be directed at fully hidden resources (e. g., accessing the taskbar, using ALT-TAB) as well as partially hidden resources in the background (e. g., bringing windows covered by others to the foreground). As opposed to the ' <i>Navigation within</i> ' category, the majority of the content of the resource is hidden. The resource does not need to end up visible at the end of the annotated interval. For example, when using the taskbar preview function of Windows to thoroughly inspect a resource for a long period of time, this is annotated as ' <i>Navigation to opened</i> ', followed by ' <i>Navigation within</i> '.
<i>Navigation to closed</i>	When users open a closed resource part of the task set, when currently not within the task set. Most commonly, when no application window for the resource is open. However, resources are defined as containers of content which can be shown or hidden separately, thus reusing, or opening new tabs is also considered as navigation to a closed resource. Likewise is navigating to a file path outside of the current task set using the file explorer. When a resource is already open, but the user's strategy follows that of opening a closed resource, the interval is still labeled as ' <i>Navigation to closed</i> '.
<i>Pause</i>	When users interrupt their current action, e. g., by repositioning themselves, by verbally expressing relief, or by closing their eyes for an extended duration.
<u>INTENT</u>	Applied to actions, capturing their purpose.
<i>Identify</i>	When the purpose of the action is the identification of the task description or task resources, rather than making progress on it. E. g., figuring out which resource of the task set is required. This is generally observed at the start of task resumption.
<i>Reorganize</i>	When the purpose of the action is to reorganize the task set. This intent can be observed both before the start of the task switch (e. g., leaving visual clues to facilitate future resumption) as well as after (e. g., restoring required resources to resume work).
<i>Task</i>	When the purpose of the action is to make progress on the current task. When the user reorganizes the workspace while simultaneously working on the task, the ' <i>Task</i> ' category takes precedence over ' <i>Reorganize</i> '.
<u>INTERFACE</u>	Applied to actions: the user interface used.
<i>Taskbar</i>	Using the taskbar (including preview functionality) to navigate to a resource.
<i>Alt-Tab</i>	Using the ALT-TAB shortcut to navigate to a resource.
<i>Foreground</i>	Navigate to a window which is partially or fully visible in the foreground.
<u>MODIFIER</u>	Applied to actions.
<i>Error</i>	Used to highlight an erroneous action given the intent, observable from the data (e. g., the resumption of a wrong resource, the unjustified use of a shortcut).

Table 16: Framework used to analyze task switches. First 'action' intervals are coded, after which 'intent', 'interface', and 'modifier' categories are assigned to them.

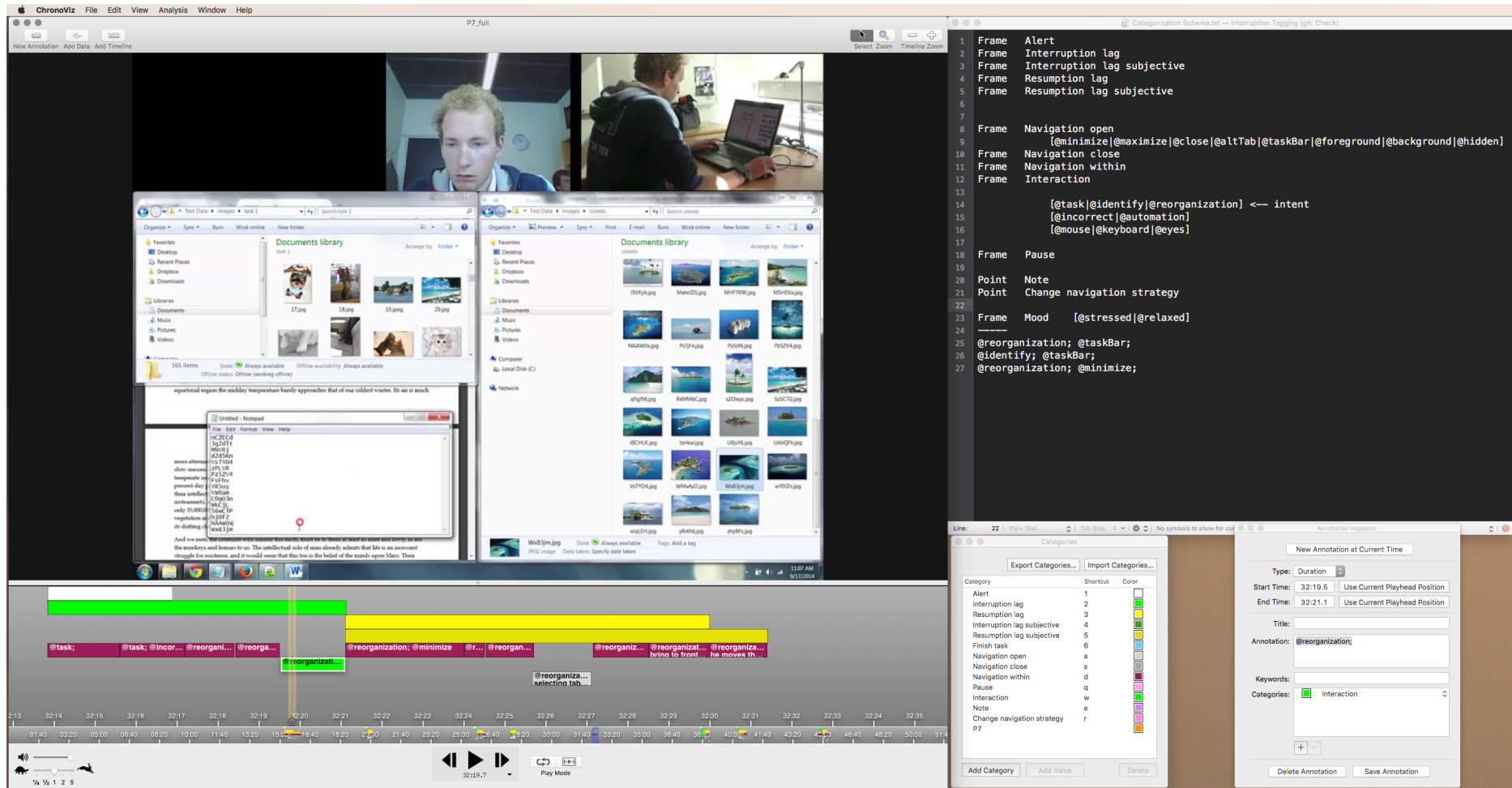


Figure 48: Data analysis of task switches in ChronoViz [45].



A full breakdown of all task switches is quantified.

44 task resumptions, covering 27 m 11 s of data were fully coded.

DISENGAGEMENT AND RESUMPTION TIME

HIGH-LEVEL RESULTS The outcome of the data analysis is a full breakdown of all observed task switches, following the framework which was developed throughout several iterations of thematic coding (Table 16). These results demonstrate how the framework can be used to quantify how well a window manager supports task switching, and can pinpoint possible areas for improvement. To demonstrate, common findings and peculiar occurrences will be presented, quantifying them according to the categories listed in the task switching framework. The main focus lies on investigating actions with the intent of *reorganizing* the task set, as these are the ones directly supported by window managers. In addition, the full coding of a set of representative task switches is visualized in Figure 52.

Out of the 12 task switches in the task sequence Figure 40, the initial four task switches participants were asked to start working on a new task; the remaining eight, they had to return to a previously constructed task. Therefore, the initial four task switches were not included in the data analysis as they are not representative of task resumption. Data of one of the participants had to be dropped due to a miscomprehension of the experiment which resulted in non-comparable data to the other participants: he did not resume tasks, rather restarted them from the beginning at each task switch. P4 initially did not anticipate having to return to tasks, hence closed task resources. His first four task resumptions were therefore not considered (task switch 5 through 8). This results in a total of 44 annotated task switches representative of task resumption, covering 27 minutes and 11 seconds of data (out of five hours of recorded material of participants working on the tasks).

Figure 49 shows the average disengagement and resumption time per participant. On average participants spent 8.9 s when suspending tasks (SD = 9.8, Min = 0.7, Max = 43.5), and 25.5 s when resuming tasks (SD = 15.0, Min = 8.0, Max = 72.3). In addition to subjective resumption time, the difference with average objective resumption time is shown. On average, objective and subjective resumption time differ by 4.8 s (SD = 6.6, Min = 0.0, Max = 36.7).

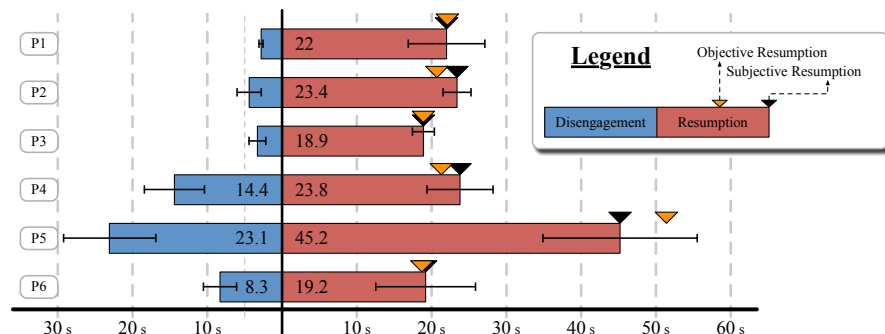


Figure 49: Average disengagement and resumption time per participant.

Figure 50 shows a percentile breakdown of actions and intents of all task switches per participant for both the disengagement and resumption stage. Time intervals where objective resumption time exceeded subjective resumption time were dropped, as they were not deemed part of the task switch. There are no actions with the intent of identifying the task during the disengagement stage, as this would indicate the start of the resumption stage. Similarly, during the resumption stage there are almost no actions with the intent of working on the task (only 2 short instances), since such a first action generally denotes the end of the task switch and full resumption of the new task.

In addition, Figure 50 highlights the percentage of erroneous time intervals per type of action and intent (e.g., retrieving an incorrect resource, saving an incorrect file). Half of the task switches (22 out of 44) contained one or more erroneous actions. Table 17 shows the distribution of error instances across participants. Eight out of 43 errors could be attributed to users performing ingrained actions part of task switching, with limited observable attention. Most notably, P1 failed six times in opening a resource when using ALT-TAB, after which he fell back to using the taskbar. In one occasion while minimizing windows, P3 used a shortcut operation to save a text file which did not need any saving as it did not need to be modified as part of the experiment. Lastly, P4 intently investigated window thumbnails when using the taskbar, only to realize later he did not know which resource he was looking for. The remaining errors mainly involved resuming incorrect resources (noticeable in navigation to opened) and typos or erroneous clicks (noticeable in interaction).

PERCENTILE
BREAKDOWN

Half of the task switches contained errors.

	P1	P2	P3	P4	P5	P6
# ERRORS	11	2	3	4	20	3
TIME	36.1 s	7.8 s	11.1 s	23.2 s	85.0 s	6.1 s
PERC.	18.2%	3.5%	6.2%	15.2%	15.6%	2.8%

Table 17: Time and percentage of total task switch time spent on errors for each participant.

Raw NASA-TLX [54] was used to assess cognitive load. Averages for all subscales are shown in Table 18. On average work load was rated 8.8 out of 21 (SD = 0.8, Min = 7.8, Max = 10.0).

COGNITIVE LOAD

	MENTAL	PHYS.	TEMP.	PERF.	EFFORT	FRUSTR.
μ	9.33	6.83	13.50	7.33	11.17	4.83
SD	5.01	3.82	2.26	3.88	4.17	0.75

Table 18: Breakdown of averages for each scale used as part of the Raw TLX test in the in-depth study on sequential multitasking.

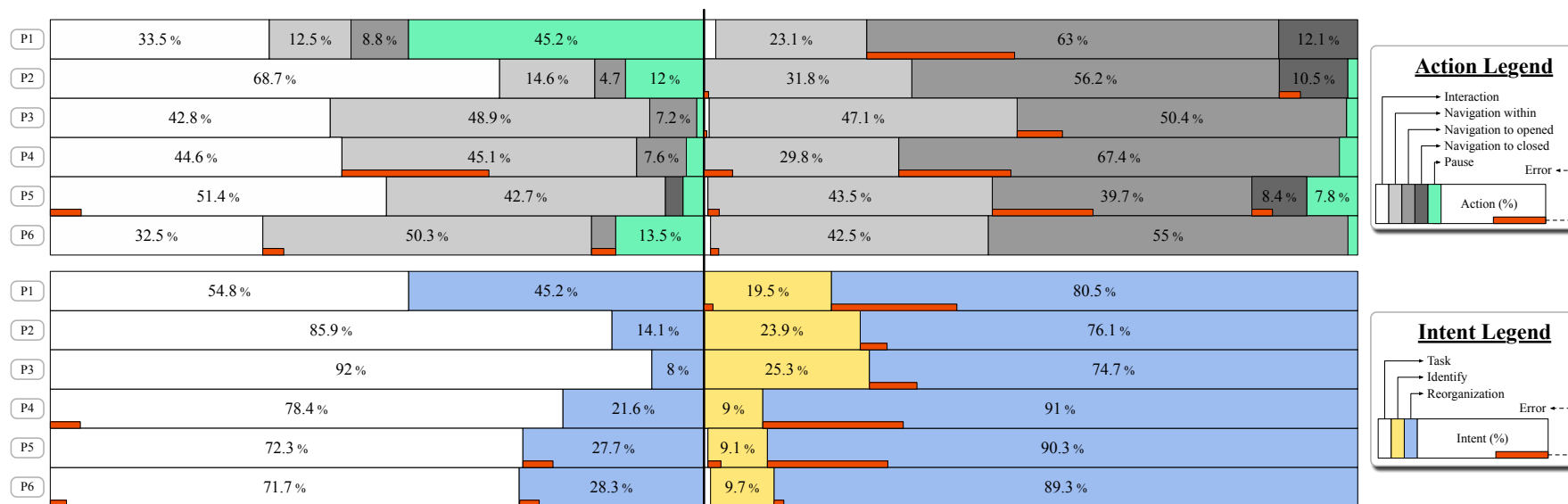


Figure 50: Percentile breakdown of actions and intents of all eight task switches per participant (except just four for P4), for both the disengagement and resumption stage.

ACTIONS AND INTENTS With a high-level overview of the makeup of disengagement and resumption time in place, it is now possible to describe what these containing actions and intents entail. The main interest of this study is to gain a better understanding on how window managers support *reorganizing* the workspace (Figure 51). Therefore, these are the main time intervals which will be presented in the subsequent results. Table 19 lists the percentages as part of *reorganization work* for each of the actions from the framework for both the disengagement and resumption stage, thus excluding actions with the intent to identify or to work on tasks.

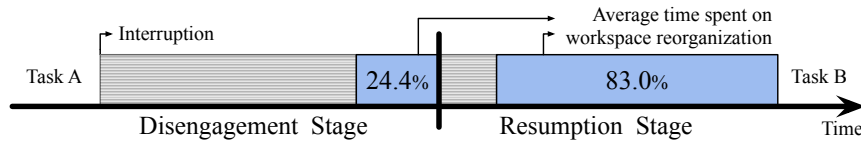


Figure 51: The average percentage of time spent during task switching on reorganizing the workspace.

	INTERACT.	WITHIN	OPENED	CLOSED	PAUSE
□	30.2%	21.4%	1.8% (1 ⁱ)	0%	46.6%
▷	0.9% (3 ⁱ)	39.0%	50.9%	6.5%	2.7%

Table 19: The makeup of reorganization work during disengagement (□) and resumption (▷). Rare instances: ⁱ

After having received the alert to switch to the next task, participants generally finished up their current subtask (e. g., Figure 52, 3). In just one instance this did not occur (Figure 52, 4), as P1 had just finished a subtask when the alert was given. Participants spent the remaining disengagement time on reorganizing the task set for future resumption (e. g., saving resources, renaming files, highlighting text, and leaving notes in text files) or taking a break (e. g., Figure 52, 2). In 15 cases there was no reorganization work, thus disengagement time constituted solely wrapping up work on the previous task (e. g., Figure 52, 1).

An overview of the average breakdown of actions during organization work is shown in Table 19. Reorganization work within the disengagement stage for the most part is composed of actions facilitating future resumption of the task, namely *interactions with* and *navigation within* resources. Interactions mainly reflect save and rename operations. Navigation within reflects users repositioning, resizing, and scrolling within windows in order to position them for later resumption. However, only half of the participants (P4, P5, P6) prepared tasks for future resumption on a regular basis; P1 never did, and P2 and P3 only on a single occasion. All participants (P1 in particular) regularly took *breaks* during the disengagement stage, which lasted on average



Actions labeled as reorganization work are analyzed in more detail.

DISENGAGEMENT
STAGE (□):

1. finish subtask
2. reorganize

Reorganization to:

1. facilitate future resumption
2. short break

1.8 s per task switch ($N = 18$, $SD = 1.2$, $Min = 0.7$, $Max = 5.9$). In particular, 14 out of the 44 task switches ended with a short break prior to the start of the task switch (e. g., [Figure 52, 3](#)). This break was generally associated with a change in posture (leaning backwards) and temporarily closing one's eyes. *Navigation to opened* occurred in just one instance (2.2 s) during reorganization work in the disengagement stage. P6 considered opening a resource to resume the next task, but aborted this action in order to write one final character acting as a placeholder for future resumption of the task.

RESUMPTION
STAGE (▷):

1. identify resources
2. set up workspace

At the start of the resumption stage many participants could not recall the exact resources they needed in order to resume the task (31 out of 44 task switches), in which case they commonly would reopen the task description. When this occurred, it took on average 6.0 s before participants actually continued resuming the required task ($N = 31$, $SD = 3.4$, $Min = 1.9$, $Max = 18.8$). The remaining time (disregarding a few short task intervals) was spent on setting up the workspace to start work on the new task (reorganization averages shown in [Table 19](#)). This took in total on average 21.3 s ($N = 44$, $SD = 13.2$, $Min = 7.8$, $Max = 54.6$).

Reorganization to:

1. navigate to open resources (or retrieve closed)
2. short break
3. navigate within resource

On average, *navigation to opened* occurred 2.9 times per task switch ($SD = 0.9$) with the purpose of reorganizing the workspace during task resumption, taking up 10.3 s in total ($N = 44$, $SD = 5.2$, $Min = 1.5$, $Max = 26.4$); *navigating within* resources 3.6 times ($SD = 1.9$), taking up 8.8 s in total ($N = 43$, $SD = 8.7$, $Min = 1.3$, $Max = 41.1$). When *pauses* occurred (14 out of 44 task switches), they were always indicative of having restored the necessary resources, and were followed by navigating within the retrieved resource to find the location where to continue work (e. g., [Figure 52, 2](#)). These pauses on average lasted 2.6 s ($N = 14$, $SD = 2.2$, $Min = 0.2$, $Max = 7.7$). In a few instances, participants closed resources as part of ongoing work and had to reopen them during the resumption stage using either Windows explorer or by reopening browser windows or tabs. *Navigation to closed* took up 7.2 s in total on average ($N = 10$, $SD = 3.6$, $Min = 3.0$, $Max = 13.5$). Lastly, just three actions during the resumption stage represented *interactions*. During these, files were saved (0.5 s, 0.6 s) and renamed (6.2 s).

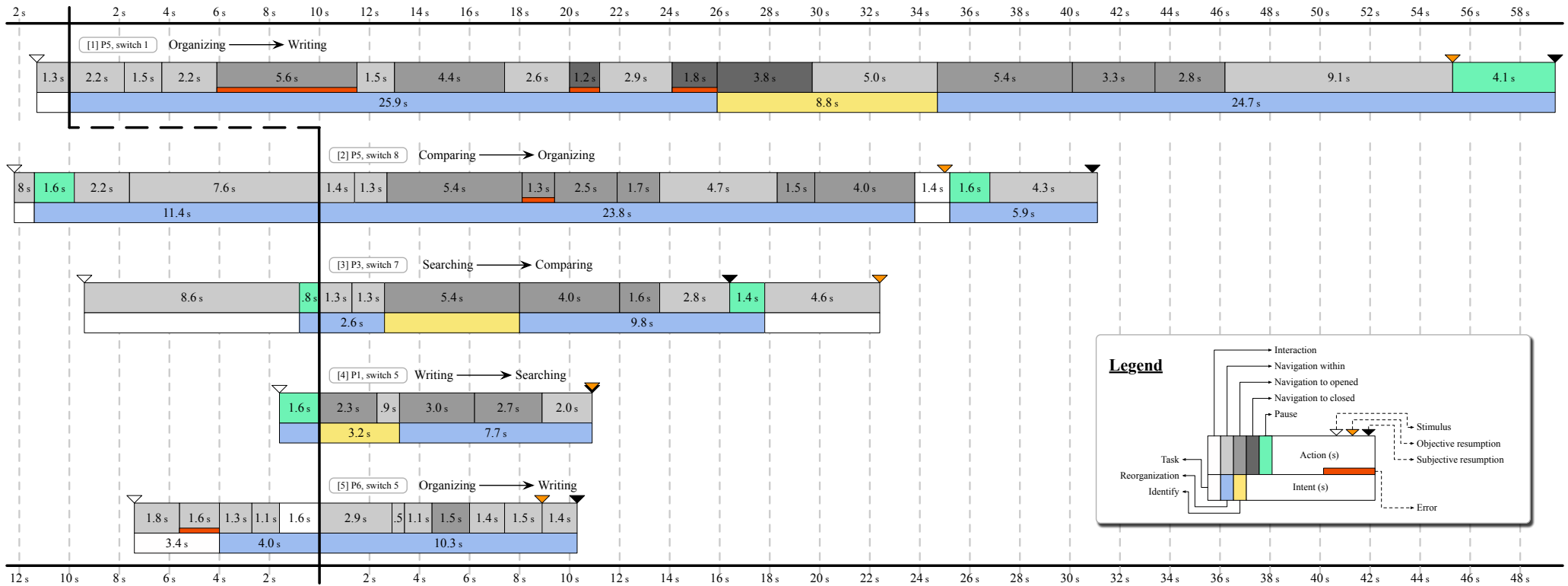


Figure 52: Detailed breakdown of representative task switches for recurring observations.



Users rely predominantly on the taskbar during task switches.

WINDOW MANAGEMENT STRATEGIES All actions marked as either using ALT-TAB or the taskbar, except one, are marked as *navigation to opened*. One action using the taskbar was categorized as navigation within, as P5 spent a total of five seconds investigating a set of window thumbnails in order to select the right one. Therefore solely the window management features used as part of navigating to opened resources will be analyzed further. On average, this took 3.5 s ($N = 165$, $SD = 2.4$, $Min = 0.5$, $Max = 20.0$). Predominantly, the taskbar was used to this end by all participants (Table 20). Only two users (P1 and P3) occasionally used ALT-TAB. Remarkably, P3 never retrieved a window by selecting a partially visible window from the foreground. Using the taskbar to open resources on average took 3.8 s ($N = 132$, $SD = 2.5$, $Min = 0.5$, $Max = 20.0$); using ALT-TAB 3.2 s ($N = 8$, $SD = 2.2$, $Min = 1.5$, $Max = 7.5$); opening from the foreground 2.1 s ($N = 25$, $SD = 1.1$, $Min = 0.9$, $Max = 6.2$).

	P1	P2	P3	P4	P5	P6
ALT-TAB	12.9%	0%	15.7%	0%	0%	0%
TASKBAR	83.7%	98.6%	84.3%	84.6%	87.1%	87.2%
FOREGROUND	3.4%	1.4%	0%	15.4%	12.9%	12.8%

Table 20: Features used for ‘navigation to opened’.

Different strategies:

1. window movement
2. side-by-side
3. overlapping
4. minimizing
5. tiling

Appropriation of window management features during task switching varied greatly from user to user. Participants used a combination of five different strategies. The first strategy (*Window Movement*: P5) involved extensive repositioning and resizing of windows (e. g., Figure 52, 2). Seemingly P5 was unaware of the window docking feature. In contrast, in a second strategy participants relied on strict side-by-side window configurations using this feature (*Side-by-side*: all except P5). This simplified switching between tasks by mainly relying on selecting two required windows from the taskbar. A third strategy involved overlapping windows in various ways, keeping the content of multiple windows partially visible (*Overlapping*: P1, P4, P6). Irrelevant windows were often minimized, revealing windows for the required task underneath. Later, the taskbar was used to bring them back to the foreground when resuming tasks. A fourth strategy consisted of minimizing all windows during disengagement (*Minimizing*: P3, P5). Interestingly, participants never used the shortcut key (show desktop) to this end. Possibly users were hoping to find relevant windows underneath (as observed when overlapping windows). A final observed strategy was to tile windows, placing them at semi-fixed positions (*Tiling*: P6). While working on a task, there was little overlap among foreground windows, and windows were rarely repositioned. This allowed P6 to keep more than two windows permanently visible (as in the side-by-side strategy) while working on a task.

9.5 TASK RESUMPTION AND CONSTRUCTION TIME

Prior systems supporting multitasking have consistently referred to task switching as a problematic practice that warrants attention. These systems typically assume that novel approaches (such as dedicated workspaces) would outperform the traditional window manager. For example, Bardram et al. [17] stated: “We have refrained from measuring task completion time. It is trivial to see that users of [our system] would out-perform standard Windows XP users if asked to perform tasks that require handling parallel tasks, accessing lots of digital material, [and] coping with frequent interruptions.” Despite the fact that dedicated workspaces are specifically designed to facilitate task switching, no prior studies have quantified task resumption time for window management with or without dedicated workspaces. The studies presented here are the first to quantify in detail what prior work has only assumed.

The first study shows it takes on average 10 seconds longer to switch between tasks under a traditional Windows 7 environment than when using dedicated workspaces (N = 13)⁴, thereby providing empirical evidence *to which degree* dedicated workspaces improve task resumption time. The second study investigated task switching under the same Windows environment (N = 6), and reported on similar task resumption times (Figure 53). It is important to keep in mind that these measures are representative of a heavy multitasking scenario (four tasks with multiple application windows), and that not all multitasking scenarios involve the same amount of application windows which need to be retrieved. Therefore, these results are best viewed as an indication in which order of magnitude the results of a large scale study would lie. Nonetheless, both studies provide useful insights into differences between participants and individual task switches, and which factors to control for in future studies.

Quantitative research on window management is lacking.

Results indicate dedicated workspaces are an order of magnitude faster.

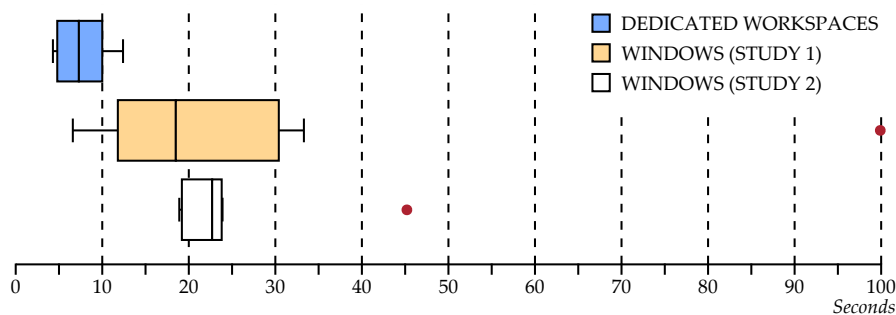


Figure 53: Box plots for task resumption times under both studies. Two outliers are identified bigger than 1.5 times the interquartile range.

⁴ Three experiments were conducted monitoring two participants simultaneously. During these experiments the researcher could only focus on measuring task resumption time for one of the participants, resulting in 13 (as opposed to 16) task resumption time measures.

Task resumption time increases steeply after a certain point of failure.

A high variance in task resumption time was observed in both experiments when using the traditional window manager. This indicates that not all users benefit from dedicated workspaces equally. Some participants⁵ encountered task switches lasting over one (P^2_5) to two minutes (P^1_{11} , P^1_{12} , and P^1_{13}) using the traditional Windows 7 environment, indicating a complete breakdown in finding the required resources to resume a previously suspended task. Two of these participants (P^2_5 and P^1_{12}) were inexperienced Windows users (OS X user), unfamiliar with some of the features of Windows 7. However, the other two participants were a consultant and clerk, using Windows 7 on a regular basis as part of their work. The consultant (P^1_{13}) at times expressed severe annoyance when being unable to retrieve the necessary resources, after which she would clean up her workspace entirely by closing all windows and start over. The clerk (P^1_{11}) struggled particularly when navigating between folders (using Windows explorer), and sometimes had to reopen the assignment description to look up the required path for the task.

Navigating to open and closed resources is time-consuming and error-prone.

The results from the second study confirm that participants spent a considerable amount of time on simply recreating window configurations they lost as part of switching to another task, indicated by *navigation to opened* and *closed* actions during the resumption stage (Figure 54). These actions do not seem to contribute to the overall goal of the task at hand and are shown to be error-prone. In the presence of a significant number of resources, participants struggled to effectively use the features offered by the operating system. Even though the Windows taskbar aggregates resources by application as a way to reduce taskbar clutter (more effective than ALT-TAB which lists all resources), users in the second study on average spent 3.5 s per resource that needed to be opened. Application grouping is a sub-optimal solution since documents opened using the same application are not necessarily related to each other contextually—they are not always part of the same activity.

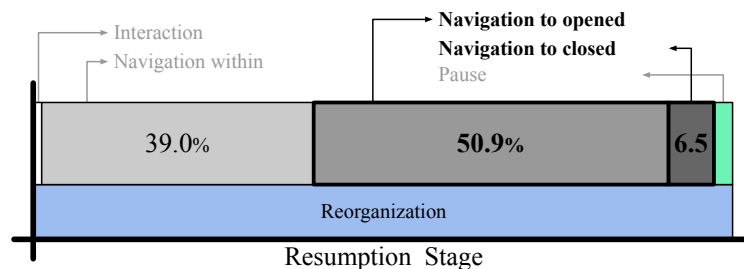


Figure 54: Average breakdown of reorganization actions during task resumption.

⁵ I will refer to participants from the first experiment as P^1 and from the second experiment as P^2 .

Many activity-centric systems, such as the ActivityBar [17], Giornata [136], co-Activity Manager [58], and Laevo [64] have shown through qualitative field studies that task-centric work environments (dedicated workspaces) help users during multitasking. What can be shown now through these findings is the exact impact that these systems have on task switching. Having a task set organized around one single task description makes it possible to automate navigating to open and closed resources during task resumption as the entire process can be reduced to identifying a task definition among a list of open parallel tasks, e.g., an activity name. In line with observations from both studies, it is anticipated that users often close resources because of the overhead of keeping them open; both studies show empirical evidence that severe detrimental effects can occur when too many application windows are open. Dedicated workspaces alleviate this problem by hiding irrelevant resources that might otherwise interfere while switching between tasks. However, the data from the second study suggests that further improvements can be made by providing tool support to more easily navigate within resources, i.e., helping users figure out where to continue work once all necessary resources have been retrieved (Figure 54).

Dedicated workspaces thus allow the user to set up a dedicated environment for each individual goal-oriented task, which is associated with a longer set-up time (13 seconds on average) within the user interface used in the first study (Laevo, Chapter 6). However, at the end of the experiment all but two participants spent less time on task switching when using dedicated workspaces than when using a traditional Windows 7 environment (Figure 45); for the remaining two participants the trend predicts a similar result given a longer experiment. Providing support for dedicated workspaces opens up the opportunity for rapid access to a multitude of long-lived workspaces (rather than just four), which I hypothesize would greatly benefit even experienced users. P¹² confirms this by stating:

[dedicated workspaces] would mainly be useful for tasks I haven't worked on for a longer period of time. Since [during the experiment] there is not much time in between task switches, using Windows 7, it is not that much more difficult.

What seemingly sets more 'expert' users apart is they take great care in making the work environment their own. This is in line with observations made by Kidd in "The Marks are on the Knowledge Worker" [77]:

Only the knowledge worker can give meaning to the marks on the paper or on the screen and they do not know and cannot predict what this meaning is until they have been informed by it.

Dedicated workspaces automate navigation to opened and closed.

The construction cost for dedicated workspaces is quickly made up for by faster task resumption.

Expert users leave behind clear cues on where to resume work.

Participants introduced additional cues which allowed them to more easily disambiguate between tasks. For example, P¹₁ gave more meaningful names to documents so they could easily be recognized on the task bar during task switching. Rather than reusing a single Windows explorer window, some users opened multiple instances with different file paths open in them, positioned optimally for each individual task. The second study confirms these findings by observing that the purpose behind *navigation within* resources during disengagement (Figure 55) was to leave behind visual markers to facilitate later resumption. For instance, a common strategy included scrolling within a window so that the next subtask would be displayed first. For the same reason, participants *interacted* with resources: e. g., selecting text in a PDF file, or even typing reminders in text documents part of the task set. More experienced users thus seem to be able to keep more resources open, yet easily keep track of which tasks they belong to.

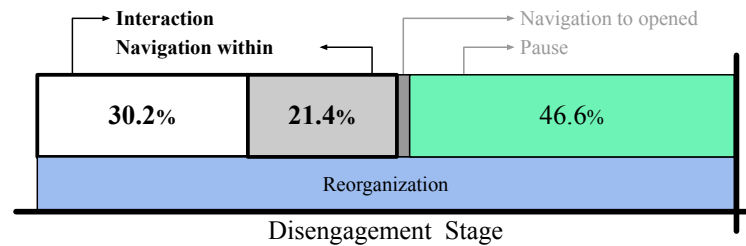


Figure 55: Average breakdown of reorganization actions during task disengagement.

Dedicated workspaces encourage leaving behind visual cues.

Dedicated workspaces encourage *all* users to label and group contextually related resources together, thereby rehearsing the task description, useful for later identification of the workspace. Such actions are indicative of formulating plans to return to the task at a later moment in time. This has been shown to alleviate the detrimental effects of unconsciously remaining focused on unfulfilled goals [92]. In addition, prior studies have shown that leaving behind visual cues reduces the amount of required time to resume a task [6], highlighting their importance as part of task switching. However, the observations made here show the kind of application-specific workarounds that participants need to use in order to leave behind such cues. Although dedicated workspaces simplify recognizing a task, visual cues are still needed to resume work within that task. Some text editors provide basic features in support of task resumption, for instance, by allowing documents to be reopened with the cursor positioned where it was last left behind. However, the results presented here indicate a need for more general cross-application support for leaving behind visual cues facilitating the resumption of suspended tasks, e. g., freehand drawings on top of application windows.

9.6 COGNITIVE LOAD AND PERFORMANCE

The first study found a significant difference in cognitive load between dedicated workspaces (as supported in Laevo) and a traditional desktop environment ($N = 16$). In the second study participants performed the same tasks and procedure, but only under one condition—the Windows environment ($N = 6$). As a result, the second study lasted considerably shorter. This might explain why participants indicated a lower cognitive load compared to both conditions in the first study (Figure 56). Raw TLX [54] was used to measure cognitive load, meaning the significant result reflects the participants' subjective preference for dedicated workspaces over a traditional desktop environment during heavy multitasking. These results indicate task switch interruptions are more disruptive (cause more annoyance and anxiety) under a traditional desktop environment.

Participants rated dedicated workspaces as less mentally demanding.

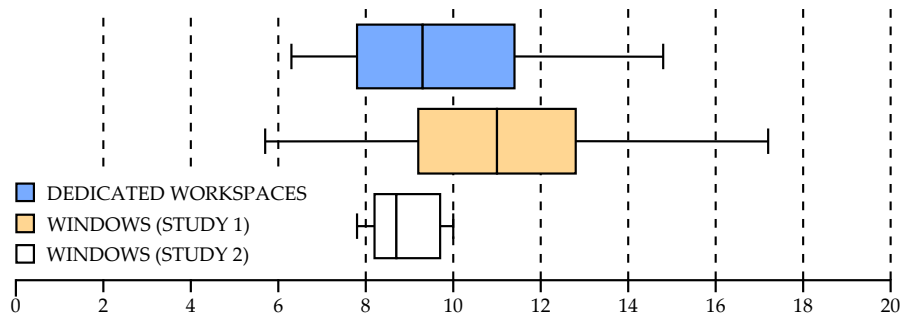


Figure 56: Box plots for Raw TLX measures under both studies.

Interestingly, in both studies it is mainly *effort* (how hard did you have to work to accomplish your level of performance) and *temporal demand* (perceived time pressure—was the pace slow or rapid) which influenced cognitive load⁶. The difference in effort can be explained by the additional necessary operations to switch between tasks under a traditional desktop environment. However, this does not explain why participants felt more rushed; under both conditions participants worked on the same tasks, and the incentive remained the same. Information overload—a possible cause of stress [95]—might be at play here. The mere visibility of previously suspended tasks might be what results in an increase of perceived time pressure. Even participants fluent in task switching under a traditional desktop environment seem to confirm this, e.g., P¹⁴ stated having more “breathing space” when using dedicated workspaces. In contrast, participants were not significantly more frustrated under a traditional desktop environment, contradicting the finding of the overall measure. However, quite a few of the participants seemed to be accustomed to heavy multitasking, thus struggling minimally with task switches.

A traditional window manager requires more effort and increases perceived time pressure.

⁶ Study 1: Figure 46, study 2: Table 18.

Task switches seem conceptually similar to secondary tasks in interruption studies.

Task switches thus seem to require a significant amount of concentration. An interesting pattern was observed in the second study which further emphasizes this finding: navigation to opened and closed resources in many cases was followed by a pause. Only after taking a short break, would users navigate within documents to find the exact location where to pick up work. Intrigued by this finding, the data was reinvestigated and very short subtle hints—hence not coded—were found indicating pauses at the end of most navigate to opened/closed sequences which precede navigations within; e. g., closed eyes, and small readjustments of posture. It can therefore be speculated that the entire sequence of reconstructing the work environment could be considered (and is experienced by users) as a task in and of itself (resumption times average around 20 seconds). In this sense task switching could be seen as a secondary task interrupting a primary task, conceptually similar to interruption studies.

The task sets used give rise to a high variance in task productivity and accuracy.

Interruption studies indicate that more errors are made right after an interruption, and that longer, more demanding interruptions lead to longer resumption times [94]. Therefore, the first study hypothesized a measurable effect on task performance. However, due to the systematic nature of the task sets used, severe learning effects were observed (Table 14). The resulting high variance in task productivity and accuracy eliminated the opportunity to measure any significant effects. Rather than trying to make task sets measurable, future studies could intersperse them with traditional cognitive tests (e. g., Stroop test, n-back) on which to measure progress and accuracy instead. This would allow participants to work on even more representative task sets. On the other hand, this would introduce additional interruptions which need to be controlled for under both conditions. An alternative approach could be, rather than measuring overall task performance, only measuring task performance during the interval immediately following task switches. Although differences in cognitive load can impact overall performance, interruption studies predict a bigger impact immediately following the interruption [94].

9.7 THREATS TO VALIDITY

The second study validates the resumption times measured in the first.

The choice of letting participants work within a real desktop environment (in order to increase ecological validity with respect to the resumption stage) imposed several practical limitations. Task resumption time measures could not be automated. In the first study task resumption was measured by a researcher using a stopwatch. Measuring resumption time using a stopwatch implies inaccuracies due to the experimenter's response time and possible bias. To prevent bias, a protocol was defined up front which defines what constitutes task resumption, based on prior research [6, 94, 131]. To optimize accuracy, rather than starting and stopping the stopwatch to measure time

intervals, only instants in time were noted. This allowed the experimenter to easily update a measure when a later observation indicated a task was not yet resumed. Time intervals were only calculated after a task switch had occurred. The similarity between measurements in the first study and the second (which used precise video annotations) indicates this approach to be adequate (Figure 53). In contrast to automated measures, a benefit is intent can be interpreted as well, as not all mouse or keyboard input on a task set implies task resumption; one application window, part of the task set, could be correctly retrieved and receive input, but a secondary window might still be needed which takes the participant some additional time to retrieve. In addition, verbal utterances by participants often provided rich cues about the exact time a task was resumed. The (at times) large differences in between subjective and objective resumption times measured in the second study show the importance of these cues (Figure 49).

One of the core challenges in setting up a laboratory experiment assessing productivity and accuracy of knowledge work, as influenced by a manipulated variable, is how to control for the knowledge work itself. Allowing participants to work freely on their actual work would provide maximum validity. This, however, makes any comparison across tasks impossible due to the 'non-routine' problem solving nature of knowledge work [77]. In such a setup, productivity and accuracy can vary widely from one session to another. Creating comparable task sets is a trade-off between the sensitivity of task performance measures, and the ecologically valid circumstances the methodology presented here strives for [49]. More concretely, equally spaced short subtasks simplify measuring performance and allow for shorter experiments, but are not as representative of knowledge work. In contrast, longer subtasks can be more ecologically valid, but by their nature will introduce more variance in performance measures. Using the task sets presented here, the effects would have to be of a considerable size in order to be measurable when running a study with a limited number of participants.

Given the experimental setup, the NASA-TLX test arguably does not provide sufficient sensitivity in order to easily interpret effect size beyond an indication of subjective preference of one condition over the other. Future work could investigate how to obtain and interpret objective measurements of cognitive load while performing knowledge work in a desktop environment. There is an opportunity to use biometric data; skin conductance, blood volume, and pulse rate in particular seem promising [61, 91], which are indicative of *arousal*. Arousal however is not indicative of *valence*—the pleasantness or unpleasantness of stimuli. Therefore, first there is a need for a better understanding of the different triggers which influence arousal during knowledge work in order to assess whether they are indicative of cognitive load or not.

Creating task sets for studies on sequential multitasking involves a trade-off.

Better measures for cognitive load are needed.

Part IV

DISCUSSION AND CONCLUSION

Our approach ... is to try to understand how we would design computer tools differently if they were focussed on supporting the act of informing people, rather than on storing or processing information on peoples' behalf.

— Alison Kidd [77]

The contributions presented in this dissertation are three-fold: (1) a theoretical framework for HCI is introduced—the *interaction framework*—used to outline a clear goal for activity-centric computing; (2) in pursuit of this goal, a conceptual model supporting *activity management*—the *activity life cycle*—is incorporated into two activity-centric computing systems—*Laevo* and *co-Laevo*; (3) results from an *in field evaluation of Laevo* and two experimental studies on *sequential multitasking* strengthen the case for activity-centric computing. In this chapter¹, I relate the empirical findings collected during these three studies to two open issues in activity-centric computing, which I identify as issues of *scalability* and *intelligibility*. In addition to validating the design of *Laevo*, this validates the suitability of the activity life cycle as a conceptual model for other interactive computing systems.

This dissertation contributes to:

1. *theory*
2. *technology*
3. *empirical findings*

10.1 THE HIDDEN COST OF TASK SWITCHING

In an observational study of knowledge workers, González and Mark [49] report that the time spent within a specific working sphere² before switching to another ranges from 7 m 41 s to 16 m 24 s. Considering an 8 h work day and using the data obtained in the second study on sequential multitasking (Section 9.4), it is possible to predict for each of the participants the daily cost of switching between tasks. Given that all observed task switches were fully coded, it is possible to disregard time intervals where participants were merely wrapping up work on a previous task after been given the notification to switch. Likewise, time intervals where participants were struggling to identify the necessary resources to resume a task could be dropped, since they are arguably caused by the participants not being the original owners of the tasks. Given this data (treating P²5 as an outlier), on average the knowledge workers analyzed in this study would spend between 9 m 45 s and 20 m 50 s of their day *reorganizing* their workspace during heavy multitasking (Table 21).

Extrapolating from study results, an estimate of time spent on task switching can be made.

¹ Parts of this discussion are adopted from earlier publications [64, 66].

² Working sphere: “units of work or activities that people divide their work into on a daily basis” [49].

	P ² ₁	P ² ₂	P ² ₃	P ² ₄	P ² ₅	P ² ₆
AVG	19.0 s	18.6 s	14.5 s	23.6 s	47.6 s	19.6 s
Ω ^[1]	9 : 44	9 : 04	7 : 04	11 : 29	23 : 12	9 : 32
Ø ^[2]	20 : 47	19 : 21	15 : 05	24 : 32	49 : 33	20 : 22

Table 21: Predicted time spent on reorganizing the workspace per day based on 16 m 24 s^[1] and 7 m 41 s^[2] estimate of time in between task switches.

The advantages of using dedicated workspaces far outweighs their cost.

Although these results are highly dependent on individual differences among participants and the specific conditions of the experiment within which they were measured, they clearly suggest a *hidden cost* associated with task switching. Retrieving a single resource as part of reconstructing a previously suspended task took on average 3.5 s. Not only does this rapidly add up over the course of a day, but as the first study indicates, it also introduces a perceived time pressure (measured as part of NASA TLX), which can easily be alleviated through the use of dedicated workspaces. This greater time pressure experienced by users when using a traditional Windows environment is hypothesized to arise due to *information overload*; some window management features can clearly overwhelm users during heavy multitasking (e.g., [Figure 57](#)). The overhead which is sometimes ascribed to setting up dedicated workspaces is shown to be negligible: the time lost on setting up a workspace (13.6 s on average) is almost instantaneously compensated for by faster task resumption ([Figure 45](#)). Thus, the *perceived overhead* seems far greater than the actual cost, as was confirmed by all participants during interviews in the evaluation of Laevo ([Chapter 8](#)).

“These small things do have a tendency to grow, and the next day it turns out I do sometimes have to continue working on them. In the long run I imagine I would structure even all smalls tasks within activities, because it still is more convenient to access when you do have to return to them. There is little overhead in creating them; the gain is bigger when you have to return to them.” – P¹₁

The framework for task switching could be considered a more general taxonomy.

As part of the data analysis during the detailed study on sequential multitasking, a framework was developed which formed the foundation for many of the presented insights. The framework underwent several iterative refinements and was validated by three researchers. Each new discovery within a task switch which triggered a change in the coding scheme was: (1) discussed among the researchers; (2) tested on previously coded data; (3) if accepted, further applied to the entire data set. The framework covers 27 m 11 s worth of data,

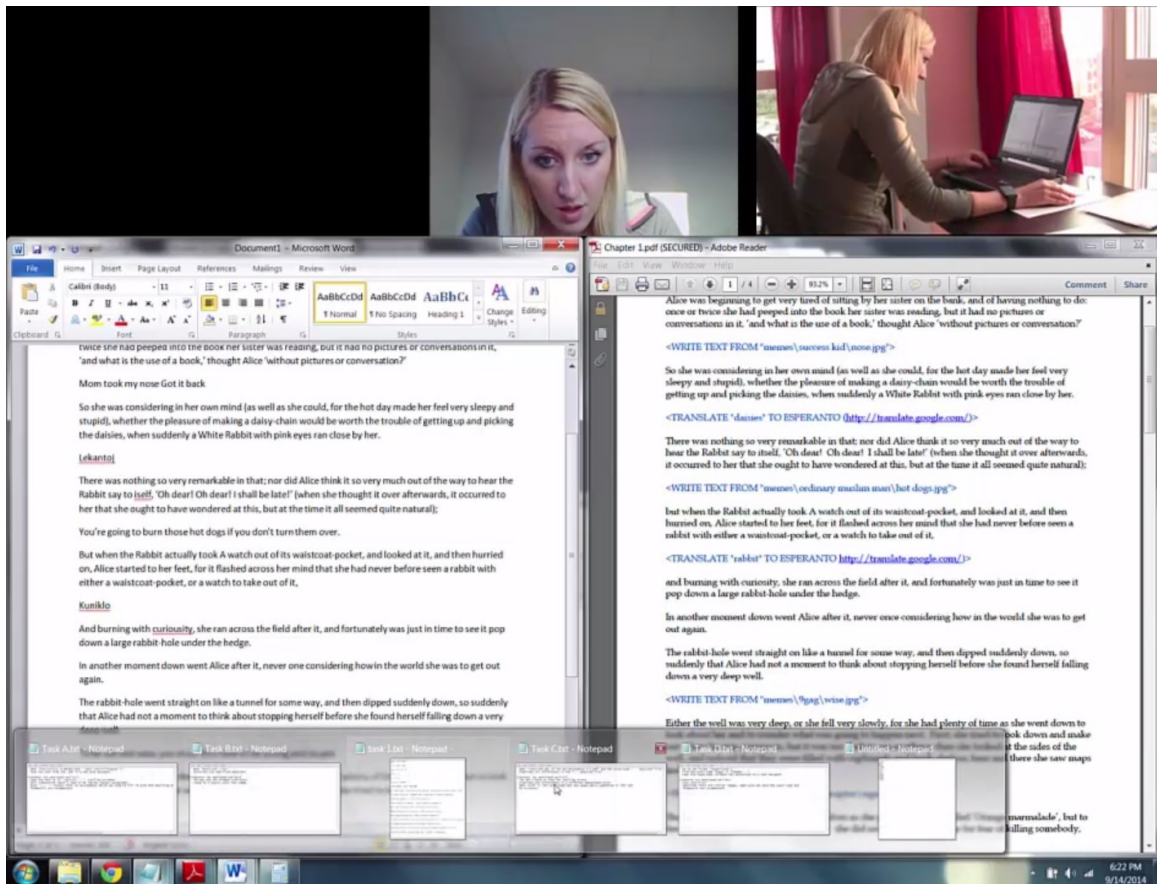


Figure 57: Information overload experienced by P22 during task switching using the taskbar.

covering 44 task switches, executed by participants with varying experience using the Windows 7 window manager. Thus, it is proposed to consider this framework to be a more general *taxonomy* for task switching, which could be used in future sequential multitasking studies for a variety of different window managers.

10.2 INTEGRATED KNOWLEDGE WORK

The empirical studies on sequential multitasking clearly show it is beneficial for knowledge workers engaging in heavy multitasking to structure their work within the context of dedicated workspaces. Using Laevo, on average resuming activities was 9.53 s faster ($\mu = 6.26$, $SD = 1.92$). However, over time, as users amass a great number of activities, the issue of retrieving a single resource from a large collection of open resources simply resurfaces in the form of retrieving an activity from a large collection of open activities. The temporal dimension for activity management introduced as part of Laevo provides a lightweight *scalable* solution, capable of visualizing activities of *varying scope* (from long-term projects to ad hoc tasks) in parallel. A prolonged evaluation of Laevo (lasting longer than two weeks)

Temporal activity management in Laevo resolves the issue of scalability.

is necessary to thoroughly evaluate this claim. However, given the experimental nature of activity-centric computing (often conflicting with existing practice), such an investment cannot be expected from participants who need to evaluate the system as part of their ongoing work. One user (me), however, did evaluate Laevo over the course of several months. Looking at the time line I constructed around July–August, 2013, gives an impression of how I personally appropriated long-term activities³ (Figure 58): the (ongoing) efforts towards the deployment of Laevo are visible, side-by-side with short-term activities representing the implementation of features, and activities representing arrangements for past and future music festivals.

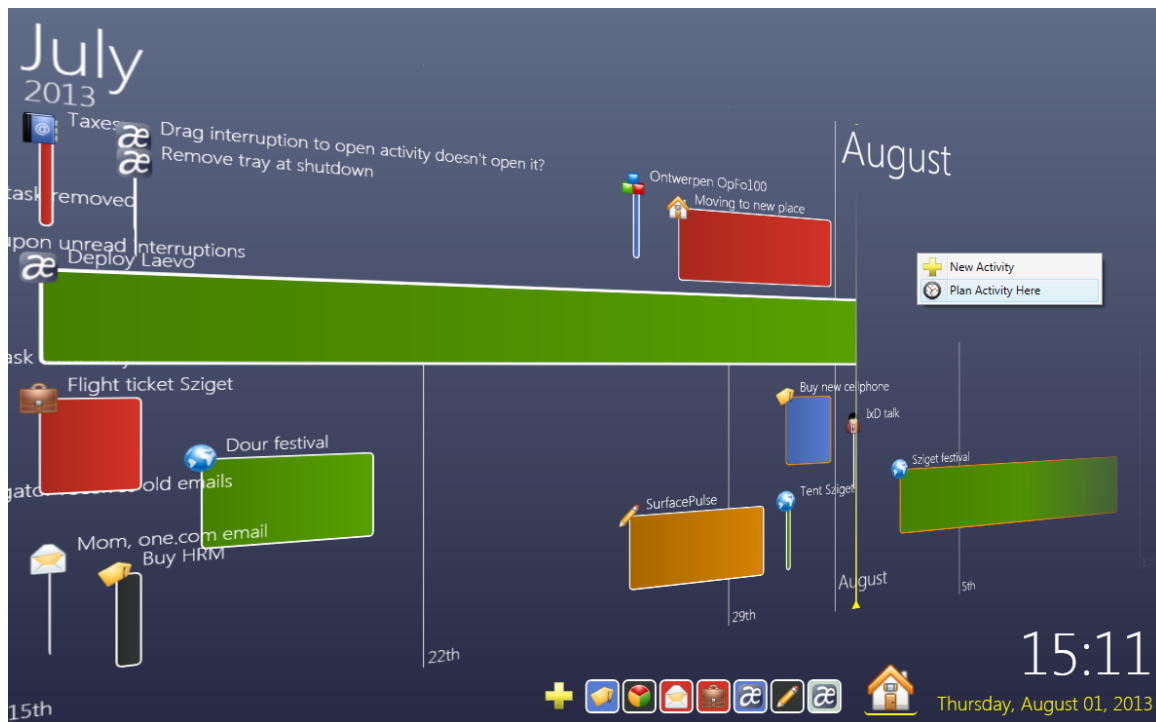


Figure 58: Real-world use of Laevo by the author of this dissertation.

Laevo supports structuring information in time, reducing information overload.

Structuring work within the context of activities, and providing strong provenance cues of their life cycle and interrelations with other activities, is how I envision to support what Whittaker refers to as *the curation lifecycle* [138]—the keeping, management, and exploitation of information. As part of everyday activity management, personal information is stored, and can easily be retrieved at a later moment in time. The rich time line visualization incorporates most of the required task types to easily access information [122] (overview first, zoom and filter, then details-on-demand), thereby reducing information overload. This was confirmed by participants, who reported being more *focused*, *productive*, and *efficient* in their work during the evaluation of Laevo.

³ Similarly, as part of the UIST publication [64], I provided footage of my personal time line around July 2014: <https://www.youtube.com/watch?v=BAC7sBvViFg>

The in depth study on sequential multitasking provides more detailed insights into these findings: dedicated workspaces automate actions which normally are required as part of reorganizing the workspace. Not only are these time-consuming, but also error-prone. Although these results provide an initial validation of the activity life cycle as a suitable conceptual model, a long-term study is still needed in order to assess to which degree users' activity histories will be able to inform later activity construction and/or resumption.

10.3 THE MARKS ARE ON THE KNOWLEDGE WORKER

One of the more challenging open questions in activity-centric computing is: "Who should define activities, when, and how?" Activity-centric computing systems typically rely on users to define their own activities, but "experience has shown that people are very poor at remembering to update system representations of their own state" [23]. Since work on each activity takes place within a dedicated workspace (showing only those resources related to one activity), commencing work on new activities without setting up a separate workspace necessarily means resources become intertwined. Furthermore, users do not always know beforehand whether a particular sub-task warrants the creation of a new activity, given the perceived overhead in doing so. *Interruptions* (often associated with a change in work context [67]) are even more likely to inadvertently incorporate unrelated resources into the ongoing activity context. These are issues of *intelligibility*: changes in *intent* might occur unwittingly, resulting in users starting work on unrelated resources within the context of previously defined activities.

Problems of intelligibility lead to unclear borders in between activities.

To this end, some context-aware systems automate the creation of activities based on an analysis of the users' actions. In contrast to activity-centric computing, supporting the creation of workspaces which can be labeled with a preexisting intent, context-aware systems automatically derive intent from interactions with the system and associate context to it. In other words: where users in activity-centric computing assign *intent to context*, context-aware systems assign *context to intent* (Figure 59). However, issues of intelligibility can still arise in the form of incorrect presumption of human intention, which needs to be accounted for to prevent severe annoyances [23]:

Intelligibility is also an issue in context-aware computing.

Intelligibility: Context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it.

The concerns with a lack of intelligibility, therefore, albeit arising from different reasons, seems similar in both approaches to computing: *a discrepancy between context and intent occurs during interaction with the system.*

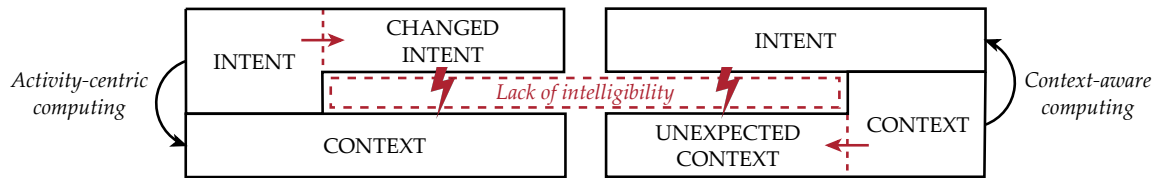


Figure 59: A lack of intelligibility can lead to a discrepancy between context and intent.

Activities should be defined by the user, but can be enhanced using context-aware computing.

In contrast to most context-aware computing systems, the conceptual model adopted in Laevo follows Kidd’s recommendation; we should not try to predict what users want to do with represented information since “the marks are on the knowledge worker”. Rather, we should provide users with “structures which are both flexible in their semantics and generative in nature” [77]. Laevo provides users with the necessary structures to allow externalizing ongoing and planned work, storing the marks made on them for future reference. However, it is also evident that activity-centric computing should avoid the need for users to spend too much time on defining new activities. Therefore, a hybrid approach was suggested of explicitly defined activities, enhanced by AI to predict and suggest activity operations [20]. This can reduce activity construction costs while simultaneously supporting episodic memory by making the construction of activities more explicit [138].

Laevo resolves intelligibility by making users more aware about their own activities.

To prevent issues of intelligibility, the presentation of activities in Laevo is aligned with the mental model users have of them—the activity life cycle. Laevo reinforces reasoning about the state of activities by making state modifications an integral part of setting up their work environment. Users need to open and close activities in order to create a multitasking session which provides a representative overview of the activities they are working on. In addition, interruptions arising from within supported applications (e. g., emails) are automatically introduced as new activities (to-do items), making merging their context with preexisting activities a conscious choice, as opposed to default behavior. Similarly, the ‘home’ activity was adopted by users as a staging area for uncertain activities, preventing their context from intertwining with unintended activities. In case application windows do end up in the wrong activity, cut and paste operations allow to move windows around easily. Lastly, the evaluation of Laevo showed that the explicit full screen interface made users more *aware* about their ongoing and planned activities, empowering them to make more conscious decisions on how to structure and organize their work. Therefore, I posit that the activity life cycle improves intelligibility.

co-Laevo introduces activity awareness within cooperating teams.

To support cooperation, co-Laevo associates each activity with an independent *time line*, which can be shared with other users. The *local dedicated workspace*, introduced as part of Laevo, remains unchanged. Within the local workspace users can set up their own private work

environment, yet identical to an ordinary desktop environment, users can also access shared resources and initiate collaboration using traditional collaboration tools. From a shared activity time line, the sub-activities of the parent activity can be coordinated, planned, and distributed. Similar to Laevo, users are encouraged to update the state of their activities, thereby making team members more aware about each other's activities. This type of 'proactive' awareness is often left unsupported in collaborative computing systems [34]:

Too often the fact that actors actively monitor and proactively display awareness information is disregarded in favor of undifferentiated mechanisms of notification. In doing so, the fact that the proactive part of the phenomenon remains unsupported, especially in asynchronous remote collaboration, is weighted against the fact that the resulting technology might seem easier for the user to appropriate and surely simpler for the designer to construct.

However, updating the state of activities is part of every day knowledge work. Since users need to access the hierarchy view to create or search for new activities to work on, they are regularly confronted with the ongoing, past, and future work of all participants, thereby not only supporting proactive awareness, but also improving *passive* awareness within cooperating teams.

In designing an integrated computational environment the most basic heuristic demand is to try to generate a small set of structures out of which all necessary functionality can be built.

— *Andrea A. diSessa [139]*

This dissertation contributes to research within *activity-centric computing*, an alternative computing paradigm which aims to provide direct support for human activities, rather than through intermediate abstractions such as files and applications. This is achieved by structuring work within computational representations of human activity, with a particular focus on providing better support for users during *sequential multitasking*. The contributions in this dissertation address two open issues which have been formulated in prior work in this line of research [20]:

This dissertation addresses two open issues in activity-centric computing.

ACTIVITY LIFE CYCLE As users work on their activities, the context of different activities easily intertwine with one another. There are no clear demarcations between the start of one activity and the end of another. During the discussion, I referred to these as issues of *intelligibility*: accidental changes in *intent* of the user due to unsupported or unclear borders in between separate activities.

ORGANIZING AND MANAGING ACTIVITIES A second issue concerns *scalability*. It is unclear how current activity-centric solutions would scale when used both over longer periods of time and by larger groups of users. The design of current systems does not take into account the large number of activities which would be amassed in such scenarios.

Encapsulating these two open issues, I have formulated one main and two underlying research questions which are the focus of this dissertation:

How can support for the full life cycle of activities, from creation to completion, be incorporated into an activity-centric computing system?

- R1 *What constitutes the activity life cycle? How do users manage their activities in contemporary computing systems, and what influences the creation and lifetime of an activity?*
- R2 *How can support for long-term activity management be included in an activity-centric computing system?*

Part i outlined a clear goal for activity-centric computing.

To explore the design space of activity-centric computing and to investigate how the current underlying principles are related to a larger research agenda, I have introduced a theoretical framework for human-computer interaction—the *interaction framework* (Chapter 3). This framework represents two *motor themes* which were identified as recurring topics in HCI and are related to a broad range of disparate technologies (Chapter 2): the (1) nature of *human activity* and (2) the *interface*. As such, it can be used to frame *common concepts*, enabling collaboration among differing lines of research with opposing points of view. By systematically outlining the activity-centric computing principles within the interaction framework, the *integrative* nature of this line of research became apparent (addressing *information fragmentation*) and clear gaps in current technological support for the full spectrum of human-computer interaction were identified (Table 22). Most notably, *task management* and *workflow management* are not supported, which I collectively refer to as *activity management* (Chapter 4).

	LAEVO		CO-LAEVO			
ACTIVITY	1. Activity-centered		4. Adapt.	5. Sharing		
WORKSPACE	2. Multiplexing		6. Cont.-aw.	3. Roaming		
ITEM						
MATERIAL						

Table 22: The current activity-centric computing principles outlined within the interaction framework. This highlights how Laevo and co-Laevo target previously unaddressed issues of information fragmentation.

Part ii introduces technologies based on a conceptual model for activity management—the activity life cycle.

In order to investigate what needs to be supported by activity management, i. e., what constitutes the activity life cycle (R1), I performed a literature review of common tools used by knowledge workers as part of their daily work practices: *task*, *window*, and *file management*. This highlighted commonalities and conflicts between individual work practices, which need to be accounted for as part of *integrated knowledge work*. A review of prior activity-centric computing systems confirms that such an integration is currently missing, and that the focus of prior systems lies on constructing the *activity context*, as opposed to the *management of activities* these contexts represent (Chapter 5). This literature review gave rise to the *activity life cycle*—a conceptual model for activity management, encapsulating both how users reason about their activities and how computational support can be provided for them. This model describes the constant and interleaved construction, interruption, resumption, and closure of activities. Through the development of two interactive computing systems (Laevo and co-Laevo, Chapter 6), I demonstrated how this model can be incorporated into the design of activity-centric comput-

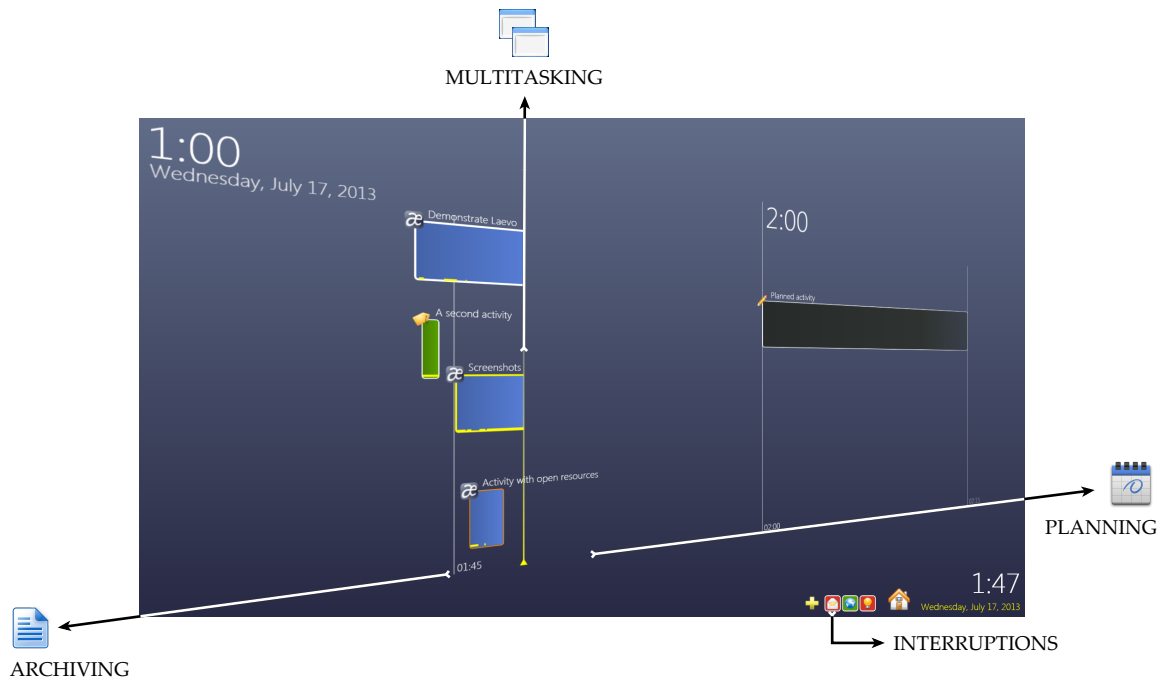


Figure 60: The activity life cycle is supported in Laevo through the management of activities on a time line and by constructing work within the context of dedicated workspaces.

ing systems (Figure 60), thereby providing support for activity management (R2). However, in order to enable the creation of these two systems, and to ease further development in this line of research, a dedicated workspaces (DW) toolkit was created, simplifying the complex work of setting up activity workspaces (Chapter 7).

Laevo was evaluated during a two-week field study by six knowledge workers with varying professional backgrounds (Chapter 8). Results indicate that participants were more *focused* and *aware* about their ongoing activities, and increasingly relied on creating activities, even for short-term tasks. This indicates that the conceptual model and features supported in Laevo encourage users to keep activities separated, thereby reducing issues which can otherwise arise due to a lack of *intelligibility*. In addition, from the evaluation of Laevo new features for co-Laevo were derived which support the management of even larger numbers of activities: *activity hierarchies*, and *activity instances in time*. This demonstrates how the activity life cycle addresses issues of *scalability*. A more detailed understanding of this conceptual model (R1) was obtained by conducting two empirical studies on *sequential multitasking* (Chapter 9). Results from these two studies clearly indicate the benefits of structuring work within the context of activities, compared to a traditional desktop environment: dedicated workspaces (1) automate *error-prone* task resumption, (2) *reduce cognitive load*, and (3) are *an order of magnitude faster*. These results provide much-needed quantitative data in support of activity-centric computing, and bring forward a *framework for analysis* for future work.

Part iii introduces empirical studies highlighting how the activity life cycle addresses issues of scalability and intelligibility.

Activity-centric computing has only achieved incremental innovation.

Several times throughout this dissertation I have highlighted how much the design of interactive computing systems relies on, and is influenced by underlying architecture [42]. Therefore, design and computer science (as practiced in HCI) are—and are likely to remain—extremely interconnected. This is a *fundamental limitation* for an activity-centric approach to system design, which envisions a radically new computing paradigm incompatible with the current antiquated desktop metaphor for office work. Building activity-centric computing systems is a continuous uphill battle against an industry which is only trying to achieve local maxima. In contrast, activity-centric computing pursues *radical innovation*, aiming for one unifying global maximum. To conclude this dissertation, I will reflect on how current efforts in this line of research have only accomplished *incremental innovation*, and will base a vision for future work on an analysis of early interactive computing systems (less restricted by prior infrastructure) through the use of the interaction framework.

11.1 LIMITATIONS: RADICAL INNOVATION

The main limitation for activity-centric computing is present-day technology.

While introducing the interaction framework in Chapter 3, and reviewing PIM tools in support of managing tasks, windows, and files in Section 5.1, I highlighted the fragmented nature of technology. From a historical point of view this can be explained by considering that the computer is ‘reaching out’ [51], thus new technologies are constantly being built on top of prior ones. Referring to terminology introduced as part of the interaction framework: new technologies are composed of basic building blocks made available from lower levels of abstraction. This enables rapid development of new technologies, but unfortunately also builds up technical debt: “older problems are never totally solved but remain beside the newer, larger ones; only their relative importance diminishes gradually” [80]. More critically, the design of interactive computing systems is not only influenced by underlying infrastructure, but is also largely inhibited by it [42]. Given that the goal of activity-centric computing is to conceptually overhaul the way we interact with present-day technologies, the main limitation in this line of research for the successful deployment of new computing systems is the technological stack which we inescapably inherit from the past [101]:

The main barrier to the introduction of technology that is aligned with people’s real needs and desires, with people’s real activities is the mindset of the computer industry. This industry has grown up being dominated by technology. The result has been the development of powerful tools that have become essential to modern life. ... The problem is that the resulting device is technology-centric.

Therefore, as presented in [Chapter 4](#), current research efforts in activity-centric computing can only achieve *incremental innovation*—patching up issues of interoperability between isolated tools by constructing computational integration around the central notion of human activity. The underlying vision, however, can only be realized by making breaking changes to the current infrastructure. No *intermediate* architecture will ever be able to fully support the concept of activity-centric computing¹, which is why the [DW](#) toolkit was first and foremost designed to inspire the foundation of, and support *transitioning* to, an altogether new infrastructure. In practice (as presented in [Chapter 7](#)), the [DW](#) toolkit is still unable to work around several *interjected abstractions* [42]: e. g., the need to suspend workspaces in order to free up computational resources, and the need for at least one workspace to be visible at any given point in time. The current application model (and therefore all applications) need to be restructured drastically in order to work around such issues. Only then can the unintuitive technological abstractions which still persist in computing systems to date be fully hidden from view.

Norman and Verganti compare incremental innovation with *radical innovation* through the use of a hill-climbing analogy: “Incremental innovation attempts to reach the highest point on the current hill. Radical innovation seeks the highest hill” ([Figure 61](#)) [105]:

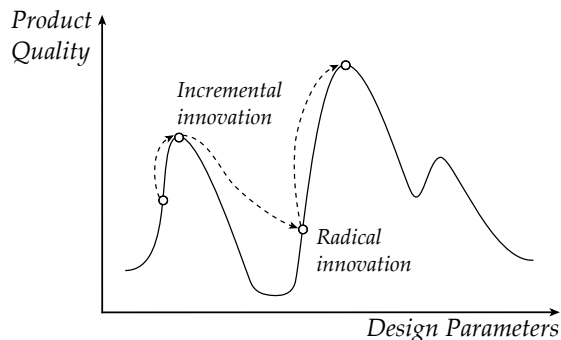


Figure 61: Incremental innovation compared to radical innovation. Figure adapted from original by Norman and Verganti [105].

This poses an interesting question for [HCI](#): How can we become aware of surrounding hills with more promising vantage points, and how do we weigh the cost of ascending them against the increase in product quality there is to be gained? In [Section 2.5](#) I argued [HCI](#) is lacking the necessary motor themes (representing accumulated knowledge) in order to make such qualified decisions. Often topics are no

True activity-centric computing cannot be built on top of current infrastructure.

HCI favors short-term novelty over sustained radical innovation.

¹ Even Microsoft struggles with supporting dedicated workspaces in Windows 10. The newly introduced ‘task view’ feature exhibits the same underlying problems encountered in previous [VDMs](#): *single-instance applications* lead to application windows ending up in unexpected workspaces. The task view in Windows 10 ‘resolves’ this by automatically switching to a different workspace whenever an application gets focus.

longer pursued simply because they fall out of fashion, reflecting short bursts of radical innovation only for as long as contributions are deemed noteworthy by the community. There seems to be a general disregard for investigating whether a hill is worth ascending in the first place.

Radical innovation requires pursuing a consistent vision.

However, as per Steve Job’s ‘parable of the stones’ (Appendix A), there is a great amount of work to be done in between a great idea and the final product. Unfortunately, “[m]ost radical innovations fail. Those that succeed can take decades before they are successful” [103]. For example, as mentioned earlier, although the Xerox Star [68] defined the current desktop metaphor (which was only adopted several years later), it did not become a commercial success. It envisioned radical innovation through a change in the way people give meaning to personal computers: before the Xerox Star, personal computers were solely targeted at computer enthusiasts; in contrast, the Star was designed for users without a technical background, to be used within an office environment. Bringing about radical innovation requires defining a clear goal—a vision for what technology can be—and consistently persevering in the pursuit to make it happen.

11.2 FUTURE WORK: A LONG-TERM GOAL

A long-term goal for activity-centric computing should not rely on current infrastructure.

As a theoretical contribution I have brought forward the *interaction framework* (Chapter 3). Within this framework the current goal for activity-centric computing was systematically delineated and subsequently extended upon (Chapter 4). Activity-centric computing primarily attempts to integrate different levels of abstraction (inherited from a technology-oriented past), using the central notion of human activity. However, imagining a fundamentally different infrastructure, a *long-term goal* for activity-centric computing can be formulated. Such a goal pursues *radical innovation*, unbounded by (and incompatible with) contemporary technologies, but capable of addressing persisting underlying issues of interactive computing systems.

Looking into early literature helps understanding the design of current technologies.

Unlike a *bottom-up* approach to system design, a *top-down* approach tries not to be constrained by prior technology. However, it does need to rely on a basic understanding of underlying implementation details in order to discriminate between purely technological (or arbitrary²) system constraints, and those that have been put in place for very specific reasons. This is essential in deciding whether such constraints should be eliminated rather than adhered to. For this very reason, it is extremely valuable to revisit early literature in computer science preceding the mainstream adoption of personal computers. This can give insights into long-forgotten design choices regarding the fragmented nature of current technology.

² For example, it is frustratingly common for passwords to have a maximum length, even though this weakens password strength considerably.

The first graphical window management system was introduced as part of the Smalltalk programming environment in the 1970s [129]. The degree to which the design of graphical user interfaces is intertwined with the development of object-oriented programming (OOP) cannot be overstated. OOP is more than just a vision for software engineering—it is a vision for computing; I highly doubt that it is a coincidence that windows were conceived as ‘objects’ in this computing paradigm. After hearing Bob Barton state that “[t]he basic principal [sic] of recursive design is to make the parts have the same power as the whole”, Alan Kay’s vision for OOP materialized as follows [129]:

Early window management was based on recursive system design.

For the first time I thought of the whole as the entire computer and wondered why anyone would want to divide it up into weaker things called data structures and procedures. Why not divide it up into little computers, as time sharing was starting to? But not in dozens. Why not thousands of them, each simulating a useful structure?

He later described his vision for computing as [75]:

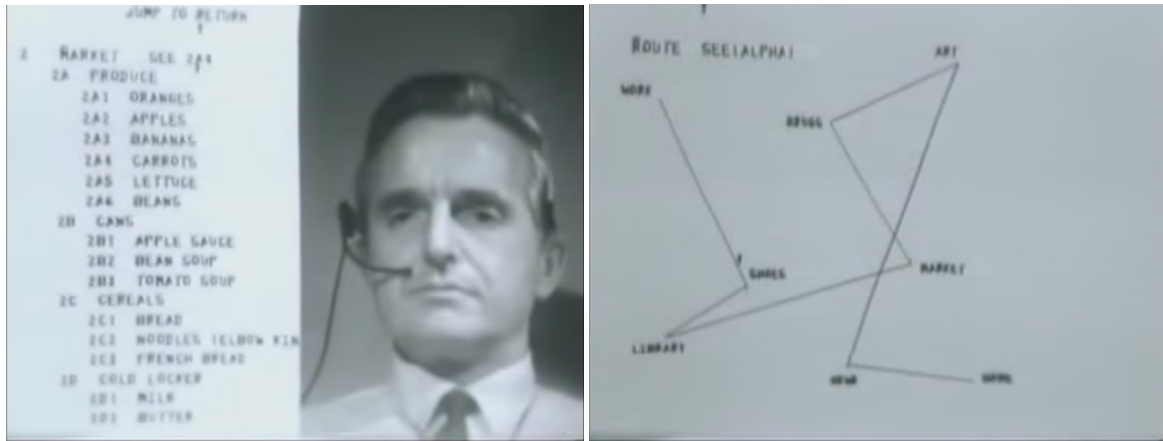
In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing “computer stuff” into things each less strong than the whole—like data structures, procedures, and functions which are the usual paraphernalia of programming languages—each Smalltalk object is a recursion on the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network.

This *recursive approach* to the design of interactive computing systems can be recognized in the design of many early computing systems, reaching all the way back to NLS [43] in 1968. As a “research center for augmenting human intellect”, NLS was much more than a mere technological contribution; it was a vision for what computing can be. Through the recursive design of interconnected nodes, NLS supported anything from programming, to paper writing, performing a presentation at a conference, to mundane human activities like setting up a grocery shopping list and an associated itinerary (Figure 62). By querying for nodes and changing the view properties of retrieved nodes, dedicated workspaces could effectively be constructed by the user. Thus, recursive design is an extremely powerful tool for the design of interactive computing systems. It provides users with the necessary flexibility to tailor a computing system to their needs.

Recursive design offers flexibility to the user.

In contrast, Rooms [55], and modern systems supporting dedicated workspaces, are fundamentally different from early experimental systems such as NLS and Boxer [139]. The latter category embodies a different computing paradigm than the one that we have grown

Recursive design is not tied to fixed layers of abstraction.



(a) Grocery shopping.

(b) Itinerary for groceries.

Figure 62: The design of NLS [43] supported dedicated workspaces, e. g., for grocery shopping.

accustomed to, devoid of files and applications. Rather than a feature patched on in order to be able to aggregate different resources, workspaces are an emergent property of the way information is stored and presented; there is no functional difference between a workspace and a resource as known in traditional computing. Considering the interaction framework, these systems do not provide separate computational support for each of the levels of abstraction (Table 23). Instead, each computational unit is as versatile as the next. It is through composition by the user, that items, workspaces, and activities can be set up.

ACTIVITY							
WORKSPACE							
ITEM							
MATERIAL							

Table 23: Historically, computational support for different levels of abstraction has not always been separated.

Possibly these recursive approaches (resembling end-user programming) were abandoned as part of the introduction of personal computers, in favor of less complex, but more strict, software architectures. However, given the amount of problems which keep arising due to a lack of integration between documents, applications, and more recently human activity, one might wonder at what cost. With the added capabilities of modern day devices, it would be worthwhile investigating the design of new ‘recursive’ interactive computing systems, where the basic computational unit is no longer a simple piece of data, but a connected, shared, atomic representation of ‘activity’.

Part V

APPENDIX



THE PARABLE OF THE STONES

INTERVIEW

Robert X. Cringely interviews *Steve Jobs* in “*Steve Jobs: The Lost Interview (2012)*”, footage from 1995.

TRANSCRIPT

Robert X. Cringely: “Now, you motivated this team, you had to guide them—”

Steve Jobs: “We had to build the team!”

Robert X. Cringely: “Build the team, motivate it, guide them, deal with them. We’ve interviewed just lots and lots of people from the Macintosh team. And, what it keeps coming down to is your passion, your vision. How do you order your priorities in there? What is important to you in the development of the product?”

Steve Jobs: “You know, one of the things that really hurt Apple was after I left, John Sculley got a very serious disease. And that disease, I’ve seen other people get it too, it’s the disease of thinking that a really great idea is 90% of the work. And that if you just tell all these other people ‘here’s this great idea,’ then of course they can go off and make it happen. And the problem with that is that there’s just a tremendous amount of craftsmanship in between a great idea and a great product. And as you evolve that great idea, it changes and grows. It never comes out like it starts because you learn a lot more as you get into the subtleties of it. And you also find there are tremendous trade-offs that you have to make. There are just certain things you can’t make electrons do. There are certain things you can’t make plastic do. Or glass do. Or factories do. Or robots do. And as you get into all these things, designing a product is keeping five thousand things in your brain, these concepts, and fitting them all together in, kind of continuing to push to fit them together in new and different ways to get what you want. And every day you discover something new that is a new problem or a new opportunity to fit these things together a little differently. And it’s that process that is the magic. And so we had a lot of great ideas when we started. But what I’ve always felt that a team of people doing something they really believe in is like, is like when I was a young kid there was a widowed man that lived up the street. He was in his eighties. He was a little scary looking. And I got to know him a little bit. I think he may have paid me to mow his lawn or something. And one day he said to me, ‘come on into my garage I want to show you something.’ And he pulled out

this dusty old rock tumbler. It was a motor and a coffee can and a little band between them. And he said, 'come on with me.' We went out into the back and we got just some rocks. Some regular old ugly rocks. And we put them in the can with a little bit of liquid and little bit of grit powder, and we closed the can up and he turned this motor on and he said, 'come back tomorrow.' And this can was making a racket as the stones went around. And I came back the next day, and we opened the can. And we took out these amazingly beautiful polished rocks. The same common stones that had gone in, through rubbing against each other like this, creating a little bit of friction, creating a little bit of noise, had come out these beautiful polished rocks. That's always been in my mind my metaphor for a team working really hard on something they're passionate about. It's that through the team, through that group of incredibly talented people bumping up against each other, having arguments, having fights sometimes, making some noise, and working together they polish each other and they polish the ideas, and what comes out are these really beautiful stones."

LAEVO EVALUATION MATERIAL

B.1 LAEVO MANUAL

AUTHOR

Steven Jeuris

DATE

August 2013

Laevo

[Download the latest version: Alpha v0.1.4](#)

Laevo is a personal information management tool for Windows 7 and 8 which offers you additional functionality by which to organize your work, with a particular focus on supporting **multitasking**. Our goal is to do this without changing the everyday interaction with Windows drastically. In case you have ever used a **virtual desktop manager** the general concept will seem quite similar, but in essence we are striving for much more, and several additional features are already available.



What is a virtual desktop manager? When using a virtual desktop manager, you have not one, but several desktops available to you between which you can switch easily. An over simplified analogy would be having several PCs available to you, each running their own programs and being able to switch between them without having to change seats.

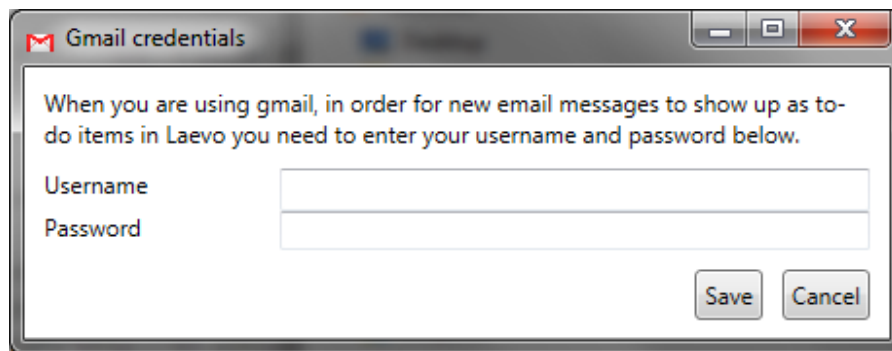
Overview

The two core concepts which Laevo supports are **activities** and **to-do** items, comparable to many existing calendar and planning tools. Laevo effectively tries to integrate your calendar into your desktop environment.

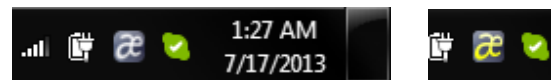
Activities: these can be seen as the **entire context** of a certain task you are working on, worked on before, or are planning to work on at a predetermined time. E.g. when writing a report you might be working in a text editor, but you also might need relevant emails of your supervisor and additional information from certain web pages. For each activity, Laevo offers you the opportunity of creating a separate virtual desktop, giving you a clean slate to work on.

To-do items: these are very similar to activities. The only difference is they represent work that still needs to be done, without having specified when. Like activities however, they can still hold their own context on a separate virtual desktop.

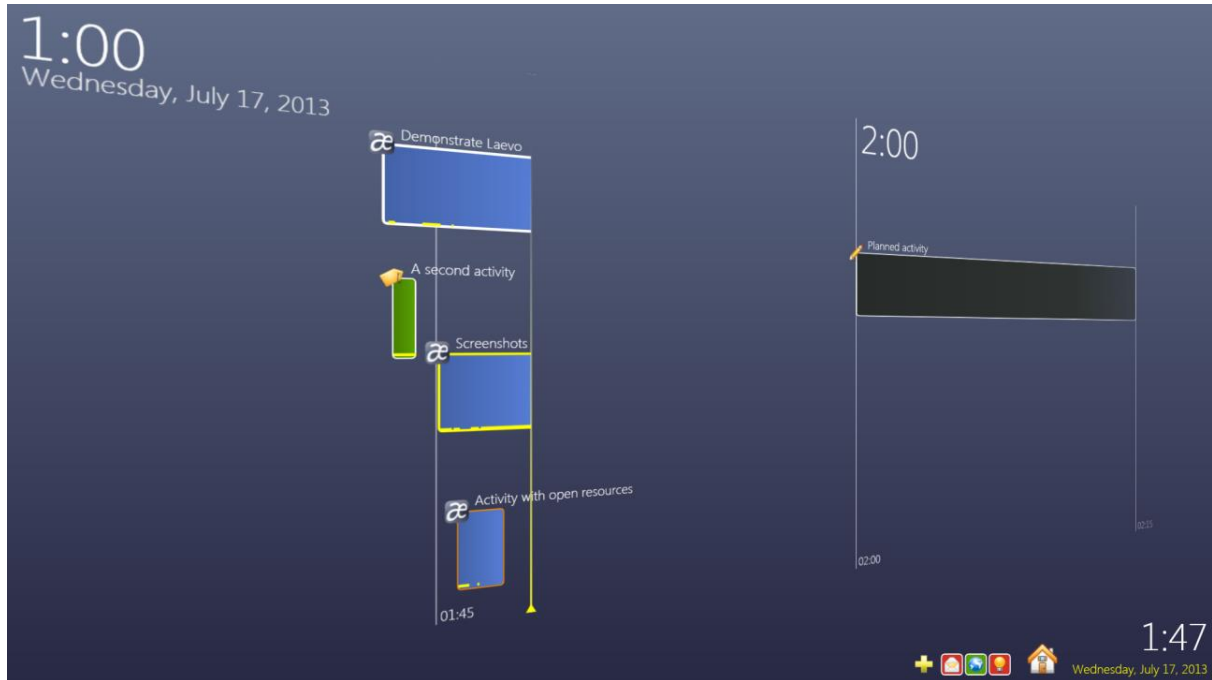
When first running Laevo, you have the option of entering your [Gmail](#) login and password. In case you don't have Gmail, just click *Cancel*. Laevo uses this information to retrieve unread emails from your inbox and adding them as to-do items. In case you entered a wrong username and password, the dialog box will pop up again. (Don't forget to create a new password in case you are using [2 step verification](#)!)



After Laevo has started, you won't see much of a difference. As stated before, our goal is to intrude as little as possible in your daily work. You can find Laevo as a tray icon in the bottom right. It might be hidden, in which case you will need to click the little triangle in order to find it. We advise you to go to *"Customize"* by clicking the triangle and setting Laevo to *"Show icon and notifications"* so that it is always visible. The Laevo icon will light up when you have received a new email in case you entered your Gmail credentials.



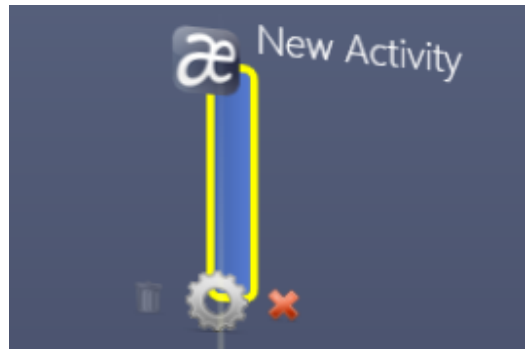
Double clicking the icon will open up a full screen overview of your activities and to-do items, presented on a time line. Alternatively (and recommended) you can press *Caps Lock* (named *Shift Lock* on some keyboard layouts) to show or hide the overview. The key's traditional behavior has been overridden, and all short keys of Laevo start out with *Caps Lock*. You can still enable/disable Caps Lock by pressing *Caps Lock - A*. Since you haven't created any activities or to-do items yet, the time line will be empty, but as will be shown shortly the time line might be populated as follows.



Time is presented on the horizontal axis, and activities as work occurs on them (possibly in parallel) are organized on the vertical axis. Try click-dragging to move the time line around, and scrolling to zoom in/out. The yellow vertical marker indicates the current time. Activities which are currently being worked on continue expanding, getting wider and wider, as long as they aren't closed. In the image above, there are two activities currently open, and two closed ones which were open in the past. Additionally, activities can be planned in the future at a specified time, as shown by the activity with the black background behind the yellow marker. At the bottom you can find a list of to-do items, represented by icons. Next to it you can find a bigger "Home" icon, representing the desktop you were working on prior to starting Laevo. Clicking it brings you back to your original desktop. Clicking an activity or to-do item opens up a clean desktop in which you can work, exactly the same as the original Windows desktop, but only showing windows for that activity or to-do item.

Your first activity

Press *Caps Lock - N*; a clean desktop will show up. You have just created your first activity, and immediately have started working on it. For now it is empty. If you go back to the overview you will see a "New activity" on your time line with a yellow border, indicating this is the activity which is currently open. Click dragging the activity allows you to choose where to place it vertically. Hovering over it with your mouse pops up a menu with 3 icons. A garbage bin, a gear, and a close button.



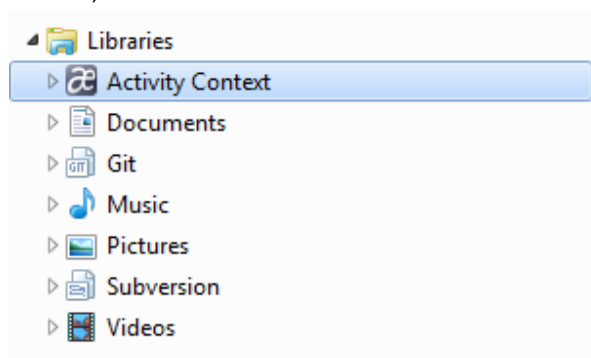
Garbage bin: removes an activity from the time line. You can only remove an activity when it is no longer open (you aren't working on it) and there are no windows open in it. A closed activity which has windows open inside of it has an orange border.

Gear: opens up a menu in which you can give the activity a name and icon. You can also adjust the name of the activity directly by clicking the label on the time line.

Close: indicates on the time line that you are no longer working on the activity. However, you can still access it as you would otherwise just by clicking on it. It is up to you to decide when you keep an activity closed or open.

When you are working on the desktop of your activity, all the windows you open up in there will become part of that desktop. Try switching back and forth between the *Home* activity and your newly created one after opening up a couple of windows. (Hint: *Caps Lock - Tab* allows you to switch between the last two accessed activities without having to go through the overview.)

For each activity, there is a directory which is automatically created in which you can place relevant files. When inside an activity (on the desktop) this directory can always be accessed from Windows Explorer under the "Libraries" list, named "Activity Context". Alternatively, you can open up this library when inside an activity



using *Caps Lock - L*. You can even add additional folder locations to the library to import existing files. For more information on Windows Libraries, check out [the official documentation](#).

Moving windows between activities

You can **cut and paste windows**, just like you cut and paste files, in order to move them from one desktop to another. Highlight a window (click on it) and press *Caps Lock - X*. The selected window will disappear. Press *Caps Lock - V* to show it again. You can cut multiple windows in a row; when pasting, all previously cut windows will show up again.

By cutting windows when inside one activity, and pasting them when inside another, you can **move windows in between activities**.

Planning, to-do items and interruptions

Clicking the "+" icon on the overview adds a new to-do item. As you will notice they behave very similar to activities, you can also click on them to open up a new desktop to work on. New to-do items show up on the left hand side. You can drag the icons left and right to organize them. Once you decide to start working on a to-do item you can drag it to the time line, effectively turning it into an activity.

Start working: when you drag the icon *in front of* the current time marker and drop it you will convert the to-do item into an open activity, indicating you start working on it.

Plan: you can drag it *behind* the current time marker, positioning it where you are planning to work on it. Through the edit menu you can change the date and time more accurately in case needed. You can start working on a planned item by clicking the yellow arrow in the mouse-over menu. To plan to-do items you can also open up the context menu on the time line (right mouse button) *behind the current time marker* at the location where you want to place the activity, and clicking "*Plan activity*".



Assign: drag a to-do item to an existing activity (or the home icon) in order to make it part of that activity. All windows open in the to-do item will be merged with those from the activity.

In case you provided login information for Gmail, new emails will show up as new to-do items with an animated yellow border, until they have been opened. Opening an email "*interruption*" will besides opening the desktop also open up the context of the interruption; in case of Gmail, the email will open up in a new browser window.

Shortkeys overview

Caps Lock: show/hide overview

Caps Lock - N: open new activity

Caps Lock - Tab: switch between last two open activities

Caps Lock - L: open activity context library

Caps Lock - W: close activity

Caps Lock - X: cut window

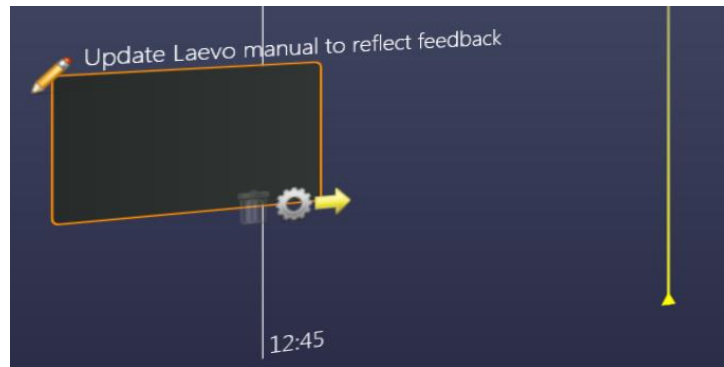
Caps Lock - V: paste window

Caps Lock - A: enable/disable caps lock

Exiting and restarting Laevo

When you want to shut down Laevo, right click the tray icon, and select "*Exit*". All open windows from all your activities and to-do items will show up again. All activities are closed when shutting down Laevo, and will be represented as such when Laevo is restarted.

When restarting Laevo, the windows which were previously part of an activity or task context will be reassigned to those and thus be hidden upon startup. All new windows, not previously part of any activity or task, become part of the "*Home*" activity. Since all activities close when exiting Laevo, the activities which were open at the time Laevo was shut down won't stretch up to the current time after a restart. You might need to zoom out and/or move the time line to the past to find them. By clicking the yellow arrow you can make the activity stretch to the current time again, indicating that work on it is continued.



Troubleshooting

In case the time line overview is running a bit slow, try lowering the quality in the settings which can be accessed through the context menu (right click) of the tray icon.

When the application crashes there should be a *"log.txt"* file available in *"C:\Users\<username>\Documents\Laevo"*. Please email this to me. (*sjeu AT itu.dk*)

B.2 LAEVO DIARY STUDY QUESTIONS

The following questions were filled out every day over the course of two weeks:

1. Why was or wasn't Laevo useful for you today? At a minimum state one positive and one negative points, but open feedback is encouraged.
2. What activities have you done today that weren't represented in Laevo at some point?
3. What was/were your main activities today? In case they were represented in Laevo, where did they originate from (self-initiated, to-do item, email to-do, other)?
4. Have you scheduled any activities today? Did you also plan them on the time line? Why (not)?
5. Did you use Laevo's to-do list today? Why (not)? How?
6. Did you use Laevo's Activity Context library today to store or retrieve files?
7. Were there occasions where you considered creating an activity or to-do item but eventually decided not to? If so, why?
8. Please have a look at your time line. Does the overview of today reflect the actual activities you did today? Why (not)?

BIBLIOGRAPHY

- [1] Eytan Adar, David Karger, and Lynn Andrea Stein. "Haystack: Per-user Information Environments." In: *Proceedings of the Eighth International Conference on Information and Knowledge Management*. CIKM '99. Kansas City, Missouri, USA: ACM, 1999, pp. 413–422. ISBN: 1-58113-146-1. DOI: [10.1145/319950.323231](https://doi.org/10.1145/319950.323231). URL: <http://doi.acm.org/10.1145/319950.323231>.
- [2] Rachel F. Adler and Raquel Benbunan-Fich. "Juggling on a High Wire: Multitasking Effects on Performance." In: *International Journal of Human-Computer Studies* 70.2 (Feb. 2012), pp. 156–168. ISSN: 1071-5819. DOI: [10.1016/j.ijhcs.2011.10.003](https://doi.org/10.1016/j.ijhcs.2011.10.003). URL: <http://dx.doi.org/10.1016/j.ijhcs.2011.10.003>.
- [3] Rachel F. Adler and Raquel Benbunan-Fich. "Self-interruptions in discretionary multitasking." In: *Computers in Human Behavior* 29.4 (2013), pp. 1441–1449. ISSN: 0747-5632. DOI: [10.1016/j.chb.2013.01.040](https://doi.org/10.1016/j.chb.2013.01.040). URL: <http://www.sciencedirect.com/science/article/pii/S0747563213000435>.
- [4] Rachel F Adler and Raquel Benbunan-Fich. "The effects of task difficulty and multitasking on performance." In: *Interacting with Computers* 27.4 (2015), pp. 430–439.
- [5] Anand Agarawala and Ravin Balakrishnan. "Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréa, Québec, Canada: ACM, 2006, pp. 1283–1292. ISBN: 1-59593-372-7. DOI: [10.1145/1124772.1124965](https://doi.org/10.1145/1124772.1124965). URL: <http://doi.acm.org/10.1145/1124772.1124965>.
- [6] Erik M. Altmann and J. Gregory Trafton. "Task Interruption: Resumption Lag and the Role of Cues." In: *Proceedings of the 26th Annual Conference of the Cognitive Science Society*. CogSci '04. Chicago, Illinois, USA: Lawrence Erlbaum Associates, Inc., 2004, pp. 43–48. ISBN: 0-8058-5464-9.
- [7] Brian P Bailey and Joseph A Konstan. "On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state." In: *Computers in human behavior* 22.4 (2006), pp. 685–708.
- [8] Brian P Bailey, Joseph A Konstan, and John V Carlis. "The effects of interruptions on task performance, annoyance, and anxiety in the user interface." In: *Proceedings of INTERACT*. Vol. 1. 2001, pp. 593–601.

- [9] Liam J. Bannon. "From human factors to human actors: The role of psychology and human-computer interaction studies in system design." In: *Design at work: Cooperative design of computer systems* (1991), pp. 25–44.
- [10] Liam J. Bannon. "Perspectives on CSCW: From HCI and CMC to CSCW." In: *Proceedings of the International Conference on Human-Computer Interaction*. EW-HCI '92 (1992), pp. 148–158.
- [11] Liam Bannon, Allen Cypher, Steven Greenspan, and Melissa L. Monty. "Evaluation and Analysis of Users' Activity Organization." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '83. Boston, Massachusetts, USA: ACM, 1983, pp. 54–57. ISBN: 0-89791-121-0. DOI: [10.1145/800045.801580](https://doi.org/10.1145/800045.801580). URL: <http://doi.acm.org/10.1145/800045.801580>.
- [12] Jakob E. Bardram. "Plans as situated action: an activity theory approach to workflow systems." In: *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work*. Springer. 1997, pp. 17–32.
- [13] Jakob E. Bardram. "Collaboration, Coordination and Computer Support: An Activity Theoretical Approach to the Design of Computer Supported Cooperative Work. Ph. D. Thesis." In: *DAIMI Report Series 27.533* (1998).
- [14] Jakob E. Bardram. "Activity-based computing—lessons learned and open issues." In: *ECSCW 2005 workshop, Activity-From a theoretical to a computational construct*. Citeseer. 2005.
- [15] Jakob E. Bardram. "Activity-based computing: support for mobility and collaboration in ubiquitous computing." In: *Personal and Ubiquitous Computing 9.5* (2005), pp. 312–322.
- [16] Jakob E. Bardram. "Activity-based computing for medical work in hospitals." In: *ACM Transactions on Computer-Human Interaction (TOCHI) 16.2* (2009), p. 10.
- [17] Jakob E. Bardram, Jonathan Bunde-Pedersen, and Mads Soegaard. "Support for activity-based computing in a personal computing operating system." In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM. 2006, pp. 211–220.
- [18] Jakob E. Bardram, Jonathan Bunde-Pedersen, Afsaneh Doryab, and Steffen Sørensen. "CLINICAL SURFACES: Activity-Based Computing for Distributed Multi-Display Environments in Hospitals." In: *INTERACT'09: Proceedings of the Conference on Human Computer Interaction*. Vol. 5727. Lecture Notes in Computer Science. Springer, Aug. 26, 2009, pp. 704–717. ISBN: 978-3-642-03657-6. URL: <http://dblp.uni-trier.de/db/conf/interact/interact2009-2.html#BardramBDS09>.

- [19] Jakob E. Bardram, Sofiane Gueddana, Steven Houben, and Søren Nielsen. “Reticularspaces: Activity-based Computing Support for Physically Distributed and Collaborative Smart Spaces.” In: *CHI'12: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2845–2854.
- [20] Jakob E. Bardram, Steven Jeuris, and Steven Houben. “Activity-based computing: computational management of activities reflecting human intention.” In: *AI Magazine* 36.2 (2015), pp. 63–72.
- [21] P. Barthelmeß and K. M. Anderson. “A View of Software Development Environments Based on Activity Theory.” In: *Comput. Supported Coop. Work* 11.1-2 (Apr. 2002), pp. 13–37. ISSN: 0925-9724. DOI: [10.1023/A:1015299228170](https://doi.org/10.1023/A:1015299228170). URL: <http://dx.doi.org/10.1023/A:1015299228170>.
- [22] Russell Beale and William Edmondson. “Multiple Carets, Multiple Screens and Multi-Tasking: New Behaviours with Multiple Computers.” In: *Proc. BCS HCI*. British Computer Society, 2007, pp. 55–64.
- [23] Victoria Bellotti and Keith Edwards. “Intelligibility and accountability: human considerations in context-aware systems.” In: *Human-Computer Interaction* 16.2-4 (2001), pp. 193–212.
- [24] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. “Taking Email to Task: The Design and Evaluation of a Task Management Centered Email Tool.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 345–352. ISBN: 1-58113-630-7. DOI: [10.1145/642611.642672](https://doi.org/10.1145/642611.642672). URL: <http://doi.acm.org/10.1145/642611.642672>.
- [25] Ofer Bergman, Ruth Beyth-Marom, and Rafi Nachmias. “The Project Fragmentation Problem in Personal Information Management.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 271–274. ISBN: 1-59593-372-7. DOI: [10.1145/1124772.1124813](https://doi.org/10.1145/1124772.1124813). URL: <http://doi.acm.org/10.1145/1124772.1124813>.
- [26] Jasmin Blanchette. “The little manual of API design.” In: *Chair for Logic and Verification* (2008).
- [27] Jeanette Blomberg and Helena Karasti. “Reflections on 25 Years of Ethnography in CSCW.” In: *Computer Supported Cooperative Work (CSCW)* 22.4 (2013), pp. 373–423. ISSN: 1573-7551. DOI: [10.1007/s10606-012-9183-1](https://doi.org/10.1007/s10606-012-9183-1). URL: <http://dx.doi.org/10.1007/s10606-012-9183-1>.

- [28] Richard Boardman and M. Angela Sasse. "'Stuff Goes into the Computer and Doesn'T Come out': A Cross-tool Study of Personal Information Management." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 583–590. ISBN: 1-58113-702-8. DOI: [10.1145/985692.985766](https://doi.org/10.1145/985692.985766). URL: <http://doi.acm.org/10.1145/985692.985766>.
- [29] Susanne Bødker. "When Second Wave HCI Meets Third Wave Challenges." In: *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles*. NordiCHI '06. Oslo, Norway: ACM, 2006, pp. 1–8. ISBN: 1-59593-325-5. DOI: [10.1145/1182475.1182476](https://doi.org/10.1145/1182475.1182476). URL: <http://doi.acm.org/10.1145/1182475.1182476>.
- [30] Deborah A. Boehm-Davis and Roger Remington. "Reducing the disruptive effects of interruption: A cognitive framework for analysing the costs and benefits of intervention strategies." In: *Accident Analysis & Prevention* 41.5 (2009), pp. 1124–1129. ISSN: 0001-4575. DOI: [10.1016/j.aap.2009.06.029](https://doi.org/10.1016/j.aap.2009.06.029). URL: <http://www.sciencedirect.com/science/article/pii/S000145750900164X>.
- [31] Frederick P. Brooks Jr. "No Silver Bullet: Essence and Accidents of Software Engineering." In: *Computer* 20.4 (Apr. 1987), pp. 10–19. ISSN: 0018-9162. DOI: [10.1109/MC.1987.1663532](https://doi.org/10.1109/MC.1987.1663532). URL: <http://dx.doi.org/10.1109/MC.1987.1663532>.
- [32] Duncan P Brumby, Anna L Cox, Jonathan Back, and Sandy JJ Gould. "Recovering from an interruption: Investigating speed-accuracy trade-offs in task resumption behavior." In: *Journal of Experimental Psychology: Applied* 19.2 (2013), p. 95.
- [33] Graham Button and Paul Dourish. "Technomethodology: Paradoxes and Possibilities." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '96. Vancouver, British Columbia, Canada: ACM, 1996, pp. 19–26. ISBN: 0-89791-777-4. DOI: [10.1145/238386.238394](https://doi.org/10.1145/238386.238394). URL: <http://doi.acm.org/10.1145/238386.238394>.
- [34] Federico Cabitza and Carla Simone. "Computational Coordination Mechanisms: A tale of a struggle for flexibility." In: *Computer Supported Cooperative Work (CSCW)* 22.4-6 (2013), pp. 475–529.
- [35] Patrick P. Chan. "Learning considerations in user interface design: The Room model." MA thesis. 1984.
- [36] Henrik Bærbak Christensen and Jakob E Bardram. "Supporting human activities—exploring activity-centered computing." In: *UbiComp 2002: Ubiquitous Computing*. Springer, 2002, pp. 107–116.

- [37] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. "A Diary Study of Task Switching and Interruptions." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 175–182. ISBN: 1-58113-702-8. DOI: [10.1145/985692.985715](https://doi.org/10.1145/985692.985715). URL: <http://doi.acm.org/10.1145/985692.985715>.
- [38] Afsaneh Doryab, Julian Togelius, and Jakob Bardram. "Activity-aware Recommendation for Collaborative Work in Operating Rooms." In: *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*. IUI '12. Lisbon, Portugal: ACM, 2012, pp. 301–304. ISBN: 978-1-4503-1048-2. DOI: [10.1145/2166966.2167023](https://doi.org/10.1145/2166966.2167023). URL: <http://doi.acm.org/10.1145/2166966.2167023>.
- [39] Paul Dourish. "Implications for Design." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 541–550. ISBN: 1-59593-372-7. DOI: [10.1145/1124772.1124855](https://doi.org/10.1145/1124772.1124855). URL: <http://doi.acm.org/10.1145/1124772.1124855>.
- [40] Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L. Herlocker. "Task-Tracer: A Desktop Environment to Support Multi-tasking Knowledge Workers." In: *Proceedings of the 10th International Conference on Intelligent User Interfaces*. IUI '05. San Diego, California, USA: ACM, 2005, pp. 75–82. ISBN: 1-58113-894-6. DOI: [10.1145/1040830.1040855](https://doi.org/10.1145/1040830.1040855). URL: <http://doi.acm.org/10.1145/1040830.1040855>.
- [41] Geoffrey B. Duggan, Hilary Johnson, and Petter Sørli. "Interleaving Tasks to Improve Performance: Users Maximise the Marginal Rate of Return." In: *International Journal of Human-Computer Studies* 71.5 (May 2013), pp. 533–550. ISSN: 1071-5819. DOI: [10.1016/j.ijhcs.2013.01.001](https://doi.org/10.1016/j.ijhcs.2013.01.001). URL: <http://dx.doi.org/10.1016/j.ijhcs.2013.01.001>.
- [42] W. Keith Edwards, Mark W. Newman, and Erika Shehan Poole. "The Infrastructure Problem in HCI." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 423–432. ISBN: 978-1-60558-929-9. DOI: [10.1145/1753326.1753390](https://doi.org/10.1145/1753326.1753390). URL: <http://doi.acm.org/10.1145/1753326.1753390>.
- [43] Douglas C. Engelbart and William K. English. "A Research Center for Augmenting Human Intellect." In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: ACM, 1968, pp. 395–410. DOI: [10.1145/1476589.1476645](https://doi.org/10.1145/1476589.1476645). URL: <http://doi.acm.org/10.1145/1476589.1476645>.
- [44] Yrjö Engeström, Reijo Miettinen, and Raija-Leena Punamäki. *Perspectives on activity theory*. Cambridge University Press, 1999.

- [45] Adam Fouse, Nadir Weibel, Edwin Hutchins, and James D. Hollan. "ChronoViz: A System for Supporting Navigation of Time-coded Data." In: *CHI '11 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '11. Vancouver, BC, Canada: ACM, 2011, pp. 299–304. ISBN: 978-1-4503-0268-5. DOI: [10.1145/1979742.1979706](https://doi.org/10.1145/1979742.1979706). URL: <http://doi.acm.org/10.1145/1979742.1979706>.
- [46] Eric Freeman and David Gelernter. "Lifestreams: A Storage Model for Personal Data." In: *SIGMOD Rec.* 25.1 (Mar. 1996), pp. 80–86. ISSN: 0163-5808. DOI: [10.1145/381854.381893](https://doi.org/10.1145/381854.381893). URL: <http://doi.acm.org/10.1145/381854.381893>.
- [47] Soroush Ghorashi and Carlos Jensen. "Leyline: Provenance-based Search Using a Graphical Sketchpad." In: *Proceedings of the Symposium on Human-Computer Interaction and Information Retrieval*. HCIR '12. Cambridge, California, USA: ACM, 2012, 2:1–2:10. ISBN: 978-1-4503-1796-2. DOI: [10.1145/2391224.2391226](https://doi.org/10.1145/2391224.2391226). URL: <http://doi.acm.org/10.1145/2391224.2391226>.
- [48] Adele J Goldberg. "SMALLTALK-80: the interactive programming environment." In: (1984).
- [49] Victor M. González and Gloria Mark. "'Constant, Constant, Multi-tasking Craziness': Managing Multiple Working Spheres." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 113–120. ISBN: 1-58113-702-8. DOI: [10.1145/985692.985707](https://doi.org/10.1145/985692.985707). URL: <http://doi.acm.org/10.1145/985692.985707>.
- [50] Sandy JJ Gould, Duncan P Brumby, and Anna L Cox. "What does it mean for an interruption to be relevant? An investigation of relevance as a memory effect." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 57. 1. SAGE Publications. 2013, pp. 149–153.
- [51] Jonathan Grudin. "The Computer Reaches out: The Historical Continuity of Interface Design." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, Washington, USA: ACM, 1990, pp. 261–268. ISBN: 0-201-50932-6. DOI: [10.1145/97243.97284](https://doi.org/10.1145/97243.97284). URL: <http://doi.acm.org/10.1145/97243.97284>.
- [52] Jonathan Grudin. "Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use." In: *Proc. CHI*. Seattle, Washington, USA: ACM, 2001, pp. 458–465. ISBN: 1-58113-327-8. DOI: [10.1145/365024.365312](https://doi.org/10.1145/365024.365312). URL: <http://doi.acm.org/10.1145/365024.365312>.

- [53] Megan Hardy and Douglas J Gillan. "Voluntary task switching patterns in everyday tasks of different motivational levels." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 56. 1. SAGE Publications. 2012, pp. 2128–2132.
- [54] Sandra G Hart. "NASA-task load index (NASA-TLX); 20 years later." In: *Proceedings of the Human Factors and Ergonomics Society 50th Annual Meeting*. HFES '06. SAGE, 2006, pp. 904–908. DOI: [10.1177/154193120605000909](https://doi.org/10.1177/154193120605000909).
- [55] D. Austin Henderson Jr. and Stuart Card. "Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-based Graphical User Interface." In: *ACM Trans. Graph.* 5.3 (July 1986), pp. 211–243. ISSN: 0730-0301. DOI: [10.1145/24054.24056](https://doi.org/10.1145/24054.24056). URL: <http://doi.acm.org/10.1145/24054.24056>.
- [56] Sarah Henderson. "Genre, task, topic and time: facets of personal digital document management." In: *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*. ACM. 2005, pp. 75–82.
- [57] Steven Houben. "An Activity-Centric Approach to Configuration Work in Distributed Interaction." PhD thesis. IT University of Copenhagen, 2014.
- [58] Steven Houben, Jakob E. Bardram, Jo Vermeulen, Kris Luyten, and Karin Coninx. "Activity-centric Support for Ad Hoc Knowledge Work: A Case Study of Co-activity Manager." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, 2013, pp. 2263–2272. ISBN: 978-1-4503-1899-0. DOI: [10.1145/2470654.2481312](https://doi.org/10.1145/2470654.2481312). URL: <http://doi.acm.org/10.1145/2470654.2481312>.
- [59] Steven Houben, Søren Nielsen, Morten Esbensen, and Jakob E. Bardram. "NooSphere: An Activity-centric Infrastructure for Distributed Interaction." In: *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*. MUM '13. Luleå, Sweden: ACM, 2013, 13:1–13:10. ISBN: 978-1-4503-2648-3. DOI: [10.1145/2541831.2541856](https://doi.org/10.1145/2541831.2541856). URL: <http://doi.acm.org/10.1145/2541831.2541856>.
- [60] Steven Houben, Paolo Tell, and Jakob E. Bardram. "ActivitySpace: Managing Device Ecologies in an Activity-Centric Configuration Space." In: *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. ITS '14. Dresden, Germany: ACM, 2014, pp. 119–128. ISBN: 978-1-4503-2587-5. DOI: [10.1145/2669485.2669493](https://doi.org/10.1145/2669485.2669493). URL: <http://doi.acm.org/10.1145/2669485.2669493>.

- [61] Christian P Janssen, Sandy JJ Gould, Simon YW Li, Duncan P Brumby, and Anna L Cox. "Integrating knowledge of multitasking and Interruptions across different Perspectives and research methods." In: *International Journal of Human-Computer Studies* 79 (2015), pp. 1–5.
- [62] Steven Jeuris and Jakob E. Bardram. "Dedicated workspaces: Faster resumption times and reduced cognitive load in sequential multitasking." In: *Computers in Human Behavior* 62 (2016), pp. 404–414. ISSN: 0747-5632. DOI: <http://dx.doi.org/10.1016/j.chb.2016.03.059>. URL: <http://www.sciencedirect.com/science/article/pii/S0747563216302308>.
- [63] Steven Jeuris and Steven Houben. "Temporal Model for Reflective Multitasking." In: *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '13. 2013.
- [64] Steven Jeuris, Steven Houben, and Jakob E. Bardram. "Laevo: A Temporal Desktop Interface for Integrated Knowledge Work." In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14. Honolulu, Hawaii, USA: ACM, 2014, pp. 679–688. ISBN: 978-1-4503-3069-5. DOI: [10.1145/2642918.2647391](http://doi.acm.org/10.1145/2642918.2647391). URL: <http://doi.acm.org/10.1145/2642918.2647391>.
- [65] Steven Jeuris, Paolo Tell, and Jakob E. Bardram. *co-Laevo: Supporting Cooperating Teams by Working 'within' Shared Activity Time Lines*. Tech. rep. ITU-TR-2016-193. IT University of Copenhagen, June 2016. URL: <http://en.itu.dk/Research/About-ITUs-Research/Technical-Reports/Technical-Reports-Archive/2016/TR-2016-193>.
- [66] Steven Jeuris, Paolo Tell, Steven Houben, and Jakob E. Bardram. "The Hidden Cost of Task Switching: How Well Do Window Managers Support Sequential Multitasking?" In: In submission.
- [67] Jing Jin and Laura A Dabbish. "Self-interruption on the computer: a typology of discretionary task interleaving." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2009, pp. 1799–1808.
- [68] Jeff Johnson, Teresa L Roberts, William Verplank, David C Smith, Charles H Irby, Marian Beard, and Kevin Mackey. "The Xerox Star: A Retrospective." In: *Computer* 22.9 (1989), pp. 28–29.
- [69] William Jones, Ammy Jiranida Phuwanartnurak, Rajdeep Gill, and Harry Bruce. "Don't take my folders away!: organizing personal information to get ghings done." In: *CHI'05 extended abstracts on Human factors in computing systems*. ACM. 2005, pp. 1505–1508.

- [70] Eser Kandogan and Ben Shneiderman. "Elastic Windows: Evaluation of Multi-window Operations." In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '97. Atlanta, Georgia, USA: ACM, 1997, pp. 250–257. ISBN: 0-89791-802-9. DOI: [10.1145/258549.258720](https://doi.org/10.1145/258549.258720). URL: <http://doi.acm.org/10.1145/258549.258720>.
- [71] Victor Kaptelinin. "Context and Consciousness." In: ed. by Bonnie A. Nardi. Cambridge, MA, USA: Massachusetts Institute of Technology, 1995. Chap. Computer-mediated Activity: Functional Organs in Social and Developmental Contexts, pp. 45–68. ISBN: 0-262-14058-6. URL: <http://dl.acm.org/citation.cfm?id=223826.223829>.
- [72] Victor Kaptelinin. "UMEA: Translating Interaction Histories into Project Contexts." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 353–360. ISBN: 1-58113-630-7. DOI: [10.1145/642611.642673](https://doi.org/10.1145/642611.642673). URL: <http://doi.acm.org/10.1145/642611.642673>.
- [73] Victor Kaptelinin. "Activity Theory." In: *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* Ed. by Mads Soegaard and Rikke Friis Dam. Aarhus, Denmark: The Interaction Design Foundation, 2014. Chap. 16.
- [74] Victor Kaptelinin and Bonnie A. Nardi. *Acting with Technology: Activity Theory and Interaction Design*. The MIT Press, 2009.
- [75] Alan C. Kay. "The Early History of Smalltalk." In: *The Second ACM SIGPLAN Conference on History of Programming Languages*. HOPL-II. Cambridge, Massachusetts, USA: ACM, 1993, pp. 69–95. ISBN: 0-89791-570-4. DOI: [10.1145/154766.155364](https://doi.org/10.1145/154766.155364). URL: <http://doi.acm.org/10.1145/154766.155364>.
- [76] Mik Kersten and Gail C. Murphy. "Using Task Context to Improve Programmer Productivity." In: *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. SIGSOFT '06/FSE-14. Portland, Oregon, USA: ACM, 2006, pp. 1–11. ISBN: 1-59593-468-5. DOI: [10.1145/1181775.1181777](https://doi.org/10.1145/1181775.1181777). URL: <http://doi.acm.org/10.1145/1181775.1181777>.
- [77] Alison Kidd. "The Marks Are on the Knowledge Worker." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '94. Boston, Massachusetts, USA: ACM, 1994, pp. 186–191. ISBN: 0-89791-650-6. DOI: [10.1145/191666.191740](https://doi.org/10.1145/191666.191740). URL: <http://doi.acm.org/10.1145/191666.191740>.
- [78] Andrea Kiesel, Marco Steinhauser, Mike Wendt, Michael Falkenstein, Kerstin Jost, Andrea M Philipp, and Iring Koch. "Control and interference in task switching—A review." In: *Psychological bulletin* 136.5 (2010), p. 849.

- [79] Vassilis Kostakos. "The Big Hole in HCI Research." In: *interactions* 22.2 (Feb. 2015), pp. 48–51. ISSN: 1072-5520. DOI: [10.1145/2729103](https://doi.org/10.1145/2729103). URL: <http://doi.acm.org/10.1145/2729103>.
- [80] Kari Kuutti. "Activity theory as a potential framework for human-computer interaction research." In: *Context and consciousness: Activity theory and human-computer interaction* (1996), pp. 17–44.
- [81] Kari Kuutti and Liam J. Bannon. "Searching for unity among diversity: exploring the "interface" concept." In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 263–268. ISBN: 0-89791-575-5. DOI: [10.1145/169059.169206](https://doi.org/10.1145/169059.169206). URL: <http://doi.acm.org/10.1145/169059.169206>.
- [82] Daniël Lakens. "Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs." In: *Frontiers in psychology* 4 (2013), p. 863.
- [83] Mik Lamming and Mike Flynn. "Forget-me-not: Intimate computing in support of human memory." In: *Proc. FRIEND21, 1994 Int. Symp. on Next Generation Human Interface*. 1994, p. 4.
- [84] Kevin Larson, Maarten van Dantzich, Mary Czerwinski, and George Robertson. "Text in 3D: Some Legibility Results." In: *CHI '00 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '00. The Hague, The Netherlands: ACM, 2000, pp. 145–146. ISBN: 1-58113-248-4. DOI: [10.1145/633292.633374](https://doi.org/10.1145/633292.633374). URL: <http://doi.acm.org/10.1145/633292.633374>.
- [85] Yong Liu, Jorge Goncalves, Denzil Ferreira, Bei Xiao, Simo Hosio, and Vassilis Kostakos. "CHI 1994-2013: Mapping Two Decades of Intellectual Progress Through Co-word Analysis." In: *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, pp. 3553–3562. ISBN: 978-1-4503-2473-1. DOI: [10.1145/2556288.2556969](https://doi.org/10.1145/2556288.2556969). URL: <http://doi.acm.org/10.1145/2556288.2556969>.
- [86] Wendy E. Mackay and Anne-Laure Fayard. "HCI, Natural Science and Design: A Framework for Triangulation Across Disciplines." In: *Proceedings of the 2Nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*. DIS '97. Amsterdam, The Netherlands: ACM, 1997, pp. 223–234. ISBN: 0-89791-863-0. DOI: [10.1145/263552.263612](https://doi.org/10.1145/263552.263612). URL: <http://doi.acm.org/10.1145/263552.263612>.
- [87] Thomas W. Malone. "How Do People Organize Their Desks?: Implications for the Design of Office Information Systems." In: *ACM Trans. Inf. Syst.* 1.1 (Jan. 1983), pp. 99–112. ISSN: 1046-

8188. DOI: [10.1145/357423.357430](https://doi.org/10.1145/357423.357430). URL: <http://doi.acm.org/10.1145/357423.357430>.
- [88] Hans Marien, Ruud Custers, Ran R Hassin, and Henk Aarts. "Unconscious goal activation and the hijacking of the executive function." In: *Journal of personality and social psychology* 103.3 (2012), p. 399.
- [89] Gloria Mark, Victor M. Gonzalez, and Justin Harris. "No Task Left Behind?: Examining the Nature of Fragmented Work." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '05. Portland, Oregon, USA: ACM, 2005, pp. 321–330. ISBN: 1-58113-998-5. DOI: [10.1145/1054972.1055017](https://doi.org/10.1145/1054972.1055017). URL: <http://doi.acm.org/10.1145/1054972.1055017>.
- [90] Gloria Mark, Daniela Gudith, and Ulrich Klocke. "The Cost of Interrupted Work: More Speed and Stress." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 107–110. ISBN: 978-1-60558-011-1. DOI: [10.1145/1357054.1357072](https://doi.org/10.1145/1357054.1357072). URL: <http://doi.acm.org/10.1145/1357054.1357072>.
- [91] Gloria Mark, Stephen Volda, and Armand Cardello. "'A Pace Not Dictated by Electrons': An Empirical Study of Work Without Email." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, pp. 555–564. ISBN: 978-1-4503-1015-4. DOI: [10.1145/2207676.2207754](https://doi.org/10.1145/2207676.2207754). URL: <http://doi.acm.org/10.1145/2207676.2207754>.
- [92] EJ Masicampo and Roy F Baumeister. "Consider it done! Plan making can eliminate the cognitive effects of unfulfilled goals." In: *Journal of personality and social psychology* 101.4 (2011), p. 667.
- [93] Yoshiro Miyata and Donald A Norman. "Psychological issues in support of multiple activities." In: *User centered system design: New perspectives on human-computer interaction* (1986), pp. 265–284.
- [94] Christopher A Monk, J Gregory Trafton, and Deborah A Boehm-Davis. "The effect of interruption duration and demand on resuming suspended goals." In: *Journal of Experimental Psychology: Applied* 14.4 (2008), p. 299.
- [95] Ingrid Mulder, Henk de Poot, Carla Verwij, Ruud Janssen, and Marcel Bijlsma. "An Information Overload Study: Using Design Methods for Understanding." In: *Proceedings of the 18th Australia Conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments*. OZCHI '06. Sydney, Australia: ACM, 2006, pp. 245–252. ISBN: 1-59593-545-2. DOI: [10.1145/1228175.1228218](https://doi.org/10.1145/1228175.1228218). URL: <http://doi.acm.org/10.1145/1228175.1228218>.

- [96] Michael J. Muller, Werner Geyer, Beth Brownholtz, Eric Wilcox, and David R. Millen. "One-hundred Days in an Activity-centric Collaboration Environment Based on Shared Objects." In: *CHI'04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Feb. 10, 2006, pp. 375–382. ISBN: 1-58113-702-8. URL: <http://dblp.uni-trier.de/db/conf/chi/chi2004.html#MullerGBWM04>.
- [97] Bonnie A. Nardi. *Context and consciousness: activity theory and human-computer interaction*. The MIT Press, 1995.
- [98] Otto Neurath. "The departmentalization of unified science." In: *Erkenntnis* 7.1 (1937), pp. 240–246.
- [99] Otto Neurath. "Unified science and its encyclopaedia." In: *Philosophy of Science* 4.2 (1937), pp. 265–277.
- [100] Otto Neurath. "Universal jargon and terminology." In: *Proceedings of the Aristotelian Society*. Vol. 41. JSTOR. 1940, pp. 127–148.
- [101] Donald A. Norman. *The invisible computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. MIT press, 1998.
- [102] Donald A. Norman. "Affordance, conventions, and design." In: *interactions* 6.3 (1999), pp. 38–43.
- [103] Donald A. Norman. "Technology First, Needs Last: The Research-product Gulf." In: *interactions* 17.2 (Mar. 2010), pp. 38–42. ISSN: 1072-5520. DOI: [10.1145/1699775.1699784](https://doi.org/10.1145/1699775.1699784). URL: <http://doi.acm.org/10.1145/1699775.1699784>.
- [104] Donald A. Norman and Stephen W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1986. ISBN: 0898597811.
- [105] Donald A. Norman and R. Verganti. "Incremental and Radical Innovation: Design Research vs. Technology and Meaning Change." In: *Design Issues* 30.1 (2014), pp. 78–96. ISSN: 0747-9360. DOI: [10.1162/DESI_a_00250](https://doi.org/10.1162/DESI_a_00250).
- [106] Gerard Oleksik, Max L. Wilson, Craig Tashman, Eduarda Mendes Rodrigues, Gabriella Kazai, Gavin Smyth, Natasa Milic-Frayling, and Rachel Jones. "Lightweight Tagging Expands Information and Activity Management Practices." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 279–288. ISBN: 978-1-60558-246-7. DOI: [10.1145/1518701.1518746](https://doi.org/10.1145/1518701.1518746). URL: <http://doi.acm.org/10.1145/1518701.1518746>.
- [107] Stephen J Payne, Geoffrey B Duggan, and Hansjörg Neth. "Discretionary task interleaving: heuristics for time allocation in cognitive foraging." In: *Journal of Experimental Psychology: General* 136.3 (2007), pp. 370–388. DOI: [10.1037/0096-3445.136.3.370](https://doi.org/10.1037/0096-3445.136.3.370). URL: <http://dx.doi.org/10.1037/0096-3445.136.3.370>.

- [108] Tye Rattenbury and John F. Canny. "CAAD: An Automatic Task Support System." In: *CHI'07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, June 7, 2007, pp. 687–696. ISBN: 978-1-59593-593-9. URL: <http://dblp.uni-trier.de/db/conf/chi/chi2007.html#RattenburyC07>.
- [109] George A. Reisch. "Planning science: Otto Neurath and the international encyclopedia of unified science." In: *The British Journal for the History of Science* 27.02 (1994), pp. 153–175.
- [110] Jun Rekimoto. "TimeScope: A Time Machine for the Desktop Environment." In: *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '99. Pittsburgh, Pennsylvania: ACM, 1999, pp. 180–181. ISBN: 1-58113-158-5. DOI: [10.1145/632716.632830](https://doi.org/10.1145/632716.632830). URL: <http://doi.acm.org/10.1145/632716.632830>.
- [111] Meredith Ringel. "When One Isn't Enough: An Analysis of Virtual Desktop Usage Strategies and Their Implications for Design." In: *CHI EA*. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 762–763. ISBN: 1-58113-637-4. DOI: [10.1145/765891.765976](https://doi.org/10.1145/765891.765976). URL: <http://doi.acm.org/10.1145/765891.765976>.
- [112] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Ridsen, David Thiel, and Vadim Gorokhovskiy. "The Task Gallery: A 3D Window Manager." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '00. The Hague, The Netherlands: ACM, 2000, pp. 494–501. ISBN: 1-58113-216-6. DOI: [10.1145/332040.332482](https://doi.org/10.1145/332040.332482). URL: <http://doi.acm.org/10.1145/332040.332482>.
- [113] George Robertson, Eric Horvitz, Mary Czerwinski, Patrick Baudisch, Dugald Ralph Hutchings, Brian Meyers, Daniel Robbins, and Greg Smith. "Scalable Fabric: Flexible Task Management." In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli, Italy: ACM, 2004, pp. 85–89. ISBN: 1-58113-867-9. DOI: [10.1145/989863.989874](https://doi.org/10.1145/989863.989874). URL: <http://doi.acm.org/10.1145/989863.989874>.
- [114] Yvonne Rogers, Liam Bannon, and Graham Button. "Rethinking theoretical frameworks for HCI: report on an INTERCHI'93 workshop, Amsterdam, 24–25th April, 1993." In: *ACM SIGCHI Bulletin* 26.1 (1994), pp. 28–30.
- [115] Dario D. Salvucci, Niels A. Taatgen, and Jelmer P. Borst. "Toward a Unified Theory of the Multitasking Continuum: From Concurrent Performance to Task Switching, Interruption, and Resumption." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 1819–1828. ISBN: 978-1-60558-246-7. DOI: [10.1145/1555734.1555734](https://doi.org/10.1145/1555734.1555734).

- 1145/1518701.1518981. URL: <http://doi.acm.org/10.1145/1518701.1518981>.
- [116] Stephanie Santosa and Daniel Wigdor. "A field study of multi-device workflows in distributed workspaces." In: *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM. 2013, pp. 63–72.
- [117] Corina Sas, Steve Whittaker, Steven Dow, Jodi Forlizzi, and John Zimmerman. "Generating Implications for Design Through Design Research." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, pp. 1971–1980. ISBN: 978-1-4503-2473-1. DOI: [10.1145/2556288.2557357](https://doi.org/10.1145/2556288.2557357). URL: <http://doi.acm.org/10.1145/2556288.2557357>.
- [118] Kjeld Schmidt and Liam Bannon. "Taking CSCW seriously." In: *Computer Supported Cooperative Work (CSCW) 1.1-2 (1992)*, pp. 7–40.
- [119] Kjeld Schmidt and Liam Bannon. "Constructing CSCW: The first quarter century." In: *Computer Supported Cooperative Work (CSCW) 22.4-6 (2013)*, pp. 345–372.
- [120] Kjeld Schmidt and Carla Simone. "Coordination mechanisms: Towards a conceptual foundation of CSCW systems design." In: *Computer Supported Cooperative Work (CSCW) 5.2-3 (1996)*, pp. 155–200.
- [121] Kjeld Schmidt and Carla Simone. "Mind the gap! Towards a unified view of CSCW." In: *Proceedings of the Fifth International Conference on the Design of Cooperative Systems*. COOP '00. Sophia Antipolis, France: IOS Press, 2000, pp. 205–221.
- [122] Ben Shneiderman. "The eyes have it: A task by data type taxonomy for information visualizations." In: *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE. 1996, pp. 336–343.
- [123] D. C. Smith, C. Irby, R. Kimball, W. L. Verplank, and E. Harslem. "Designing the Star User Interface." In: *Byte 7.4 (1982)*, pp. 242–282.
- [124] Greg Smith, Patrick Baudisch, George Robertson, Mary Czerwinski, Brian Meyers, Daniel Robbins, and Donna Andrews. "Groupbar: The taskbar evolved." In: *Proceedings of OZCHI*. Vol. 3. 2003, p. 10.
- [125] David L Strayer and William A Johnston. "Driven to distraction: Dual-task studies of simulated driving and conversing on a cellular telephone." In: *Psychological science 12.6 (2001)*, pp. 462–466.
- [126] Lucy A. Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridge university press, 1987.

- [127] Ivan E. Sutherland. "Sketchpad: A Man-machine Graphical Communication System." In: *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*. AFIPS '63 (Spring). Detroit, Michigan: ACM, 1963, pp. 329–346. DOI: [10.1145/1461551.1461591](https://doi.org/10.1145/1461551.1461591). URL: <http://doi.acm.org/10.1145/1461551.1461591>.
- [128] Craig Tashman. "WindowScape: A Task Oriented Window Manager." In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. UIST '06. Montreux, Switzerland: ACM, 2006, pp. 77–80. ISBN: 1-59593-313-1. DOI: [10.1145/1166253.1166266](https://doi.org/10.1145/1166253.1166266). URL: <http://doi.acm.org/10.1145/1166253.1166266>.
- [129] Warren Teitelman. "Ten years of window systems-A retrospective view." In: *Methodology of Window Management*. Springer, 1986, pp. 35–46.
- [130] David Thomas and Andrew Hunt. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.
- [131] J.Gregory Trafton, Erik M Altmann, Derek P Brock, and Farilee E Mintz. "Preparing to resume an interrupted task: effects of prospective goal encoding and retrospective rehearsal." In: *International Journal of Human-Computer Studies* 58.5 (2003), pp. 583–603. ISSN: 1071-5819. DOI: [http://dx.doi.org/10.1016/S1071-5819\(03\)00023-5](https://dx.doi.org/10.1016/S1071-5819(03)00023-5). URL: <http://www.sciencedirect.com/science/article/pii/S1071581903000235>.
- [132] Manas Tungare, Manuel Perez-Quinones, and Alyssa Sams. "An exploratory study of calendar use." In: *arXiv preprint arXiv:0809.3447* (2008).
- [133] Liam D. Turner, Stuart M. Allen, and Roger M. Whitaker. "Interruptibility Prediction for Ubiquitous Systems: Conventions and New Directions from a Growing Field." In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 801–812. ISBN: 978-1-4503-3574-4. DOI: [10.1145/2750858.2807514](https://doi.org/10.1145/2750858.2807514). URL: <http://doi.acm.org/10.1145/2750858.2807514>.
- [134] Bret Victor. *Up and Down the Ladder of Abstraction*. 2011. URL: <http://worrydream.com/LadderOfAbstraction/> (visited on 04/03/2016).
- [135] Stephen Volda and Elizabeth D. Mynatt. "It Feels Better Than Filing: Everyday Work Experiences in an Activity-based Computing System." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 259–268. ISBN: 978-1-60558-246-7. DOI: [10.1145/1518701.1518744](https://doi.org/10.1145/1518701.1518744). URL: <http://doi.acm.org/10.1145/1518701.1518744>.

- [136] Stephen Volda, Elizabeth D. Mynatt, and W. Keith Edwards. "Re-framing the Desktop Interface Around the Activities of Knowledge Work." In: *Proc. UIST*. Monterey, CA, USA: ACM, 2008, pp. 211–220. ISBN: 978-1-59593-975-3. DOI: [10.1145/1449715.1449751](https://doi.org/10.1145/1449715.1449751). URL: <http://doi.acm.org/10.1145/1449715.1449751>.
- [137] Mark Weiser. "The computer for the 21st century." In: *Scientific american* 265.3 (1991), pp. 94–104.
- [138] Steve Whittaker. "Personal information management: from information consumption to curation." In: *Annual review of information science and technology* 45.1 (2011), pp. 1–62.
- [139] Andrea A. diSessa. "A Principled Design for an Integrated Computational Environment." In: *Hum.-Comput. Interact.* 1.1 (Mar. 1985), pp. 1–47. ISSN: 0737-0024. DOI: [10.1207/s15327051hci0101_1](https://doi.org/10.1207/s15327051hci0101_1). URL: http://dx.doi.org/10.1207/s15327051hci0101_1.

DECLARATION

This thesis is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgment of collaborative research and discussions. The work was done under the guidance of Professor Jakob E. Bardram, at the IT University of Copenhagen.

Copenhagen, July 2016

Steven Jeuris

COLOPHON

Many of the icons presented in this dissertation were made by [Freepik](#) from www.flaticon.com.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both \LaTeX and \LyX :

<https://bitbucket.org/amiede/classicthesis/>