



Realizing Elastic Design Principles for User Exploration in Bayesian Analysis

Master's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Devashish Jasani

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Dr. Matthias Kaiser, SAP SE

Registration date: 01.07.2016
Submission date: 13.02.2017

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

**Vertraulichkeits- und Sperrvermerks Erklärung bezüglich der
Masterarbeit
(Abschlussarbeit)**

Mit dem Titel: **"Establishing Contextual Adaptation in Transformative User Experience Design"**

der/des Studierenden:

Devashish Jasani

Matrikel Nr. 340996, Studiengang Software Systems Engineering

durchgeführt in Zusammenarbeit mit **SAP SE**, nachfolgend „Vertragspartner“ genannt.

Die Hochschule **„Rheinisch-Westfälische Technische Hochschule Aachen“**, vertreten durch den Rektor/in, sowie die für die Begutachtung der vorstehend bezeichneten Abschlussarbeit

1. **Prof. Dr. J. Borchers** (als Erstgutachter),
2. (als Zweitgutachter),

nachstehend „Prüfungsbeauftragte“ genannt,

bestätigen, dass sie davon Kenntnis erhalten haben, dass die Abschlussarbeit, sowie die darin enthaltenen vertraulichen Daten des Vertragspartners der Vertraulichkeit unterliegen.

Sie verpflichten sich, alle firmen- bzw. betriebsinternen Informationen, insbesondere technische und wirtschaftliche Informationen sowie Absichten, Erfahrungen, Erkenntnisse, Konstruktionen und Unterlagen, die ihnen aus der Kenntnisnahme der Abschlussarbeit zugänglich gemacht werden, vertraulich zu behandeln, und Dritten nicht zugänglich zu machen.

Diese Vertraulichkeit gilt nicht für Informationen,

1. die den Prüfungsbeauftragten bereits vor Inkrafttreten dieses Vertrages bekannt waren,
2. die die Prüfungsbeauftragten rechtmäßig von Dritten ohne Auferlegung einer Vertraulichkeitsverpflichtung erhalten,
3. die allgemein bekannt sind oder ohne Verstoß gegen in diesem Vertrag erhaltenen Verpflichtungen allgemein bekannt werden,
4. die die Prüfungsbeauftragten im Rahmen eigener unabhängiger Entwicklungen erarbeitet haben.

Sollte im Rahmen des Prüfungsverfahrens das Hinzuziehen weiterer, aufgrund von Hochschulgesetzen oder Prüfungsordnung befugter Prüfungsbeauftragter notwendig sein (z.B. Mitglieder des zuständigen Prüfungsausschusses oder der Hochschulleitung), so sind diese ebenfalls in diese Vereinbarung mit einzubeziehen und dem Vertragspartner zu nennen.


Darüber hinaus wird die Abschlussarbeit mit einem **Sperrvermerk** versehen und darf weiteren Personen nicht zugänglich gemacht werden. Eine Veröffentlichung oder Vervielfältigung der Abschlussarbeit - auch nur auszugsweise - ist ohne ausdrückliche Genehmigung des Vertragspartners nicht gestattet. *[Dieser Absatz ist zu streichen, wenn eine separate unterzeichnete Sperrvermerkserklärung seitens der Hochschule bereits vorliegt wurde]*

Haftungsansprüche des Vertragspartners gegenüber der Hochschule und den Prüfungsbeauftragten aufgrund von Verstößen gegen diese Vereinbarung können nur dann geltend gemacht werden, wenn ihnen durch den Vertragspartner grobe Fahrlässigkeit oder Vorsatz nachgewiesen werden.

Es gilt ausschließlich das Recht der Bundesrepublik Deutschland.

Erstgutachter:

Zweitgutachter:


.....
(Name:)

.....
(Name:)

Ort: Aachen,

Ort:.....

Datum: 18.05.16

Datum:

Prof. Dr. Jan Borchers
Lehrstuhl für Informatik 10
Medieninformatik und
Mensch-Computer-Interaktion
RWTH Aachen
Löhrenstr. 55 • 52074 Aachen

Contents

Abstract	xi
Acknowledgements	xiii
Conventions	xv
1 Introduction	1
1.1 Contributions	5
1.2 Outline	5
2 Related Work	7
2.1 Complex Problem Solving	7
2.1.1 Characteristics of Complex Problem Solving	8
2.1.2 Core Activities	11
2.2 Activity Theory	13
2.3 End User Development	15
3 Transformative User Experience Design	19

3.1	TUX proposed system architecture	20
3.1.1	Task Context	21
3.1.2	Task Object	22
4	Bayesian Analysis Theory	25
4.1	Theory	26
4.2	Performing Bayesian analysis with a simple example	27
4.3	Workflow	29
5	Interaction Design	31
5.1	Design Requirements and Conception	31
5.2	Design	35
5.2.1	TUX Container design	37
5.2.2	TUX object design	38
5.2.3	Sourcing Application	39
5.2.4	Planning Application	42
5.2.5	Grouping Application	43
5.2.6	Comparing Application	46
5.2.7	Visualizing Application	47
6	Implementation	51
6.1	Web Components and Polymer	52
6.1.1	Custom Elements	53

6.1.2	Shadow DOM	54
6.1.3	HTML Imports	54
6.2	Component Based Behavior	55
6.2.1	CBB Implementation	56
6.2.2	Model Behavior	58
6.2.3	Draggable Behavior	61
6.2.4	Dropzone Behavior	62
6.2.5	Behavior Sortable	62
6.3	Implementing Task objects and containers . .	62
6.4	Implementation for Bayesian Analysis	65
7	Evaluation	67
7.1	User Study Protocol	67
7.1.1	Setup	67
7.1.2	Procedure	68
7.1.3	Method of Analysis	69
7.1.4	Participants	71
7.2	Results	71
7.2.1	System Usability	71
7.2.2	Freedom offered by the system	72
7.2.3	User Feedback	74
8	Summary and future work	77

8.1	Summary	77
8.2	Future work	79
A	Appendix for User Study	81
	Bibliography	85
	Index	89

List of Figures

3.1	Contextual Consumption of Business Entities	24
5.1	Prototype showing applications arranged on a canvas a.k.a workspace	36
5.2	Describes the common design and behavior of (a) Task containers, (b) Task objects	38
5.3	Prototype of spreadsheet application with interactions for mouse hover over and mouse click for a particular cell	40
5.4	Dragging cell in a spreadsheet application	40
5.5	Describes interactions for mouse hover over a column header	41
5.6	Shows a snapshot of the final implemented version of Sourcing application	41
5.7	Describes the interface for planning container	43
5.8	Describes the interface for Grouping container.	44
5.9	Shows the interface for grouping after dropping a file in the suggestions area.	45
5.10	Describes the interface for Comparing application.	46

5.11	Shows the interface for comparing application after dropping two files. Each file is visualized as a table.	47
5.12	Comparing result gets rendered and the result is displayed as a chart. Alternative tabular visualization option is also provided. . .	48
5.13	Interface for visualizing application	49
5.14	Shows the interface after dropping group "Group=Condition" from Grouping application into the Visualizing Container.	49
5.15	Shows how to create a custom visualization.	50
5.16	All applications arranged on a canvas.	50
6.1	Shows (a) vanilla button on mouse click (b) button with a paper-ripple behavior	57
7.1	Grade ranking of SUS scores from [Bangor et al., 2009]	72
A.1	Collecting user details and demographic data form	82
A.2	SUS form	83
A.3	Feedback	84

Abstract

People use multiple tools and standardized practices during their day to day work. But while conducting complex investigations alongside these standardized tools and practices, problem solvers bring their intent, insight and ingenuity to shape data, tasks, methods, processes and strategies. Our vision centers on understanding and designing for synergetic activities in user's dynamic work to ensure that applications for complex problem solving are truly useful. Designing useful software support for such complex inquires fosters exploratory analysis, adaptation to dynamic work needs and extends knowledge and communication.

Transformative User Experience (TUX) approach emphasizes on elasticity being the inherent quality of such a system, which enables users to spontaneously reshape their environment and the meaning of objects during usage. Using research in the field of complex problem solving, Activity Theory as the theoretical HCI framework for task analysis, TUX and Direct manipulation as our core conceptual frameworks, we narrowed down requirements and design implications to build generic useful software support for complex problem solving and implement the same using an open source web-components library, Polymer. After realizing generic software support for complex problem solving we integrate abilities for appropriating our system for Bayesian analysis.

We conducted a qualitative user study with seven participants to measure the usability and exploratory power of our system. Results from the study show that the system allowed enough freedom for participants to explore a given problem landscape in their own independent way, not bound by a predefined workflow to achieve their goals.

Acknowledgements

I would like to thank Krishna Subramaniam, M.Sc., my supervisor for your constant support and feedback. I really appreciate your patience and help in spite of me being a remote thesis student. I would also like to thank Prof. Jan Borchers for providing me an opportunity to work on my thesis under your department's supervision.

Secondly, I would like to thank Markus Latzina for this opportunity to work on the very exciting area of TUX. It has been a valuable and enriching experience to be able to work on this thesis with you. I would like to thank Dr. Matthias Kaiser for becoming my thesis second supervisor. Your insights have helped me think beyond. I would like to thank Dr. Knut Manske for providing a stimulating and challenging environment at SAP for my thesis.

I would like to thank all users who gave their valuable time and participated in the user study.

I would also like to thank my friends Srivatsan, Stephen, Chitresh, Shilpa, Ravi, Sara, Mujtaba and everyone else for all the patience and advice.

Finally I would like to thank my parents, my siter and Ananya Kuri for their constant support in my life.

Thank you very much!
Devashish

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:

Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in Canadian English.

Download links are set off in coloured boxes.

File: [myFile](#)^a

^ahttp://hci.rwth-aachen.de/public/folder/file_number.file

Chapter 1

Introduction

“If it is so critical to understand the particular users of a product, then what happens when a product is designed to be used by almost anyone in the world? This is paradoxical”

—Donald Norman [Norman, 2005]

Problem solving is an elemental part of our everyday lives. We do problem solving rite from deciding which restaurant to eat at tonight; what apartment to rent; fixing and assembling furniture at home; finding the shortest commute to work; applying for income tax return; to writing a complex software; devising a new market strategy. We use multiple tools embedded in our work and home environments to resolve issues at hand. Sometimes we use standardized mechanisms like assembling furniture by looking at a user self construction guide. Follow best practices and rules like using the seven hat technique during brain storming. At times, we encounter problems where there is no clear starting point of investigation nor an end point; new problems and opportunities keep emerging the more we explore. There may be enormous amount of data involved which needs to be analyzed or there may not be a clear correct answer. A few examples of such problems are performing market research, hiring a candidate, developing a project plan, prescribing a course of medical treatment for a patient. In such situations,

People use their own creativity, insight and experience to solve a problem uniquely in their own style.

problem solvers may require automated and standardized tasks and processes from time to time, but at some point while conducting complex investigations they would be required to bring their own insight, ingenuity and intent to shape data, tasks, methods, processes and strategies. All of this makes a problem complex.

Complex interdependencies, encountering new problems on the fly and evolving goals characterize complex problems.

Complex problems are difficult to solve because of their contradicting, constantly changing and incomplete requirements, which are often difficult to recognize. And, due to complex interdependencies, the pursuit to resolving one aspect of such a problem may unveil or fabricate other problems. Complex tasks may use mostly standardized and automated processes and have definite right answers but are complex because of the volume and complexity of data and or have numerous permutations or possibilities to explore. These characteristics of complex problem solving imbibed in multiple contexts fuel the strategies that problem solvers follow.

Barbara Mirel records the following anecdotes from her interviews and study with professionals from various walks of life. The anecdotes describe how difficult it is to use softwares for complex problem solving [Mirel, 2003]:

- *Director of category management at a well known retailing firm expressed that their analysts are so overwhelmed by information that they end up hurling data without even analyzing it properly. They need software support that helps them make sense of the numbers from different perspectives.*
- *Nursing Manager at a popular countrywide hospitals association said that their drug program did not aim to disrupt or curb clinical judgment, but it certainly did. This restricted what their nurses could do and when and how they could do it. The program caused more problems than it solved.*
- *Network Analyst at telecommunications company shared that his biggest problems were in getting data from distributed resources. And, this was greater challenge to him than the problem of visualizing it.*

- *Analyst at an HR firm mentioned that data cubes are a very useful tool for performing arduous analysis but he kept getting lost once he drilled down deeper and deeper into it.*
- *John Dalton in the Forrester Research Technology report quotes that applications for complex problem solving are hideously difficult to use and usability enhancements for such systems are no better than putting makeup on a pig.*

Software support for complex problem solving needs to cater to uncertainty and exploration. While problem solvers conduct their investigations, goals, deployed methods and their strategies keep evolving and changing on the fly. For software to be truly useful in this regard, usefulness needs to be ingrained into all aspects of product development and not just the interface. Its architecture, interface, modules, features and functionality should be such that it endows users with flexibility and adaptability for their emergent and dynamic inquiries [Mirel, 2003]. Inventing novel design paradigms to support for such control and flexibility will be in vain if software architecture has not been planned for.

A flexible software architecture is required to support a novel and flexible user interface for complex problem solving.

Previously designers were predominantly concerned with understanding and supporting tasks, which people perform to achieve clear predetermined goals. Issues related to why an individual carries out a task and what does it mean to the individual were typically placed outside the scope. Now, interactive technologies have become a part of our day to day lives and only focusing on tasks has proven to be insufficient. Understanding and designing technologies in the context of purposeful meaningful activities is now a central concern for design research and practices [Kaptelinin, 2015]. [Henderson and Kyng, 1992] and [Won et al., 2006] describe the need for flexible systems and personalized work flows in enterprise environments. End User development (EUD) [Lieberman et al., 2006] emphasizes on providing composable features at runtime to endow end users with the ability to tailor a system to their personalized needs. *Design during use* has been pillar of EUD, the idea of creating static IT artifacts has still not been challenged. The idea of packaged design

Designers should develop systems that enable design by end users at runtime.

needs to be overcome with the aim to surpass the need for enabling a system for a concentrated scenario and provide seamless transition between applications.

TUX aims to support users in self defined workflows.

Transformative User Experience (TUX) is a novel approach that aims to natively support users in a variety of spontaneously self-defined task flows, not limiting them to work along highly tailored use cases or adopt predefined patterns of guided procedures. It build on concepts of End User Development, Meta Design and appropriation. It introduces concepts of task objects and task containers. Task objects are data items that are of interest to the user. Task containers are defined as contexts that host task objects and localize appearance of task objects to fit the context on the fly. These containers are underspecified functionalities which allow users to freely express their intent and appropriate the container for their specific goal. Users can use multiple such task containers to compose their own problem solving environment. Thus, the user's task flow is not predefined. It's goal is to overcome common application boundaries enabling users to interact with information in terms of task objects within dedicated contextual task environments assembled through interrelated sets of task containers. TUX is all about situational adaptation of an object as mediated by technical containers which represents certain task semantics.

Difference between elasticity and tailorability is that of water and ice.

Concepts of TUX go beyond tailorability or flexibility or user-driven adaptation in the conventional sense and introduces *elasticity* as a *generic quality of use*. The elasticity of TUX can help establish continuum between generic and purposed application by bringing together reusable platform components in a very organic way while ensuring continuity regarding the user and task objects. Tailorability requires user interface to be explicitly composed or shuffled to fit to a certain form whereas elasticity enables intrinsic adaptation due to its free-form character. TUX goes beyond packaged design. Task context are often not mechanic but grow organically as required in a given situation. TUX therefore proposes designing IT-artifacts that are elastic with respect to their meaning and user interface.

1.1 Contributions

The contributions of this thesis are as follows:

- We propose a detailed interaction design of generic applications built using elastic design principles as proposed in TUX to support users in their self defined, emergent and exploratory investigations.
- We implement a highly flexible software architecture based on web-components¹ and realize generic software support for complex problem solving. We tailor the system to showcase it in a Bayesian Analysis scenario.

1.2 Outline

The thesis is structured as follows:

- **Chapter 2.** In this chapter we present relevant work that focuses on complex problem solving and approaches for providing flexible support for it.
- **Chapter 3.** We elaborate on the principles and proposed system architectures of Transformative User Experience design.
- **Chapter 4.** Here we look into theory and steps involved in performing Bayesian Analysis.
- **Chapter 5.** In this chapter we distill system and design requirements and propose a detailed interaction design for such a system.
- **Chapter 6.** Here we discuss a technical framework based on web components and Google Polymer library² to achieve elasticity as proposed by the TUX approach.

¹<https://www.webcomponents.org/>

²<https://www.polymer-project.org/1.0/>

- **Chapter 7.** We describe a study to evaluate the exploratory power and usability of our system.
- **Chapter 8.** This chapter concludes the thesis with a summary of the thesis work and future work.

Chapter 2

Related Work

“Certainly, we cannot say that computers have made us more ‘wise’ but the interactions computers offer do give us more chances to communicate our thoughts and build wisdom if we only knew how to...”

—Nathan Shedroff [Shedroff, 1997]

We have come a long way with technological and sociological advancement since Shedroff quoted this. Our approach is primarily based on Transformative User Experience design. TUX is a novel conceptual framework and so is its realization in the scope of this thesis. There hasn't been much research in realizing TUX principles. But, there have been several research contributions in field of complex problem solving, End User Development, Meta Design and Activity Theory. In this chapter, we will look deeper into these topics and use these insights to later distill requirements and design implications.

2.1 Complex Problem Solving

Although, the term *complexity* has been used extensively in the field of psychology and has been put to use in

different methods and ways, there are no cognitive theories for it. The way researchers have defined *problem solving* has changed over time to reflect their diverse interests. Given this, it together makes it hard to define *complex problem solving*. So, what is *complex problem solving*?

Horst Willhelm Jakob Rittel, a design theorist and professor of Design Methodology at the Ulm School of Design (Germany), is best known for inventing the term *wicked problems*. He defines them as :

Definition:
Wicked Problems

WICKED PROBLEMS:

Problems that are impossible or difficult to solve because of its contradicting, constantly changing and incomplete requirements, that are often difficult to recognize. And, due to complex interdependencies, the pursuit to resolving one aspect of a wicked problem may unveil or fabricate other problems.[Australian Public Service Commission et al., 2012]

The term *wicked* is used to denote resistance to a solution, rather being evil [Churchman, 1967]. He is renowned for identifying characteristics that differentiate well-structured tasks from complex tasks. In this thesis, I accommodate his interpretation to design useful software support for complex tasks.

2.1.1 Characteristics of Complex Problem Solving

Uncertainty
characterizes wicked
problems.

Two important aspects of *wicked problems* are its multi-contextual inquiries and uncertainties. This is mainly due to large amount information, incomplete and insufficient data, changing variables with emerging insights, goals evolve with evolving conditions and discovering unforeseen insights. During such inquires problem solvers often are unsure of the effects of their choices. In complex problem solving *uncertainty* prevails. Such *uncertainty* demands special attention for designing useful software support.

People whose work is mostly powered by standardized tasks and practices or automated processes do not solve wicked problems, thus have very little need for altering routines. Complex problem solvers may require automated and standardized tasks and processes from time to time, but at some point while conducting complex investigations they need bring their own *insight, ingenuity and intent* to shape data, tasks, methods, processes and strategies. Sometimes, complex tasks may use mostly standardized and automated processes and have definite right answers but are complex because of the volume and complexity of data and or have numerous permutations or possibilities to explore. These *characteristics of complex problem solving* imbibed in multiple contexts fuel the strategies that problem solvers follow. These traits distinguish where *well-structured tasks* make way for *complex tasks*.

Well structured tasks have very little requirements for alteration. Complex problems require people's insight, ingenuity and intent.

Characteristics of complex problem solving as adapted from Horst Rittel are as follows [Rittel, 1984] [Australian Public Service Commission et al., 2012] [Mirel, 2003]:

- *Ill-defined situations and goals.* Problem solvers often find it difficult to define the problem. They devote significant time iterating and re-formulating the problem statements throughout their investigations finding an optimal level of abstraction. Also, each stakeholder has his or her own version of the problem statement. None of these versions is wrong, but a problem solver needs to account for these needs while defining the problem.
- *Complex heterogeneous data from diverse sources.* Problem solvers use large volumes of data while solving a problem. They need to collect, clean, transform and integrate data from numerous sources to comprehensively examine a situation. Integrating data from diverse sources is a complex task. Problem solvers need to overcome software and infrastructure inadequacies to be able collect and work with such data. They need to transform and re-arrange data as and when required on the fly to derive new views that cannot be foreseen.

Difficult to define the problem form multiple perspectives. Evolving problem statements leads to evolving goals.

Problem solving may involve using unstructured or non-compatible data from multiple sources.

- Problem solvers move along self-directed transformative vectors and as-such way-finding occurs. [Latzina and Beringer, 2015]
- As problem solvers explore the problem landscape more and more, with new insights new questions arise.
- Evolving uncertainties lead to evolving goals and eventually multiple different solutions.
- *No predefined entry or stopping points.* Problem solvers decide on an entry point to start their inquiry based on how they formulate the problem, what data and information is available to them, how they are oriented towards problem solving, and what has been their previous experience and knowledge they have gathered from similar problem solving. This entry point sets them on a specific path. In due course of their investigations, contextual situations deploy new influences, new insights emerge and which many a times changes their course. Sometimes, they may need to start all over again and decide on a new entry point. Similarly, there is no clearly defined stopping point. It is a subjective decision for them to infer whether they have investigated enough and stop exploring. Based on emerging *contextual dynamics* people have to plan their way in and out of problem solving.
 - *Emergent and Dynamic.* With evolving contextual dependencies change is unavoidable. Goals emerge, insights surface, constraints change, stakeholders requirements change, new data or information becomes available, and with this problem solvers reevaluate their inquiries constantly. Sometimes with new insights new unforeseen problems surface which restarts the circle of inquiry.
 - *Good enough solutions with no answer.* Dynamic and emergent queries allow for multiple possible solutions. The correctness of the solution depends on the situation. Its is important the problem solver comprehends the consequences of their candidate solutions. They need to support and present their solutions with evidence and prove its validity. Problem solvers continuously try to reduce uncertainties to find a solution, while being aware of the various entangled conditions that may create larger uncertainties.
 - *Iterative and Opportunistic with socially based patterns of inquiry.* As discussed before, the way in which problem solvers structure their inquiry depends on how they socially orient themselves to problem

solving with data that is available to them. What problem solvers do as they progress through their inquiries depends on earlier choices for selecting, arranging, coordinating, and relating relevant factors. Many renditions are possible, and the ones that problem solvers decide to compose depend on contextual demands. The less systematic a problem solver's way-finding patterns are, the more dynamic the processes of inquiry will be.

- *Attached to interests of various stakeholders.* For complex problems embedded within a larger organizational environment, numerous stakeholders are involved in defining the problem and its solution. Choices that a problem solver makes is influenced by the biases of each stakeholder. Needs of a stakeholder often change and new stakeholders may come in due course. Their diverse and often competing agendas make it necessary for problem solvers to search for alternatives and weigh trade-offs.

Emergent insights open new opportunities allowing problem solvers to iterate a situation with newly gained insights

Different stakeholders have different perspectives and needs.

It is essential that we analyze various tasks and strategies problems solvers deploy to achieve their goals. We need to conceptualize and model their patterns of investigation. We need to distill core generic activities that problem solvers perform and compose their task landscape.

2.1.2 Core Activities

People perform investigations that are guided by domain specific practices and needs. Here we list activities that can be generalized across different problems. They generalize across problems as they originate from the 2.1.1 "Characteristics of Complex Problem Solving". These activities are as follows :

- *Data-ordeals.* This refers to dealing with large volumes of multi-dimensional data from multiple heterogeneous sources. This includes validating data accuracy, integrating and arranging data from

- Dealing with unstructured and non compatible data sets.
- multiple sources, comparing data sets and filtering away irrelevant data. After this is done problem solvers need to extract valuable insight from the data and visualize it with right level of abstraction.
- People need to align themselves in line with the problem. They need to know what stage of problem solving they are currently in and how to get to the answer.
- *Wayfinding*. Wayfinding in general covers the ways in which people orient themselves in a physical space and move around. We use spatial clues like signage, indoor maps, lighting, color coding etc. to navigate inside a building. These features are information-support systems for wayfinding¹. In our case it can be defined as spatial problem solving. People use their visuospatial memory to orient themselves in space. It allows problem solvers to structure their way through inquires from indeterminacy to a solution. They go through multiple intersections, choices, tangents, collateral streams of reasoning and backtracking. As new information and insights emerge, problem solvers follow the *scent* of potentially important insights. There are four stages to basic wayfinding [Lidwell et al., 2010]:
 - *Orientation*. Is the attempt to determine one’s location, in relation to objects that may be nearby and the desired destination.
 - *Route decision*. Is the selection of a course of direction to the destination.
 - *Route monitoring*. is checking to make sure that the selected route is heading towards to the destination.
 - *Destination recognition*. Is when the destination is recognized.
 - *Sensemaking*. This refers to a problem solvers mental processes of finding relationships among data and situations to draw inferences. Problem solvers compare data sets, distributions and trends in data sets to unearth insights that help them make sense of data. These inferences and insights help problem solvers relate what they see on the display with their own intentions.
- Sensemaking is an ongoing process by which people give meaning to their actions and experiences.

¹<http://www.umich.edu/~wayfind/supplements/moreinfomain.htm>

Task is the basic component of human work. People use various tools to transform objects in their hands into shapes they desire. Tasks are actions which are performed by one or more actors to achieve a particular goal. Each task in the work process is regarded as a context-bounded activity that is directed to achieve a goal of the task under given conditions. A continuum of possible tasks can be thought of as skill-based tasks from one side and complex problem-solving tasks from the other side. Skill-based tasks are performed in a brisk automated way with minimum attention. Problem-solving tasks could be divided into two major groups: non-algorithmic and algorithmic. Algorithmic tasks are executed in accordance to some rules and logic. Complex tasks are non-algorithmic. Useful software support should cater to the above mentioned characteristics of complex problem solving [Bedny, 2014].

Earlier designers were mainly concerned with comprehending and assisting tasks, which users perform to achieve their preconceived goals. The issues of why an individual carries out a task and what does it mean to the individual were typically placed outside the scope. Now, interactive digital technologies have become a part of our everyday lives and just focusing on tasks has proven to be insufficient. Understanding and designing technologies in the context of purposeful meaningful activities is now a central concern for design research and practices.

We have a fair understanding of what activities are. Due to this there can be many different theories from person to person and may not be specific enough. How to distinguish activities from non-activities? Can activities be broken down into smaller units? What role does technology play in human activity? To answer these and other similar questions HCI needs a more elaborated concept of activity. Such concept is offered by activity theory [Kaptelinin, 2015].

2.2 Activity Theory

[Kaptelinin and Nardi, 2006]

Activity as interpreted by activity theory is the interaction of a person/actor with the world. The interaction is described as a process relating the subject (S) and the object (O), which is commonly represented as :

S <-> O

Activities and their subjects mutually determine one another. Activities are generative forces that transform both subjects and objects. It is instantly understandable that activities are influenced by the attributes of subjects and objects. But, the reverse is also true, i.e. subject and the objects evolve. Subjects do not only express themselves in their activities; in a very real sense they are produced by the activities .

Our world is structured; it comprises of discrete entities, i.e. objects. Users' interaction with the world is also structured being organized around these objects. Every object has it's objective meaning, determined by it's relationship with other entities existing in it's surroundings (including the subject). In order to meet their needs, subjects have to reveal the objective meaning of the objects, at least partly, and act accordingly. An object in an activity is dynamically aligned in the unfolding S-O interaction. The alignment involves a double transition: the subject's activity is subordinated to properties of the object which gives rise to new activity structures; in turn, new activity structures bring about new subjective phenomena, such as a more developed image of the object. In an activity system such subject-object interactions and object transformations are mediated via tools.

Subject and objects evolve as they interact more and more with each other. As and when the subject or the objects evolves so does their usage and the tools required for their mediation. Subjects may choose to modify the tool at hand or start using a new tool to further interact with the object.

2.3 End User Development

[Lieberman et al., 2006]

Current professional, leisure and learning environments are characterized by evolving work and business practices, diverse qualifications and individualized preferences in a dynamic environment. This diversity derives from people with different cultural and educational backgrounds, skill, knowledge, psychological and cognitive abilities and diversity among tasks, contexts and are of work. User centered and participatory design is just a part of the solution. Given that user requirements are evolving, diverse, and at times difficult to pinpoint and interpret accurately, following conventional design and development iterations to keep up with evolving contexts would be too slow, time consuming, and expensive. The challenge here is to develop environments that empower users to develop, modify and adapt software applications to a level of complexity that is appropriate to their individual skills and situations. End User Development (EUD) is paradigm that advocates for such flexibility. Libermann et al. define End User Development as follows:

END USER DEVELOPMENT:

"EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact." [Lieberman et al., 2006]

Definition:
*End User
Development*

They identify two types of EUD activities from a user-centered design perspective :

- *Parameterization or customization.* These are activities that allows users to choose from among alternate interaction or presentation mechanisms provided by the application. In adaptive systems such customization occurs algorithmically by the system as response to user's activities. Examples of such activities are :

- *Annotation.* Users make a note beside results and data to recall how they reached the result and how could they reproduce it.
- *Parameterization.* The user wishes to guide the system by indicating how to handle data in a different way; it could simply be applying different program functionalities or associating specific computation parameters to specific parts of the data.
- *Program creation and modification.* These are activities where users create from scratch or modify an existing software artifact. Artifacts created by end users could be objects describing a control sequence or an automated behavior, such as database requests or grammar rules which can be described via approaches like visual programming, programming by demonstration, writing and generating macros and using scripting languages [Costabile et al., 2006].

System should allow for different levels of modifications with increasing complexity and expressiveness that go beyond just annotation and parameterization, while being easier than re-programming. For example a system could offer 3 levels of complexity [Henderson and Kyng, 1992]:

- *Users can set parameters.*
- *Users might compose existing components.*
- *Users can extend the system by programming new components.*

Software design becomes outdated soon with changing and evolving requirements. Challenging this packaged design methodology of *design before use*, new approaches go beyond packaged design to establish *design during use*. Such systems changes can result due to explicit end-user actions or system initiated state transition [Mehandjiev and Bottaci, 1996].

Modular component based approaches enable reconfiguration and decomposition of software artifacts

that are themselves build up from smaller components. A system's component architecture should be designed to be meaningful for its users, so that they can correlate evolving requirements in their working domain to corresponding changes in the system's component domain. System adaptation should be unobtrusive i.e. not distracting users from their main task and cognitive load of switching from using the system to adapting the system so be low. Flexible software architecture is prerequisite for enabling adaptivity. Approaches range from changing parameters, rules, and constraints to changeable descriptions of system behavior and component-based architectures [Won et al., 2006]. A key property of an EUD-friendly architecture is to allow for substantive changes during run-time, without having to stop and restart or rebuild the system.

We need to to move past the binary choice of low level domain unspecific interactive programming and over specialized systems. These are the two end points on a spectrum:

- *Turing tarpit.* They are capable of representing any problem that computers can be used to solve, and as open systems they let users change any aspect of the system if necessary but, they provide an incorrect level of representation for most problems. Expressing a problem and designing a solution in these systems requires creating a mapping from the context of the problem to the core constructs provided by the programming language and its supporting library [Shaw, 1989]. They are powerful but are difficult to learn. Eg: Interactive programming environments like Smalltalk, Squeak, Logo, Lisp, etc.

"Beware of the Turing tarpit, in which everything is possible, but nothing of interest is easy." [Perlis, 1982]

- *Inverse of Turing Tar Pit.* These are easy to use no special training is required. They are domain specific and closed systems that do not allow for user modifications.

"Beware of over-specialized systems, where operations are easy, but little of interest is possible" [Guindon, 2013]

Meta design [Fischer and Giaccardi, 2006] is a framework for end user development based on the Seeding, Evolutionary Growth, Reseeding Model, (SER) a process model for large evolving design artifacts. One has to move beyond the binary choice of low-level domain-unspecific interactive programming environments and over-specialized systems. To address for evolving requirements at use-time, software needs to be *underdesigned* at design time. *Underdesign* advocates for developing environments rather than solutions. Under specifying the interface of an application allows users to assign different semantics of use artifacts at use time. Microsoft Excel is underspecified in terms of what a spreadsheet application usage. It can be used as an address book, to plan budget, managing minutes of meeting etc.

Chapter 3

Transformative User Experience Design

[Latzina and Beringer, 2012] [Latzina and Beringer, 2015]

“We have only scratched the surface of what would be possible if end users could freely program their own applications...”

—Bonnie A. Nardi [Nardi, 1993]

Transformative User Experience (TUX) design was proposed by Markus Latzina and Joerg Beringer in 2012 based on the concepts of *end-user development*, *meta-design* and *appropriation*. TUX is a novel approach that aims at natively supporting users in a variety of spontaneously self-defined task flows, not binding them to work along highly tailored use cases or adopt to predefined patterns of guided procedures. It proposes that user interface for next generation business application must be able to seamlessly support users in all stages of task accomplishment and switching between all these stages, routine and non-routine work. TUX calls this quality of an interface of innately accommodating the ongoing task needs of the user during use time as *elasticity*.

It's goal is to overcome common application boundaries

enabling users to interact with information in terms of task objects within dedicated contextual task environments assembled through interrelated sets of task containers. An example that mimics good TUX is:

A user may search on the web for a restaurant with a particular cuisine. The system responds with a list of suggestions. These suggestions are presented with information like name, phone number, average cost of a meal, opening times, customer reviews and an address. After shortlisting a restaurant from the list, user clicks on the location to navigate to the restaurant. User's seamless transition from the task context of searching for a restaurant to the context of navigating to the restaurant creates a transformative experience.

TUX is all about situational adaptation of an object as mediated by technical containers which represents certain task semantics. It requires a clear presence of task models in a product's runtime and not just at design time. To allow users to create a proper task setting, a system must provide mechanisms to detach application content and move it through different task containers.

3.1 TUX proposed system architecture

TUX proposes a system architecture of how to achieve contextualization and elasticity during runtime. This is based on two basic concepts of *task object* and *task context*.

Definition:
Task object

TASK OBJECT:

"A task object which acts as a proxy to a system object and provides functionality for the hosting container to contextualize its appearance and behavior to match the local semantics of the container." [Latzina and Beringer, 2015]

Paul Dourish explains the difference between system objects and task objects in Placeless documents [Dourish, 2003] with an example : A generic artifact in the form of a document produced by a text processing application is a

system object, but the use of the document by its owner in a working context may be as a project plan, contract, minutes of meeting, bug tracker, requirement specification, patent application etc. While Dourish describes this as appropriation. Appropriation in such system happens in an unmanaged manner. TUX realizes appropriation by moving task objects along task contexts rather than leaving it unmanaged. The container moderated adaptation is facilitated by the system, appropriation happens within the system rather than being imposed from outside by users.

TASK CONTEXT:

"It is a container which is hosting and displaying task objects...We refer to it in terms of a programming model in which the container class is a storage container for loosely-coupled content and functions. Contextual appropriation of system objects is realized by allowing containers to adjust system objects to task objects reflecting the local semantics of use... A container models context that is used to impose situational semantics on system objects by casting them into local task objects with local appearance and behavior." [Latzina and Beringer, 2015]

Definition:
Task context

3.1.1 Task Context

Containers embody and shape the context which imposes situational semantics on system objects by casting them into localized task objects. A container may have a pre-specified context of use with a predefined purpose or it may evolve over a period of time and change its semantics during runtime. A container can be appropriated as a task object by embedding it in another container. Elasticity in containers is characterized by:

- *Elastic Purpose.* When the context of use is underspecified the resulting system is elastic. The lesser the context is rigidly defined, the more degree of freedom a user has for casting his intentions onto the system. Underspecified design is applicable for

generic applications and is not constrained by any application domain. "They are designed for very abstract needs, but agnostic to any specific content [Latzina and Beringer, 2015]." For appropriation to happen a system need to be open for interpretation [Dix, 2007]. Over a period of time, users shape a purposeful environment by bringing in more and more content or adding tools to a given environment. There is a bargain between affordances and under specified design. If no specific purpose is informed by design itself the generic functionality must be compelling enough.

- *Elastic Collaboration.* Collaboration services tend be specialized for limited use cases. This pushes users to decide beforehand on technologies to use for their subsequent task flows. Rigid barriers force users to rebuild the task context withing a collaboration platform to be able to share it with others. User stories reflect the need to move into collaborative mode from time to time but there is no strict task flow for it. The need of collaboration cannot be defined beforehand at design time. Users should be able to move content in the form of task objects into a collaborative space i.e from a personal work context to a shared context without having to re-build the content just for the purpose of collaboration.
- *Elastic Practices.* Task flows do not follow a predefined life cycle and they alternate between regular routine and non routine situation. For non routine tasks the system should offer generic functions so that users could define their intent. For routine well defined tasks there should be a set of exclusive functions. The system should support the user while moving along different levels of problem solving.

3.1.2 Task Object

TUX separates the identity of an object from its use in a given task context. Containers add behavior to a task object and tailors the view of an object. Task objects are indicative of the principle of *contextual polymorphism*

since they can adapt semantically in various ways to their respective context.

Task contexts are implicitly or explicitly created as a result of moving task objects from one container to the other. A combination of task objects and container semantics shape a task context. TUX defines multiple mechanisms for adapting task objects to containers.

- *Contextual Casting.* Any system object could be casted to another class to model the behavior of a task object whose semantics are local to the container instead of the initial system object. TUX framework should support abstracting system objects to provide a generic task object class with a handle to its identity and behavior.
- *Contextual Postures.* It cannot be assumed that a task container knows about each and every object, still it should be possible for a container to moderate the appearance of a task object. This may be achieved by facilitating an abstract task object with a set of contextual appearances. The hosting context can select from among these postures for rendering it. This contextualization of system objects corresponds to that of late binding.
- *Contextual Volatility.* This refers to freezing or snapshotting an object when it is added to a container. This snapshot can be later used to edit or create a new version of the task object. This may be important in cases where the task object needs to be a tangible artifact local to a context.

Let's look at a simple example:

Anna, a manager in a construction company is looking for qualified plywood vendors who could satisfy her demands. While searching for vendors that could possibly fit her criteria, she would first have to go through a listing of all vendors. Here the search results for plywood vendors is presented in terms of a list, with each vendor as a list-item. In TUX terminology, vendors are system objects and the result list is a task specific container and

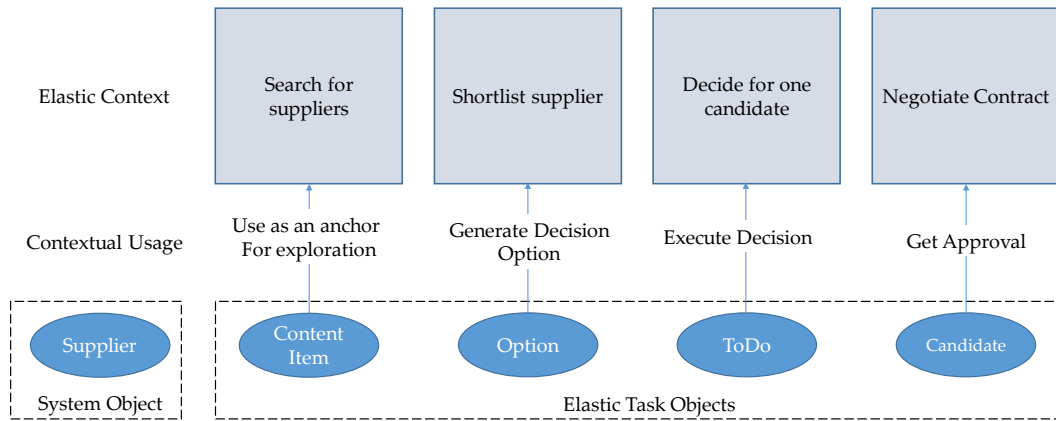


Figure 3.1: Contextual Consumption of Business Entities adapted from [Latzina and Beringer, 2015]

the result list-items are the task objects. In this result list each list-item is displayed or presented with sufficient information to probe, distill candidate vendors. At this point Anna would expect to have abilities like comparing and shortlisting vendors, inspecting each vendor individually etc. These needs arise from the context. These items now become a member of a shortlist of candidates. These members of the shortlist now inherit behaviors of a shortlisting activity like decision options, which can be prioritized or ranked and assessed. This shortlist is now a new container which represents such a shortlisting activity. The fact that each item is a plywood vendor is now less important and users vote and annotate each item to support decision making. For shortlisted candidates more investigation could be planned by putting them in a TODO list. Now, each plywood supplier is an action item i.e task object in a TODO context i.e. container. After due investigation a specific supplier is chosen which has to be approved by a stakeholder.

Through this entire example it is secondary that the plywood supplier is actually an ERP Object. Figure 3.1 illustrates how the supplier system object gets transformed into different forms based on the context of usage.

Chapter 4

Bayesian Analysis Theory

[Kruschke, 2014]

*“How often have I said to you that when you
have eliminated the impossible, whatever remains,
however improbable, must be the truth?”*

—*Sherlock Holmes [Doyle, 2010], [Kruschke, 2014]*

In this chapter we will briefly discuss the concepts of Bayesian analysis. In the scope of this thesis we focus on performing very basic rudimentary Bayesian analysis restricted to calculating posterior distribution from priors and likelihood using the Bayes rule. Bayesian data Analysis is based on two basic conceptions. Firstly, bayesian reallocates credibility across different possible outcomes. Certain outcomes have different prior possibilities as and when new data is gathered, these credibilities are redistributed across the different outcomes and the result is a new credibility distribution. Secondly, these possibilities are nothing but parameter values mathematical formulas describing trends in data.

4.1 Theory

Kruschke describes Bayesian analysis with the following example from our everyday lives:

Suppose you step outside your house one morning and notice that your front porch is wet, and start to wonder what could be the reason for it. You start thinking and first list down all the possible causes of wetness, including possibilities such as rain, garden irrigation, a leaking pipe, a newly erupted underground spring, etc. If all you know is that the porch is wet, then all those possibilities will have some prior credibility based on previous knowledge. For example, recent rain may have greater prior probability than a spilled drink. You now continue to make more observations, you gather more data. If you observe that the pavement is wet for as far as you can see, and so are the trees, then you reallocate credibility to the hypothesis of recent rain. Inversely, if you observe that the wetness is localised to a small area, and there is an empty cup lying on the ground, then you would reallocate credibility to the hypothesis of a spilled drink, even though it had a lower prior probability.

Let us discuss three important aspects of Bayesian analysis in the scope of this thesis:

- **Prior.** In Bayesian reasoning, prior is a probability distribution which expresses a person's belief about an uncertain quantity. In the example above, when we had no clue what caused the wetness, our prior belief was equally distributed across the candidate reasons for wetness. So, the possibility of each outcome was 0.25. Here the reason for wetness is the unknown quantity. A prior can be created in a number of ways. A prior could be determined using information from previously collected experiments. A prior can be elicited by just pure subjective opinion.
- **Likelihood.** Likelihood is also a probability distribution which is used to describe data. It is a function of parameters of a statistical model for a given data set. In the scope of this thesis we do not expect our users to know this concept

in depth. We handle calculating the Likelihood distribution autonomously by our system. This distribution can be calculated based on common experimental design. Depending on the amount and types of independent and dependent variables a representative model and likelihood distribution can be calculated autonomously.

- **Posterior.** Posterior is a probability distribution which is obtained by applying Bayes Rule to likelihood and posterior distributions. Bayes Rule in the discrete form, allows us to calculate the probability of an event A given event B. This is what is called a conditional probability. The posterior can be written in a memorable format as:

$$\text{Posterior distribution} \propto \text{Likelihood} \times \text{Prior probability}$$

The posterior probability distributions gives the final result as probabilities of possible outcomes.

4.2 Performing Bayesian analysis with a simple example

Let us take a simple hypothetical example to explain this:

Consider a study which tests the effectiveness of a new hair loss control pill. 20 people were assigned to the treatment group and 20 to the control group. 4 people from the treatment group still had hair loss compared to 16 from the control group. The question being asked is how strongly does this indicate that treatment is more effective than control?

To simplify matters, We convert this problem of comparing two proportions to a single proportions problem. Consider the 20 total people who still had hair loss after taking the medication, and ask how likely is it that 4 people with hair loss come from the treatment group. If the treatment and control are equally effective and the sample sizes for the two groups are the same, then the probability that a person with hair loss came from the treatment group is simply 0.5.

We first start by setting up our hypothesis. We can think of these as the models that the data come from. We know that the probability that a person with hair loss comes from the treatment group can take values between zero to one. Let us consider that the plausible chances that a person with hair loss comes from the treatment group comes from is 10% or 20% or 30% upto 90%. Hence we consider 9 models instead of one.

Next, we need go onto specifying the prior probabilities we want to assign to these models. The prior probabilities we assign must reflect our beliefs before the experiment had been conducted. They should incorporate the information, learn from all relevant research up to the current point in time. However, the prior probability should not incorporate information from the current experiment. Suppose our prior probabilities for each of the models are presented in this table.

Model(p)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Prior	0.06	0.06	0.06	0.06	0.52	0.06	0.06	0.06	0.06

We have placed a prior probability of 52% at $p=0.5$ and equally distributed the remaining probability amongst all the other models. Such an equal distribution i.e. symmetrical distribution around $p=0.5$ implies that that the treatment is equally likely to be better or worse than the standard treatment. A peak value of 52% at $p=0.5$ implies that we believe that there is a 52% chance that there is no difference between the treatment and condition group. Here, we do not get into the details of how to come up with values for priors.

Next we can look at calculating the likelihood distribution. Likelihood is defined as the probability of data given the model. Here our data is 4 people with hair loss from the treatment group out of a total of 20 people with hair loss. And our model is the same as defined above from 0.1 to 0.9. As we discussed previously likelihood function can be calculate based on experimental design. In our case it is a

binomial distribution representing 4 successes in 20 trials.
The likelihood calculated is as follows :

Model(p)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Likelihood	0.09	0.22	0.13	0.035	0.0045	0.0003	0	0	0

Once this is done and the models have been described for priors and likelihoods, we can use Bayes rule to calculate the posterior probability distribution. The posterior distribution obtained is as follows:

Model(p)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Posterior	0.17	0.43	0.25	0.07	0.08	0.0	0	0	0

In the result we see that the posterior probability has its maximum value $p=0.2$. So, this model is the most likely model, based on the observed data. The posterior probability at p is equal to 0.2 is 42.48%. The calculation of the posterior incorporated prior information and likelihood of the data observed and the concept of data. Probability that $p=0.5$, dropped from fifty two percent in the prior to about seven percent in the posterior. This shows how we update our beliefs based on observed data using Bayesian reasoning.

So coming back to our initial question is treatment more effective than the condition?. Sum of probabilities in the posterior distribution for p lesser than 0.5 gives us 92 percent. There is a 92 percent chance that the treatment is more effective than the control. Bayesian allows us to make such direct probability statements about our models.

4.3 Workflow

According to Kruschke, typical Bayesian analysis involves the following five steps:

- Identification of independent and dependent variables in the relevant data set. Determining the measurement scale for the variables.
- Defining the mathematical form that describes the given data with meaningful parameters. In this thesis we calculate this on the basis of experimental design inferred from the previous step.
- Assigning a prior distribution that can be supported using relevant justification.
- Re-allocate credibility distribution across the parameter values based on Bayesian inference. The new posterior distribution should be then interpreted based on meaningful parameters assuming the model adequately describes the data.
- Conduct a posterior predictive check that verifies whether the posterior distribution models the given data accurately.

An additional step sometimes could be collecting, cleaning and refactoring data before it can be used.

Chapter 5

Interaction Design

In the previous chapters we discussed concepts of Complex Problem Solving, end user development, activity theory, transformative user experience design and Bayesian analysis. Based on these concepts we first design an interface for generic problem solving and then appropriate it for Bayesian inference making. Before we elaborate on the design we try to filter design requirements and conceptualize the design.

5.1 Design Requirements and Conception

It is a common practice in user centered design to design for user's activity in a given context. But such a singular usage and conception of the term '*context*' and focusing on a generic context is misleading. While working with complex problems and performing emergent queries users work in a large number of contexts. Singular association of context is inadequate while designing systems for complex problem solving [Mirel, 2003].

Designing for a single or generic context of use is inadequate.

People act in multiple contexts and may require multiple such applications in order to achieve their goals. When people are in problem solving mode they perform tasks at a higher level. "Higher-order processes give coherence

<p>Problem solvers operate on higher level tasks in problem solving mode.</p>	<p>to the synergy of open-ended investigations. If a software focuses only on actions for unit tasks without unifying them into these higher-order processes, problem solvers will not be able to shape to their own ends the emergence, serendipity, idiosyncrasy, and uncertainty that come with solving perplexing problems. [Mirel, 2003]" Rasmussen et al. describes the need of smart instruments that are specialized for a specific type of task under a context for complex problem solving [Rasmussen and Vicente, 1990].</p>
<p>We conceptualize TUX containers as independent generic applications.</p>	<p>With this lens we look at TUX. The two basic concepts of TUX are <i>task objects</i> and <i>task contexts</i>. TUX defines task contexts a.k.a containers as smart instruments. Keeping this in mind we conceptualize these technical containers as separate applications. Each application represents a <i>higher order process</i> i.e. generic activity and behaves like a smart instrument. Examples of such generic activities are :</p> <ul style="list-style-type: none"> • Grouping • Comparing • Associating • Clustering • Visualizing • Keeping • Network Building • Searching • etc...
<p>Applications needs to be underspecified to be open to interpretation.</p>	<p>These are generic containers and are not designed for the purpose of achieving a specific goal. They should be designed such that users can implicitly specify context of use and appropriate them in-situ. Applications need to be underspecified so that users have enough freedom to express their intentions and context of use. When the context of use is underspecified the resulting system is elastic. The lesser the context is rigidly defined, the more degree of freedom a user has for casting his intentions</p>

onto the system. Underspecified design is applicable for generic applications and is not constrained by any application domain. "They are designed for very abstract needs, but agnostic to any specific content [Latzina and Beringer, 2015]." Containers should be able to pick up these queues and apply transformations as suggested by TUX so that appropriation takes place inside the system in a managed way. Such systems are open for interpretations and appropriation can take place.

Underspecified applications allow users to express the context of use and appropriate it for their needs.

From our above discussion and concepts discussed in 2 "Related Work" and 3 "Transformative User Experience Design" we can narrow down the following system requirements:

- An application needs to be underspecified so that it can be appropriated and context can be specified. Complex problems have ill defined situations and goals 2.1.1 "Characteristics of Complex Problem Solving". Underspecified applications allow to express such goals and situations. This need for underspecification is also characterized by TUX as elastic purpose of task contexts in 3.1.1 "Task Context".
- Users in problem solving mode would require multiple such generic applications for achieving a goal. They may need only a subset of all such available generic applications offered by a system.
- Users need a workspace where they can manage and spawn such generic applications.
- User could add or remove an application from the workspace at runtime as queries in problem solving mode are emergent and dynamic 2.1.1 "Characteristics of Complex Problem Solving".
- User may want to clone an instance of an exiting application with its data items and configurations. This would allow users perform different variations of the same activity and try different stream of inquiry to support users in their iterative and opportunistic

TUX containers a.k.a applications need to underspecified.

Allow replicating applications with or without its data.

	queries 2.1.1 “Characteristics of Complex Problem Solving”.
Allow optional cloning for task objects.	<ul style="list-style-type: none"> • A user could use the abilities of a container and apply container or context specific transformations on objects or items of interest. • A user may want to move data items from one task container to another. While moving he may want move the item itself or drop a copy and maintain the original in source container.
Allow users to fetch data from multiple sources.	<ul style="list-style-type: none"> • User may want to load data into a container from a file or from any other data source as while solving complex queries users may want to fetch complex heterogeneous data from multiple sources 2.1.1 “Characteristics of Complex Problem Solving”. • Users in problem solving mode may approach a problem top down or bottom up manner of information processing and knowledge ordering. Users may decide in advance on the set of steps to performed to achieve their goals and create a plan. Alternatively, they could start their inquires from any random starting point and build recursively by adding more and more applications and end once they are satisfied with their investigation 2.1.1 “Characteristics of Complex Problem Solving”.
Support top-down and bottom-up inquiries.	<ul style="list-style-type: none"> • Users would need support to deal with data ordeals 2.1.2 “Core Activities”. Few higher order functions that could be useful in this respect are grouping, comparing, clustering, charting, visualizing etc. • Containers should be able to infer what properties of an object are relevant to be presented in the current context. Based on these properties it should also be able to choose an appropriate presentation of the object. This refers to properties of <i>contextual casting</i> and <i>contextual postures</i> of task objects. 3.1.2 “Task Object”
Build generic application support to deal with data-ordeals	<ul style="list-style-type: none"> • Continuing with the idea of contextual casting and postures, containers should be able to monitor user activities and sense a change in context. When this happens containers should be able to adjust or change
Containers should be able to change visualization of task objects on the fly.	
Containers should update context when new task objects arrive.	

the visualization of all containing objects to fit the context of use.

- There could be multiple suitable visualizations for objects in a given container. Users should be allowed to switch between these visualizations to be able to better analyze a situation. For example, given a data set there could be multiple suitable charts that could be useful for exploring data. Here the user should be allowed to switch between these data visualizations.

Allow users to switch between multiple visualizations of task objects in a container.

An interface for realizing the above requirements should continuously represent the object of interest and allow to transform these objects by moving them along different task containers which perform dynamic, incremental operations whose impact on the object is immediately visible. Though the concepts of TUX are not confined to any specific UI paradigm, TUX would hugely benefit from the core principles of Direct Manipulation Interfaces. TUX can leverage from physical, spatial and visual representations in Direct Manipulation interfaces making it easier to retain and transform objects. Users could immediately see if their actions are furthering their goals and if the actions become non-productive then they can easily change the direction of their activity. Direct Manipulations endows users with a sense of directness i.e. a feeling of close contact with the object and actions of interest [Shneiderman, 1993].

We choose Direct Manipulation Interfaces paradigm to support TUX.

5.2 Design

On the basis of system requirements listed in the previous section, we conceptualize our design. We use Polymer, an open source web-components library for implementation in this thesis. We use some default controls provided by polymer so, some of its visual design can be seen in our prototypes and implementation.

We visualize each application as a widget or a window. These applications are arranged as tiled windows in a

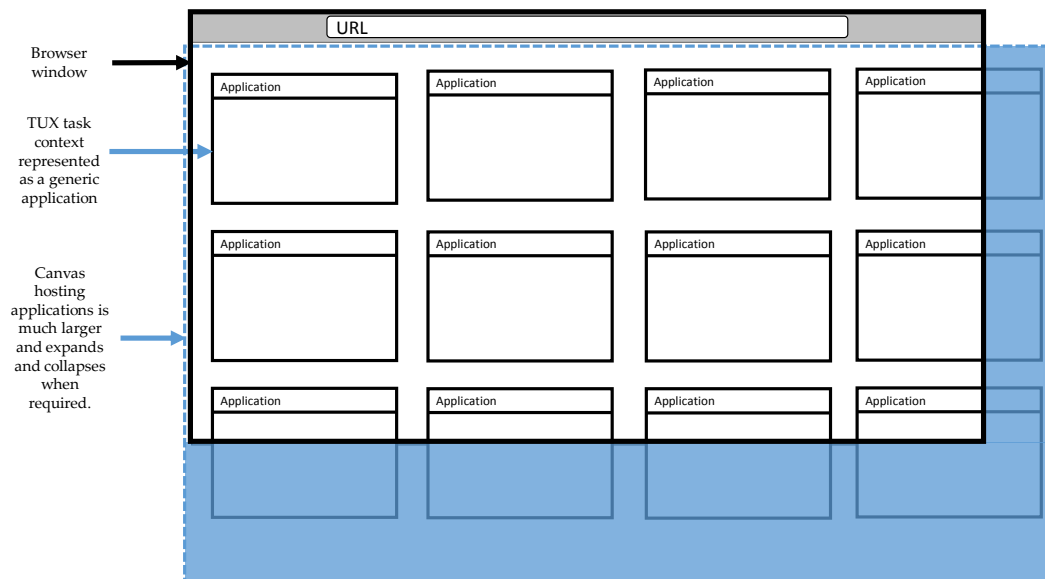


Figure 5.1: Prototype showing applications arranged on a canvas a.k.a workspace

Each application is visualized as a window on an infinite canvas.

workspace. This workspace is an infinitely expanding canvas so that it can accommodate multiple applications that users spawn at runtime. This allows users to distribute different tasks to different parts of the canvas enabling to make use of their visuospatial memory as shown in Figure 5.1.

Our goal is not to develop a highly effective zoom-able workspace.

Navigating through such an expanding canvas can be difficult. The canvas needs to be zoom-able so that users can get the right applications into their view port when required by adjusting the amount of zoom or moving the view port around. Other auxiliary aids like a mini-map or a fish eye view etc. can also be integrated to provide more sophisticated support for navigating a zoom-able interface. Users should be allowed to rearrange, sort and switch application on this canvas whenever desired. Multiple strategies can be developed to display and arrange containers based on related data items, data flow, temporal characteristics. There is a myriad of HCI research concerned with building effective zoom-able interfaces. We do not look into designing for efficient navigation or way-finding for such zoom-able canvases. In the scope of this thesis we focus on designing for elasticity.

5.2.1 TUX Container design

Each generic application as discussed above represents a TUX container and is visualized as a window or widget. All applications irrespective of its features and functionality have a common window decoration as follows:

- *A header* with
 - an application icon,
 - an application name,
 - a handle to signify druggability of the container,
 - a button to clone the application,
 - and a button remove the container from the canvas.
- *A footer* which is visualized as a tab bar. This tab bar allows users to switch between different visualizations offered by the TUX container.

This container design is also described in Figure 5.2(a).

Few generic applications a.k.a TUX containers that we selected to implement are:

- *Planning.*
- *Grouping.*
- *Comparing.*
- *Visualizing.*
- *Sourcing.*

All of these applications are generic applications and do not provide any indication that they could provide any indication that they could be used for Bayesian Analysis. Users should be able to appropriate containers to perform Bayesian Analysis. For the purpose of this thesis we build some domain specific support so that users can express their Bayesian needs and containers can interpret this context.

A windows decoration typically consists of a title bar, usually along the top of each window and a minimal border around the other three sides. ¹

Design of these applications do suggest usage for a specific use-case but rather can be appropriated.

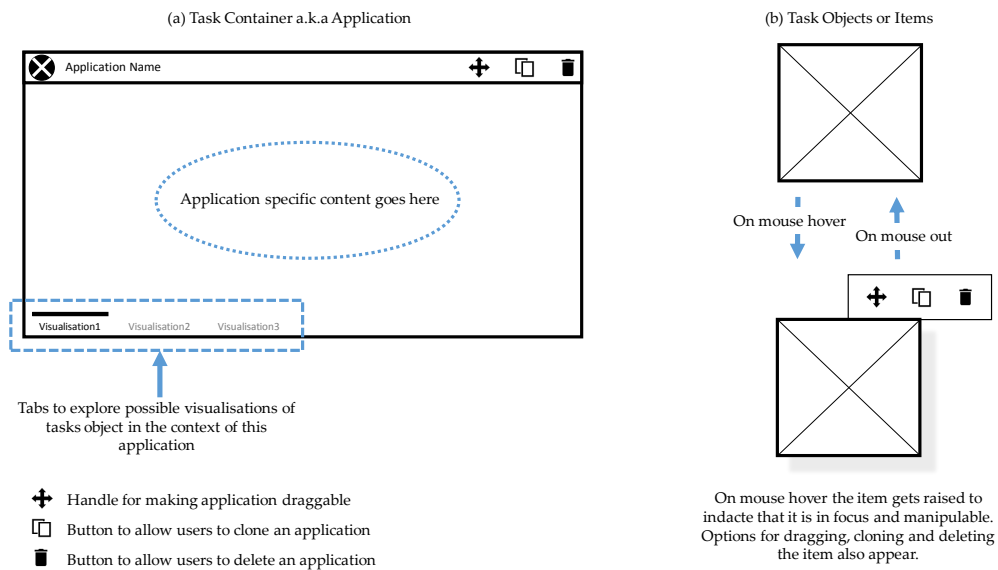


Figure 5.2: Describes the common design and behavior of (a) Task containers, (b) Task objects

5.2.2 TUX object design

Task object's visualization is decided on runtime, but all task objects share some common behaviors.

TUX objects could be any data item or even a container itself. These objects are moved from one container to another and transformed by the hosting container. The actual visualization of an object is decided on runtime by the hosting container 3.1.2 "Task Object". But, all objects show the following common behavior as described in 5.2(b) on mouse hover:

- task object rises and a shadow appear this indicates that the item is in focus and gives it an affordance of being movable,
- a small menu appears to the top right corner of the item with options of dragging, cloning and deleting the item.

This behavior disappears on mouse out. As per TUX, containers can also be appropriated as task objects. The above mentioned behavior of generic task objects is also

applicable and can be seen in TUX containers as shown in 5.2(a).

Following this we now discuss design for previously mentioned generic applications and how each application could be appropriated for Bayesian Analysis.

5.2.3 Sourcing Application

Sourcing application allows users to view, create and edit data in a tabular or spreadsheet format. It allows users to drop files and other data items into the container to view or edit it as a spreadsheet. Each row and column behave as task objects and are draggable. Each row and column can either be rearrange inside the spreadsheet or can be dragged outside the application and be dropped as a task object into another application. Each cell in the spreadsheet application is editable. Mouse pointer changes to text type while hovering over any cell. While clicking on a cell an editable text field appears. The design is made to resemble that of common spreadsheet applications.

Design is similar to that of a spreadsheet with draggable rows and columns.

Figures 5.3, 5.4 and 5.5 describes the design of sourcing application. Figure 5.6 shows a snapshot of the implemented sourcing application. Similar to any other generic application it also has the same window decorations as shown in Figure 5.2(a). Rows and column have the same behavior as that of task objects shown in Figure 5.2(b).

Users can appropriate the Sourcing application for Bayesian analysis in the following ways:

- *Import data set.* Import current or prior experimental data from a file and view it in a tabular format.
- *Create data set.* Create a new data set by typing in entries.
- *Create priors.* Express priors in a tabular form by typing in the outcomes and possibilities.

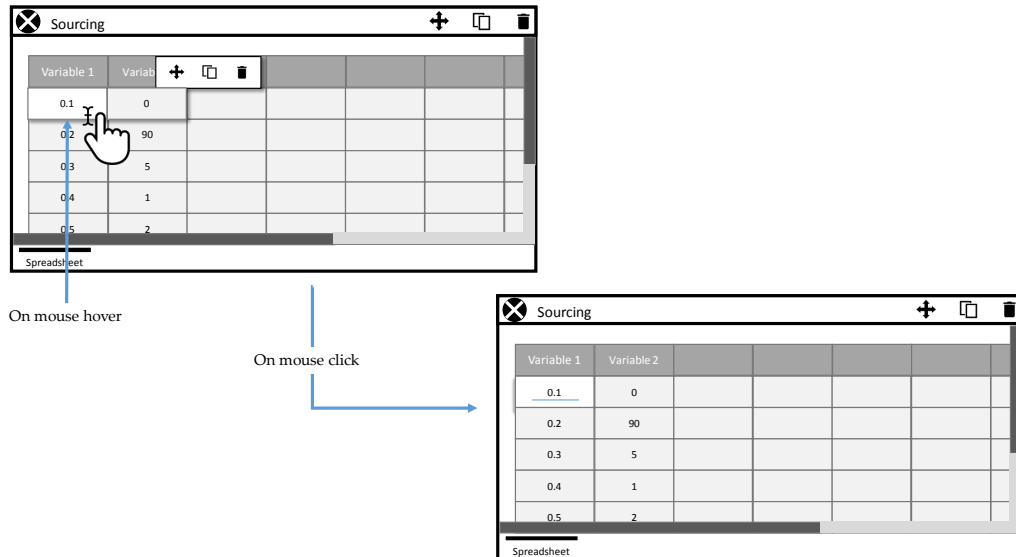


Figure 5.3: Prototype of sourcing application with interactions for mouse hover and mouse click for a particular cell

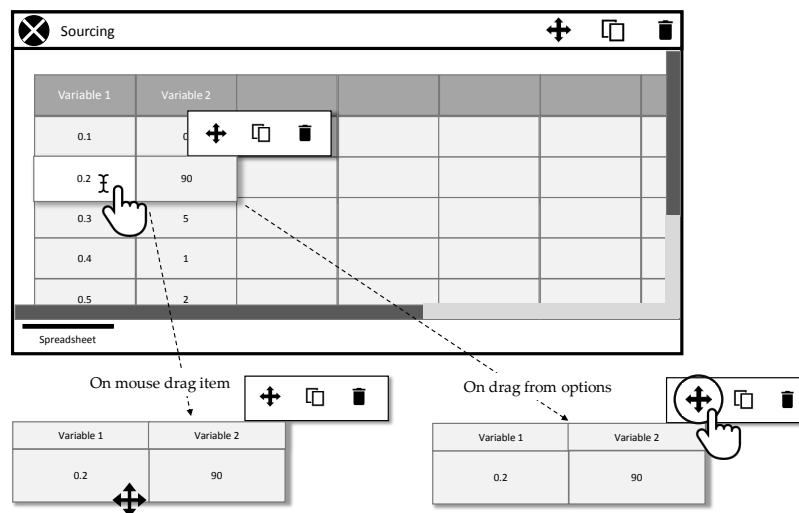


Figure 5.4: Describes interactions for dragging a cell which results into dragging of the entire row

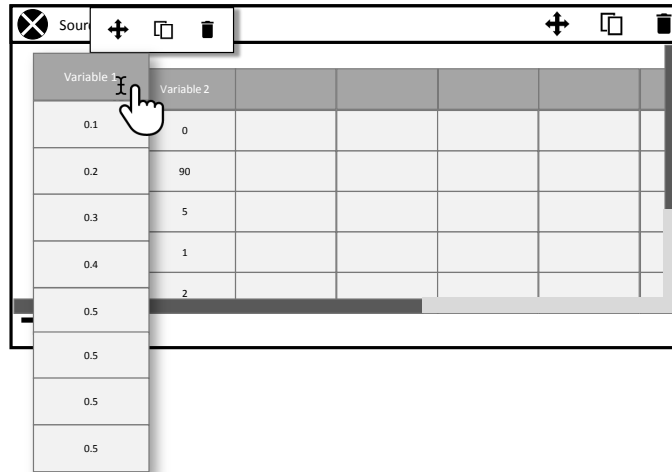


Figure 5.5: Describes interactions for mouse hover over a column header

The screenshot shows a window titled 'Sourcing' with a table. The table has four columns: 'Subject ID', 'Group', 'Result', and 'Comment'. The table contains seven rows of data. The 'Subject ID' column has values 1, 2, 3, 4, 5, 6, and 7. The 'Group' column has values 'Condition' for all rows. The 'Result' column has values 'Y' for all rows. The 'Comment' column is empty for all rows. The table is labeled 'Table' at the bottom left.

Subject ID	Group	Result	Comment
1	Condition	Y	
2	Condition	Y	
3	Condition	Y	
4	Condition	Y	
5	Condition	Y	
6	Condition	Y	
7	Condition	Y	

Figure 5.6: Shows a snapshot of the final implemented version of Sourcing application

- *Data cleaning.* Add or remove variables and data from the dataset which maybe relevant or irrelevant for analysis.

These are just a few examples. Users can appropriate it in many other ways for Bayesian analysis. Some other ways of appropriation were also suggested by participants of our user study.

5.2.4 Planning Application

Helps users in conducting top-down queries or record and report steps in existing investigation.

As discussed above in our system requirements 5.1 “Design Requirements and Conception” users may conduct their inquiries top-down or bottom-up. The Planning container helps users in carrying out top-down investigations. It allows users to list down the steps of their investigation prior to conducting them or it may be used to document the steps that were followed in order to get to their solution. In a business scenario it could be helpful for managers to define tasks and assign them to appropriate subordinates or colleagues.

Similar to any other generic application it also has the same window decorations as shown in Figure 5.2.

Figure 5.7 shows the design of the planning application. The application window has a floating plus icon to the lower left corner. This design has been adapted from Google’s material design to show a call to action button. On clicking this button a new activity is created. Each activity has the following fields :

- Activity title.
- Short activity description.
- Longer activity description.
- Field for adding users who would be working on the activity.

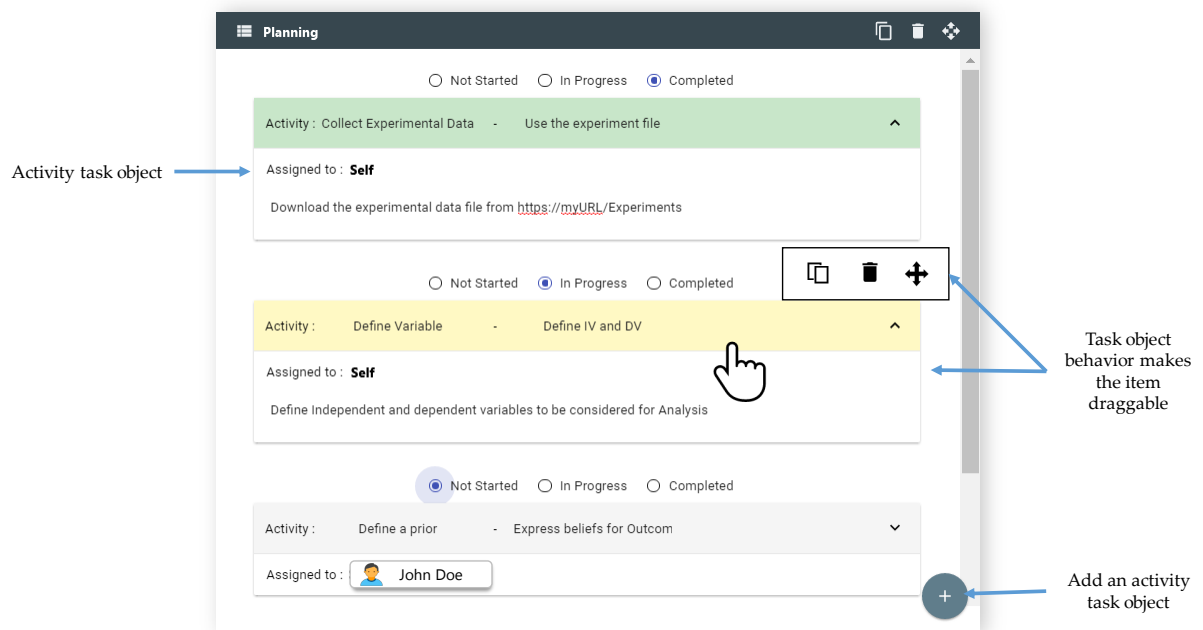


Figure 5.7: Describes the interface for Planning container

- Radio button group to indicate status of the activity as not-started, in progress and completed.

Each activity has the same behaviors as that of a task object as shown in 5.2(b).

In context of performing or appropriating planning application for Bayesian Analysis, users can use it to list down the steps they want to perform during their Bayesian investigations. This can also be seen in Figure 5.7.

5.2.5 Grouping Application

Grouping application as its name suggests, allows users to group items. The application window is divided in two panes. The top pane allows users to create new groups and add new items to any group directly. The bottom half of the application is a suggestions pane.

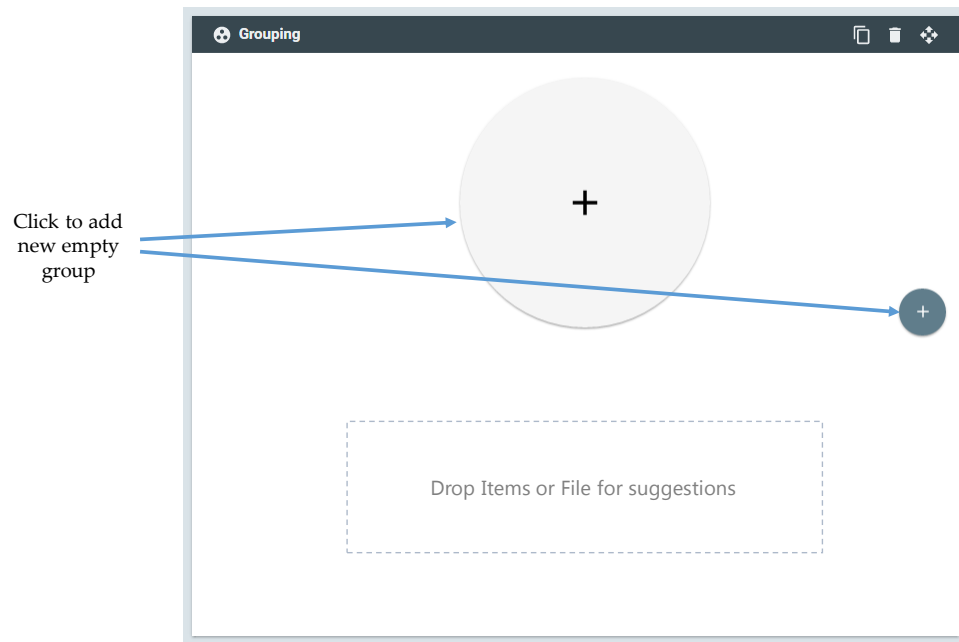


Figure 5.8: Describes the interface for Grouping container.

When a new group is created it is represented by a rounded box added to the top pane of the application. Each group is uniquely identified by a name and a box border color. When the number of groups are more than what can be displayed on the application window a horizontal scroll bar with a focus menu appears to navigate through the list of groups. This can be seen in Figure 5.9.

Each group and each data item inside a group behave like task objects as shown in Figure 5.2

The bottom pane of the application is a suggestions pane. When items of files are drooped in the suggestions pane the application provides suggestions for potential groupings based on the current configuration. When a file is dropped in this area the applications prompts the user to ask if the data should be extracted form the file and used as individual items for grouping or use the file as it is.

Figures 5.8 and 5.9 describes the interface for the grouping

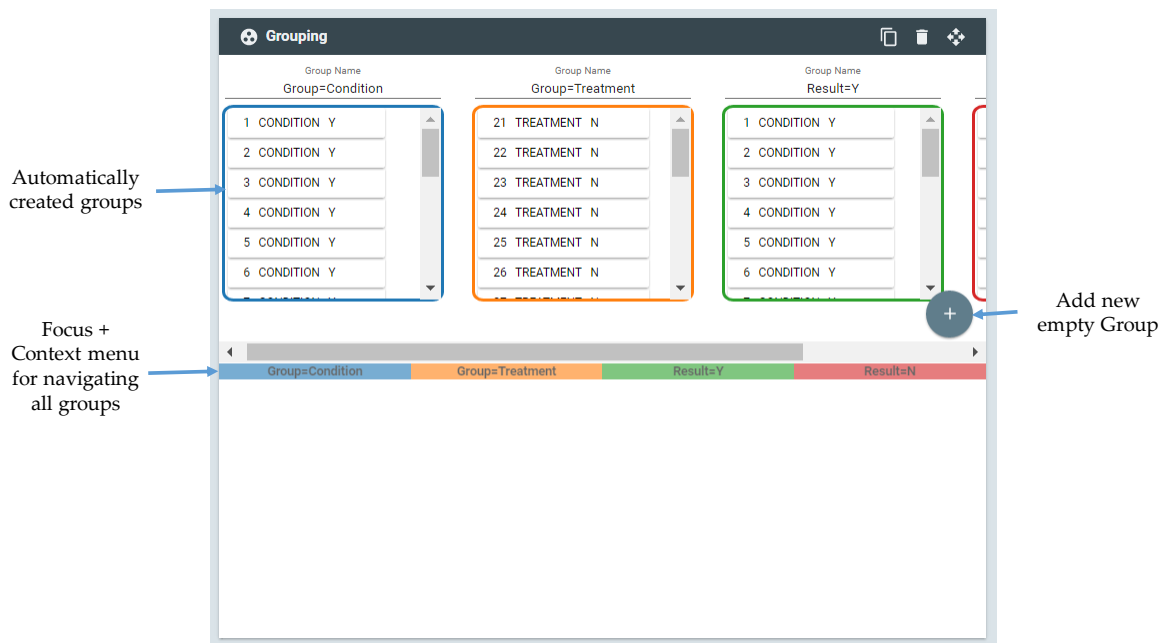


Figure 5.9: Shows the interface for grouping after dropping a file in the suggestions area.

application.

Users can appropriate this application in a Bayesian analysis context in the following ways:

- *Discover groups.* Users can find groups in data sets and use these groups separately for analysis. The grouping application iterates through the dataset and finds similarities to achieve this autonomously. Figure 5.9 shows one such example.
- *Define a hierarchical model.* Users can create groups for independent and dependent variables and fill them appropriately with variables.
- *Grouping priors and likelihoods.* Users can group prior and likelihood data sets into different groups to manage and annotate them as semantically separate data sets.



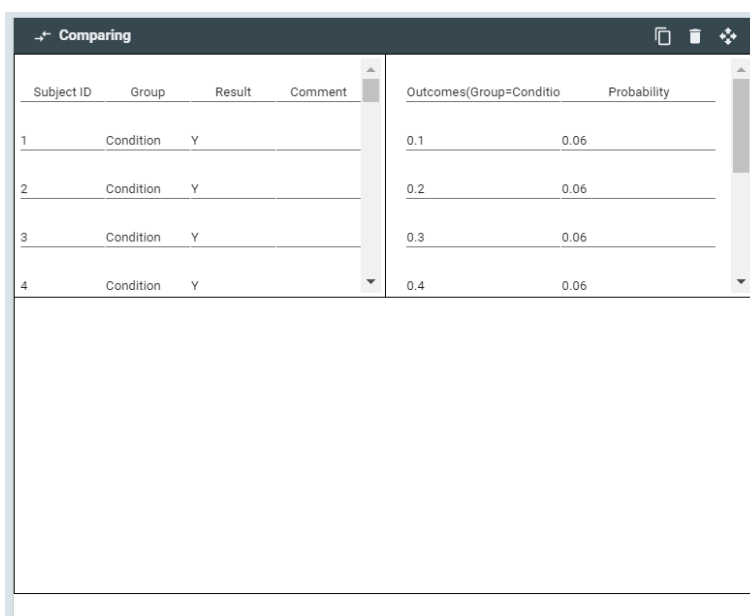
Figure 5.10: Describes the interface for Comparing application.

5.2.6 Comparing Application

Comparing application allows the users to compare any two kinds of task objects or files. Comparing application is also divided into two panes the upper pane is used to drop items that are supposed to be compared and the lower half is used to display the comparison results.

Users can appropriate this application in Bayesian analysis context in the following ways:

- *Comparing groups.* Users can compare outcomes from different groups in an experiment.
- *Comparing priors.* Users may have multiple candidate priors at their disposal they could compare these priors to find differences and effects to choose the most suitable one.
- *Find posteriors.* Users can generate the posterior by comparing prior and likelihood distributions.



The screenshot shows a window titled "Comparing" with two tables. The left table has columns for Subject ID, Group, Result, and Comment. The right table has columns for Outcomes(Group=Conditio and Probability. Both tables have four rows of data.

Subject ID	Group	Result	Comment
1	Condition	Y	
2	Condition	Y	
3	Condition	Y	
4	Condition	Y	

Outcomes(Group=Conditio	Probability
0.1	0.06
0.2	0.06
0.3	0.06
0.4	0.06

Figure 5.11: Shows the interface for comparing application after dropping two files. Each file is visualized as a table.

- *Comparing posteriors.* Users can compare posteriors generated generates using multiple candidate priors.
- *Posterior Predictive check.*

Figures 5.10, 5.11 and 5.12 describe the interface of comparing application.

5.2.7 Visualizing Application

Visualizing application allows users to visualize data sets or create their own custom visualizations. Figures 5.13, 5.14 and 5.15 describes the interface for the this application.

Users can appropriate this application in Bayesian analysis context in the following ways:

- *Visualize experimental data.*



Figure 5.12: Comparing result gets rendered and the result along with the dropped items are displayed as a chart. Alternative tabular visualization option is also provided. The container on the fly detects the context and decides that using charts is a more suitable visualization for data than table.

- *Express priors.* Create priors visually by building a chart. An example of this is shown in Figure 5.15.
- *Find posterior.* calculate and visualize posterior when both priors and likelihoods are dropped together in this container or one on after the other.
- *Build charts selectively.* to incrementally drop single data items from any container to build a custom data set visually.

Figure 5.16 shows all the applications arranged on a canvas.

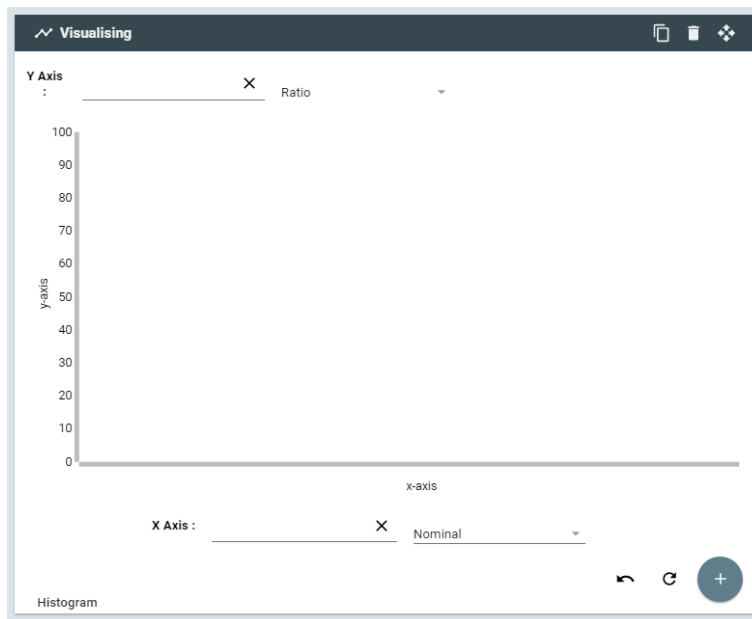


Figure 5.13: Interface for visualizing application

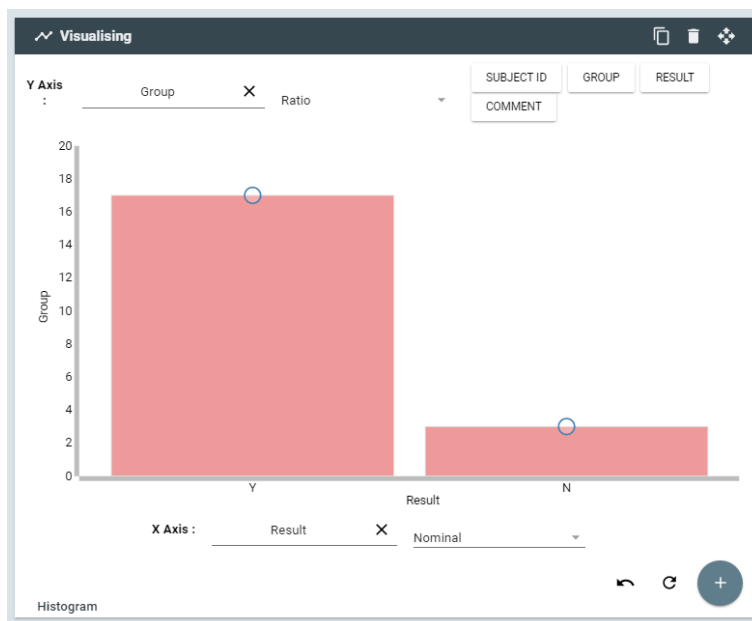


Figure 5.14: Shows the interface after dropping group "Group=Condition" from Grouping application in Figure 5.9 into the Visualizing Container.

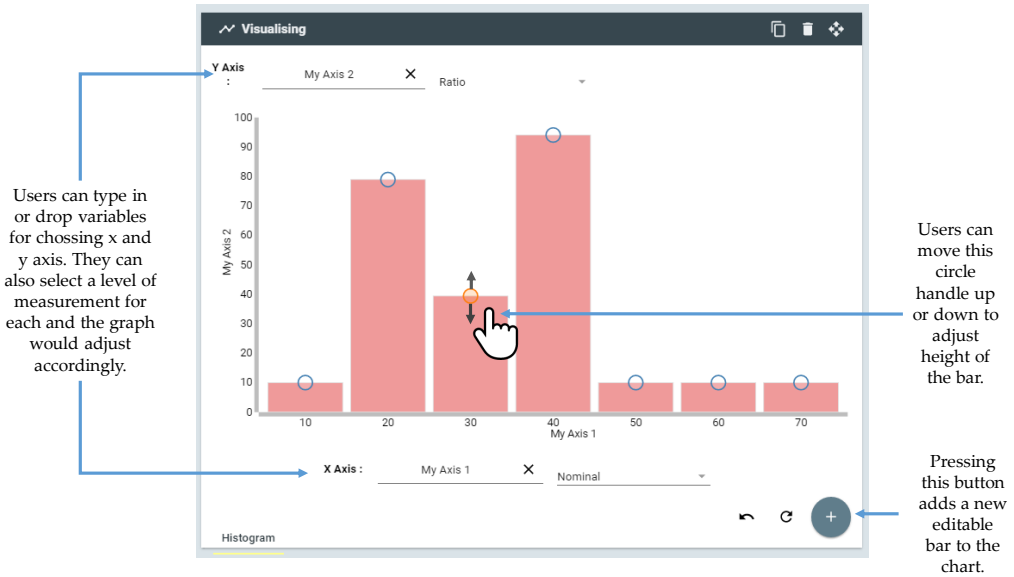


Figure 5.15: Shows how to create a custom visualization.

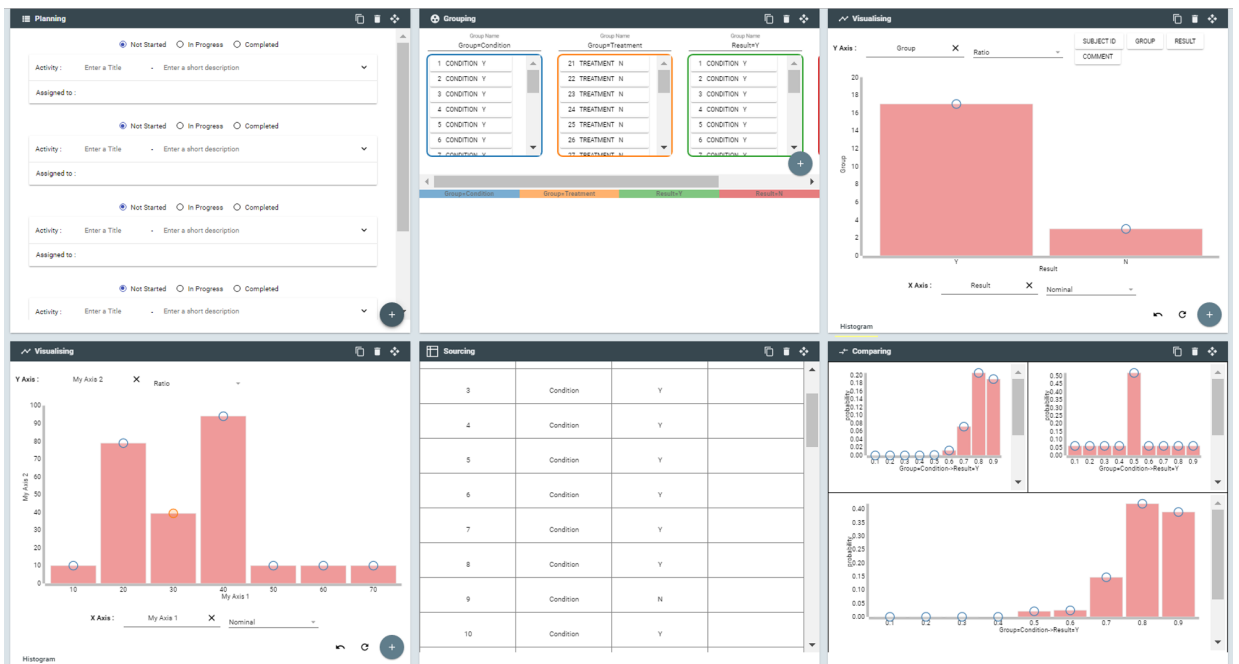


Figure 5.16: All applications arranged on the canvas.

Chapter 6

Implementation

As discussed in the previous chapters in order to accommodate elasticity as a generic system quality, not only the interface, but the software architecture, features, functionality and methods need to be flexible too. To realize elasticity, the software framework has to also be highly elastic. The framework should support task containers to influence the posture of an object and add specific behaviors to it or modify these behaviors on runtime. This allows the hosting task container to manipulate task objects such that they are tailored for a particular context despite not knowing its identity or object type. We built our system on a Component Based Behavior (CBB) framework using Google's web components library, Polymer. We first look at why web components are an ideal fit to realize such a flexible software architecture, what does Polymer offer and finally how we implemented the system. For performing Bayesian Analysis we integrate R¹ using the OpenCPU API².

Software architecture needs to be highly modular and flexible.

Our system uses a Component Based Behavior framework built on Polymer for realizing the UI architecture and R for Bayesian analysis.

¹<https://cran.r-project.org/>

²<https://www.opencpu.org/>

6.1 Web Components and Polymer

Web components facilitate modular web development.

Web components³ are being introduced with the intention to facilitate modular software development for the web. They are a set of features that are presently being supplemented to the DOM and HTML specification by the World Wide Web Consortium⁴. Some important features of web-components are as follows⁵:

- *Custom Elements.*
- *Shadow DOM.*
- *HTML Imports.*

We will discuss these features in detail and how to make use of them using Polymer 1.x in following sections.

Polymer is an open source javascript web-components library.

Polymer⁶, being developed by Google, is an open source javascript web-components library for building modular web applications. It is analogous to using lego building blocks⁷.

Some features that Polymer offers over vanilla web components are as follows:

- Easier method of creating custom elements
- Provides one and two way data binding
- conditional templates and repeated templates
- pre-defined ready to use elements -⁸

³<https://www.w3.org/wiki/WebComponents/>

⁴<https://www.w3.org/standards/techs/components>

⁵<http://w3c.github.io/webcomponents/spec/custom/>

⁶<https://www.polymer-project.org/1.0/>

⁷<https://www.itnews.com.au/news/inside-ing-directs-new-lego-block-app-architecture->

⁸<https://elements.polymer-project.org/>

6.1.1 Custom Elements

This feature enables developers to build their own customized fully equipped DOM elements. Contrary to this developers could build with non-confirming elements which are non standardized elements with application oriented behaviors added later via scripting. With custom elements developers define how the element should be processed by a parser and how would it react to specific changes⁹.

Custom Elements provide API's to create customized DOM elements.

The following code snippet shows how to create custom elements using Polymer 1.x:

```
<dom-module id="my-custom-element">

  /* DOM tree to be rendered goes inside the template */
  <template>
    <style>
      /* Local DOM CSS style */
    </style>

    <!-- Local DOM -->
    <div> Hello {{name}}!</div>

  </template>

  /* Control for the custom element */
  <script>
    Polymer({
      is: 'my-custom-element',

      properties: {
        name: String
        /* Element properties */
      },

      ready: function(){
        /* Called when the local DOM is initialized */
      },
    });
  </script>
</dom-module>
```

⁹<http://w3c.github.io/webcomponents/spec/custom/>

```

        /* Custom methods */
        changeNameToJohn : function(){
            this.name = "John";
        }
    });
</script>
</dom-module>

```

The above written code is placed in a HTML file. This HTML file can then be used as resource and imported where required as shown in

6.1.2 Shadow DOM

Elements placed in a shadow DOM are isolated from other elements.

Using the Shadow DOM feature, browsers can render DOM elements without injecting them into the main document tree. This means that browsers can render and manipulate the DOM the same way as nested elements. However, developers cannot penetrate a shadow DOM in the same way as before. This allows styles in a shadow DOM to become isolated and scoped within the shadow DOM. This allows HTML elements to be completely encapsulated without being exposed to CSS leaks. This prevents elements in a shadow DOM to get effected by unintended external code or behavior.

Events are used to communicate with elements in a shadow DOM and vice versa.

The encapsulated DOM tree inside a shadow DOM is known as a shadow tree. DOM elements in a shadow tree can communicate with rest of external DOM by firing events. These events can be picked by elements and can respond to it and vice-versa.

6.1.3 HTML Imports

HTML Imports are used to load requested resources.

HTML imports ¹⁰ allow to reuse and include web components i.e. custom elements and HTML documents. The following example shows how to import the custom element we created in 6.1.1 “Custom Elements”:

¹⁰<https://www.w3.org/TR/2016/WD-html-imports-20160225/>

```
<link rel="import" href="my-custom.element.html">
```

HTML imports also make sure that a required resource is not imported multiple times.

6.2 Component Based Behavior

Custom elements facilitate adding and registering new DOM elements at runtime into a Shadow DOM. This allows element added at runtime to be function independently. We implement task containers and objects as custom elements. This would allow to clone, add and remove task objects and containers at runtime. Custom elements facilitate such a high level composability at runtime.

These elements are independent and operate autonomously. Another important aspect is to attach functionality and behaviors to an existing element at runtime. If components are built from composable functionality and behaviors, it would also be possible to add or remove desired behaviors at runtime.

TUX containers built with such composable behaviors would accommodate as per evolving context. Since applications a.k.a containers are underspecified, users can appropriate them by expressing the context of use. With changing context such micro behaviors can be added and or removed from containers to provide users with the right set of features required in context.

TUX objects built with composable micro behaviors would be flexible. Task containers can adjust postures of containing task objects by manipulating its behaviors and appearance. With this, task containers can adjust containing task objects on the fly.

To achieve this level of system granularity we use Component Based Behaviors (CBB) which enables adjusting task objects and containers during runtime. CBB proposes to implement generic manipulatable

Task containers and object are placed in shadow DOMs and operate independently.

If system functionality could be decomposed of smaller behaviors, then each behavior could be written as custom elements to function independently.

Containers and Objects built with such behaviors could be composed at runtime to fit the needs of a context.

functions as independent components i.e independent custom elements. This way all such composable behaviors are independent from one other but can still work in a synergy. Since, each such micro functionality i.e. behavior is a DOM element they can be easily be switched at runtime.

6.2.1 CBB Implementation

Lets, first look at an example of CBB from Polymer:

```
<div style="position: relative">
  <paper-ripple></paper-ripple>
</div>
```

In the above code `<paper-ripple></paper-ripple>` is a CBB. `<paper-ripple>` provides a visual effect that other DOM elements can use to simulate a rippling effect originating from the point of contact i.e. touch or mouse down. The effect is visualized as concentric circle in motion as shown in Figure 6.1(b).

A normal button as shown in 6.1(a) is implemented as follows:

```
<button type="button">NORMAL BUTTON</button>
```

In order to give it the ripple behavior as shown in figure 6.1(b), we just add the `<paper-ripple>` custom element as a child of the button as follows:

```
<button type="button">
  NORMAL BUTTON
  <paper-ripple></paper-ripple>
</button>
```

A behavior inserted as a child node adds features to its immediate parent.

The `<paper-ripple>` behavior adds abilities to the parentNode hosting the behavior. If multiple such

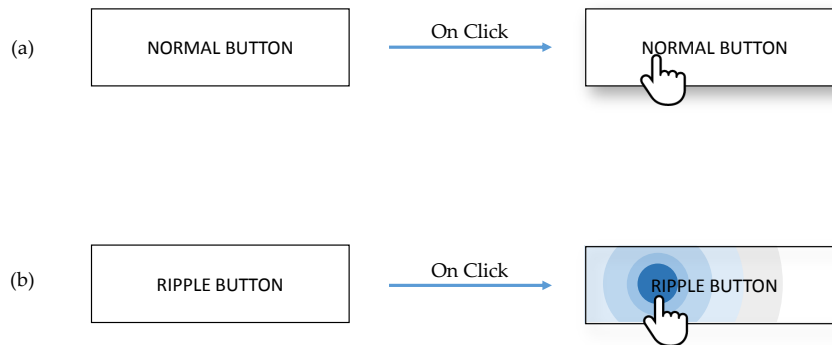


Figure 6.1: Shows (a) vanilla button on mouse click (b) button with a paper-ripple behavior

behaviors are added as child nodes. All such behaviors would add abilities to the parent node. Polymer offers the following life cycle callback methods when a custom element is created ¹¹:

- `createdCallback` This is used for a single time setup before local DOM can be initialized and property values be set.
- `ready` This method is called after the property values are set but DOM has not yet been initialized. At this stage it possible to access property values.
- `attachedCallback` This is called after a custom element has been attached to the document. Here developers can add event listeners and access computed styles. When a behavior is inserted to a DOM element in this callback the `parentNode` can be accessed and behavior specific functionality can be added.

¹¹<https://www.polymer-project.org/1.0/docs/devguide/registering-elements>

- `detachedCallback` This callback is called when a custom element is removed from the the DOM. So, when a behavior is removed from DOM, we can remove the behavior specific functionality from the parent Node.
- `attributeChanged` This is called when an element's attribute is changed. This is not the same as observing properties. in order to register a callback when a property is changed, `observers` are used.

So, the `attachedCallback` and `detachedCallback` are used to modify abilities of the `parentNodes` when behaviors are added or removed. Examples of some behaviors that we implement are as follows:

- `<model-behavior>`
- `<behavior-draggable>`
- `<behavior-dropzone>`
- `<sortable-behavior>`

6.2.2 Model Behavior

Model behavior allows to attach a model to any DOM element. This behavior can be attached to any element which represents a task object. A container can access the data of this task object by retrieving the `<model-behaviour>` DOM element and querying for its model property. This provides a generic way for all containers to access the model of any task object when it is dropped into the container. We added the ability to set and retrieve a model to every HTML element as this feature is used very frequently and it improves code readability.

The following code shows how to retrieve the model of any DOM element using Polymer:

```
HTMLElement.prototype.getModel = function(){
  var modelElement = Array.from(this.childNodes)
    .filter(el => el.tagName === "MODEL-BEHAVIOUR")[0];
  if(modelElement === undefined)
  {
    if(this['model'] !== undefined)
      return this['model'];
    else
      return undefined;
  }
  return modelElement['model'];
};
```

The following code shows how to set the model of any DOM element using Polymer:

```
HTMLElement.prototype.setModel = function(model){

  var modelElement = Array.from(this.childNodes)
    .filter(el => el.tagName === "MODEL-BEHAVIOUR")[0];
  if(modelElement !== undefined)
  {
    if(isArray(modelElement['model'])) {
      modelElement['model'] = [];
    }
    else if (isObject(modelElement['model'])) {
      modelElement['model'] = {};
    }
    modelElement['model'] = model;
  }
  if(this.model)
  {
    if(isArray(this.model)) {
      this.model= [];
    }
    else if (isObject(this.model)) {
      this.model = {};
    }

    this.model = model;
  }
}
```

```
};
```

With the above methods in place model from an element can be accessed by just calling `element.getModel()`. And setting the model can be done by just calling `element.setModel(model)`.

task objects can be of any type and can have different complexity of models attached to it. We define a simple format in which a model needs to be declared. This standardized way of declaring models allows all containers to parse the model and construct a model in a consistent manner. An example is shown below:

```
model : {  
  key1 :  
    {'common:value':[],  
     'display:render':true,  
     'type':generic},  
  
  key2 :  
    {'common:value':"SubjectID",  
     'display:render':true,  
     'type':'variable'}  
}
```

'common:value' is used to store the data. 'display:render' is a boolean flag which is used to decide if the values need to be displayed on the interface or not. type is used to register the semantic type of the element eg: variable, car, person etc...

The value of the `display:render` flag can change over time. This allows containers to display the right amount of information in a given context.

6.2.3 Draggable Behavior

Attaching this behavior to any DOM element makes it draggable. All task objects and containers are draggable. Task objects start exhibiting the behaviors described in 5.2.2 “TUX object design”. When the `<draggable-behaviour>` gets attached the following `attachedCallback` gets executed:

```
attachedCallback : function () {

this.parent = this.parentNode;
this.parent.style.position="relative";
...
...

/* elementCloneDeleteMove is a custom element which
adds the move, clone, delete menu to the element */

var element = new elementCloneDeleteMove(this);
this.parent.appendChild(element);

element.setAttribute("style",
"display:none;z-index:9999;position:absolute");

this.parent.addEventListener("mouseover",
function(e){e.stopPropagation();self.showElement();});

this.parent.addEventListener("mouseout",
function(e){e.stopPropagation();self.hideElement();});

/* this makes the parent draggable */
this.parent.setAttribute('draggable',true);

/* attaching drag-* event listeners to parentNode */
this.parent.addEventListener('dragstart',
function(e){e.stopPropagation();self.iWasDragged(e,self);});

this.parent.addEventListener('touchstart',
function(e){e.stopPropagation();});

this.parent.addEventListener('dragstart',
```

```
function(e){e.stopPropagation();});  
},
```

The above code snippet shows how properties, appearances and event listeners can be attached to the parent node hosting a behavior.

6.2.4 Dropzone Behavior

Attaching this behavior to a DOM element converts it into a drop zone where draggable items can be dropped. This behavior goes a step further and attaches history metadata to every task object. The history property records the transformation of an object. This can be later used to inspect the flow and transformation of data for sense making purposes. This behaviour also abstracts extracting data from excel files and convert it into a generic model format.

6.2.5 Behavior Sortable

This behavior allows to rearrange objects row-wise, column-wise, matrix or in a list. The row wise and column wise rearrangement behavior can be seen in the seen in the table application. The list reordering behavior can be seen in groups and the matrix reordering behavior can be seen in the workspace a.k.a canvas where applications can be rearranged.

6.3 Implementing Task objects and containers

Task objects and containers are implemented as custom elements. Each custom element has its own controller which handles the application logic. When an element is dropped into a container an event is fired. The controller

intercepts these event to get a handle of the dropped element. It extracts the model from the model behavior of the dropped element to determine properties of the object needed to be rendered. Based on semantics of these property values the container controller choose a suitable posture of the task object and realizes it by either modifying a task objects behaviors or creating an entirely new custom element to represent the task object.

All containers have a common window decoration. We implemented a high level custom element window called `<container-application>` which encapsulates this window decoration and functionalities that come with it. When an application for example `<grouping-application>` is placed inside the `<container-application>` it inherits the window decoration and behaviors from it. The following pseudo code snippet shows how this is done.

Pseudo code for `<comparing-application>` placed in `comparing-application.html` file:

```
<dom-module id="container-application">

  <template id$= "{{id}}">

    /* paper material creates a window
    on the canvas for the application*/
    </paper-material>

    <div class="header">
      /* Code for header toolbar goes here */
    </div>

    /*The content tag acts as a slot*/
    <content select = "[widget]"></content>

    <div class="footer">
      /*code for footer hosting multiple visualization goes here*/
    </div>

  </paper-material>
```

```

    </template>

    <script src = "containerController.js"></script>

</dom-module>

```

The `<content>` tag acts as a slot where UI for other applications get inserted. The `<container-application>` is a custom element which can be imported as an HTML resource via HTML import. In the `<grouping-application>` custom element, `<container-application>` is placed inside the `<template>`. Now any DOM Tree added as a child of `<container-application>` will get added to the `<content>` slot in the code snippet above. Both container and grouping applications have their own controller for container specific application logic.

Pseudo code for `<grouping-application>` placed in `grouping-application.html`:

```

<dom-module id="grouping-application">

    <template id$= "{{id}}">

        <container-application id="grouping"
            title="Grouping"
            class="groupingToolbar"
            icon="group-work">

            <section>

                ...

                /* Interface for grouping application*/

                ...

```

```
    </section>

  </container-application>

</template>

<script src="groupingController.js"></script>

</dom-module>
```

We built the interface unique to application and objects using default elements and widgets provided by the Polymer library. The polymer element catalog documentation can be found at <https://elements.polymer-project.org/>.

6.4 Implementation for Bayesian Analysis

To support Bayesian analysis we use R functions via the OpenCPU API. Computations for calculating probability distributions inferred on the basis of independent and dependent variables is done by calling R functions. To use OpenCPU it is first required to import jquery and opencpu javascript libraries. The following snippet shows how to call an R function to calculate a discrete probability distribution for 7 positive outcome in 20 independent trails for a given probability model p:

```
ocpu.seturl("//public.opencpu.org/ocpu/library/stats/R");
ocpu.rpc("dbinom", {
  x:7,
  size:20,
  prob:p
}, function(result) {});
```

Using web-components' custom elements, HTML imports, shadow DOM; components based behaviors and custom elements from Polymer we realized an elastic interface as imagined by TUX.

Chapter 7

Evaluation

We conducted a think-aloud user study to evaluate the exploratory power and usability of our system based on Transformative User Experience Design.

7.1 User Study Protocol

Since the study involved a closer observation of users in problem solving and exploratory mode, we recorded the participant's screen to capture their actions. Since this was a think aloud study we recorded participant's voice. They were asked to perform tasks representative of steps involved in Bayesian analysis.

Participant's screen and voice were recorded. They performed simple Bayesian analysis tasks.

7.1.1 Setup

The test environment selected was a typical office environment. Participant was seated in front of a 21 inch LCD monitor and given a mouse and keyboard. Mouse was used instead of a trackpad, as in the pilot study user was uncomfortable using the trackpad on our system. Since the majority of interaction involved drag and drop actions we decided to use a mouse. Users were allowed to adjust mouse sensitivity as per their comfort. This was

done so that the medium of interaction didn't become an extraneous variable in our study.

7.1.2 Procedure

Tutorial was provided to firstly, make sure they understood the concepts and secondly, to reduce the time duration of the actual user study.

Participants were sent a short tutorial on Bayesian Analysis a day before the user study. After the participants arrived for the study, we went over a few basic concepts of Bayesian analysis to ensure that they understood the bare basics of the topic and were comfortable with terms like independent variables, dependent variables, model, probability distributions, likelihood, prior, levels of measurement, etc. Once the participant arrived, they were given a consent form which also stated the purpose of the study. Some demographic data was collected as listed in A "Appendix for User Study".

First participants were shown the interface with the the five applications arranged on the canvas. They were asked to think aloud, what they understood from by just looking at the interface without using it just yet. After this, they were allowed to explore the interface for not more than five minutes and get used to moving around applications on the canvas and experiment with features of individual applications.

Participants were asked speak out their actions and expected responses before performing a task.

After participants had briefly explored the system, they were given the below mentioned tasks. For each task users were free to use the system as they desired. Before they performed an action, they were asked to speak out the action and the response they would expect from the system. For each task, participants were also asked to suggest generic methods and features that if present would have aided them to perform that specific task.

After participants completed the tasks they were asked to fill out form to measure system usability. This is measured using the System usability scale (SUS) and another form to provide general feedback. User Study forms are attached in A "Appendix for User Study".

Tasks

Tasks given to participants were some basic steps involved in performing Bayesian Analysis. Users were given the a hypothetical example discussed in 4.2 “Performing Bayesian analysis with a simple example”.

Participants of our system are given an excel file with data recorded from the example. Users are already familiar with the process of performing Bayesian Analysis. A set of predefined detailed task were not given because we wanted the user to explore. But a set of steps were given to them for reference:

- Data Identification
- Defining Independent and Dependent Variables
- Expressing Priors
- Finding the posterior

7.1.3 Method of Analysis

Our goal is to measure System Usability and the Freedom offered by the system.

System usability

We measured system usability is measured using th system usability scale (SUS). It is “a quick and dirty, reliable tool for measuring usability” [Brooke et al., 1996]. SUS has become a standard in the industry. The befits of using SUS are as follows¹:

- It is very easy for participants to understand and give responses, thus making it easy to administer.

¹<https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>

- It provides reliable results with even small sample sizes.
- It effectively differentiates between usable and unusable systems.

Participants are given the following 10 statements and they score these statements on a 5 point likert scale from strongly-disagree to strongly agree:

- I think that I would like to use such a system frequently.
- I found the system unnecessarily complex.
- I thought that the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in the system were well integrated.
- I thought that there was too much inconsistency in the system.
- I would imagine that most people would learn to use this system quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I need to learn a lot of things before I could get going with the system.

The SUS consists of ten questions. Each question can be answered with a response on a five point likert scale from strongly disagree to strongly agree and numerically marked from 1 to 5 respectively. The final outcome of the SUS is a number that represents the usability of the system. The score for each individual item is not meaningful in itself.

To calculate the SUS score, for statements 1,3,5,7 and 9 the score contribution is the score minus 1. For statements 2,4,6,8,10 the score contribution is calculated as 5 minus the score. After this the score contributions are summed up and the sum is multiplied by 2.5 to obtain an overall system usability score on a scale of 0 to 100.

Freedom offered by the system

This is a qualitative measure. We analyze users actions and voice recordings to investigate this. We look at the number of ways in which a user tries to accomplish a particular task.

7.1.4 Participants

A total of seven participant were a part of this study aged 27-30($M=28.3$, $SD=1.1$,3 females and 4 females). four participants were developers, two participants were university students and one participant was a user experience designer. All participants were from computer science background. Only one of seven participant, had studied Bayesian Analysis formally and was familiar with concepts of Bayesian Analysis. All seven participants had formally studied statistics.

7.2 Results

7.2.1 System Usability

The combined SUS score, $M=78.2$, $SD=3.5$. Figure 7.1 shows the grade ranking of SUS scores from [Bangor et al., 2009]. Our results indicate a good SUS score with a high system acceptability.

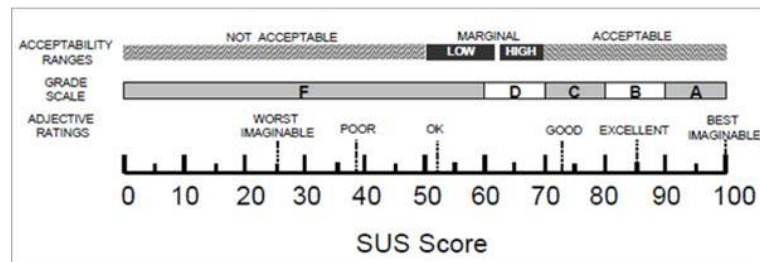


Figure 7.1: Grade ranking of SUS scores from [Bangor et al., 2009]

7.2.2 Freedom offered by the system

Here we look at the set of actions users performed to accomplish a particular task. Each series of actions is followed by a fraction which indicates how many users performed this action out of a total of seven users.

- Actions performed for Data Identification
 - Dropping data file directly into the sourcing container, then grouping data using the grouping container followed by visualizing the required group by dropping it in the visualization container: (7/7)
 - Dropping data file directly into the visualizing container, followed by assigning required variables to x and y scale to obtain the required data set. (5/7)
 - Dropping data file directly into the grouping application to discover groups in data followed by dropping the required group in the visualizing container. (5/7)
- Defining Independent and Dependent Variables
 - Annotating variables as independent and dependent in the Sourcing application: (3/7)
 - Create two groups for independent and dependent variables and group variables accordingly: (7/7)

- Dropping dependent variable and independent variable on the x-axis and y axis of the visualizing application respectively, assuming x axis by default to represent independent variable and y axis to represent dependent variable: (3/7)
- Expressing Priors
 - Using Sourcing application to create priors by text entry: (7/7)
 - Using Visualizing application to visually create priors: (7/7)
- Finding the posterior
 - Creating two groups, likelihood and prior, putting relevant data into these groups and dropping the Grouping application directly into Visualizing application: (6/7)
 - Creating two groups, likelihood and prior, putting relevant data into these groups and dropping the Grouping application directly into Comparing application: (4/7)
 - Creating two groups, likelihood and prior, putting relevant data into these groups and dropping the grouping application into the Sourcing application: (3/7)
 - Dropping likelihood and prior data one after the other into the Visualizing application without grouping: (5/7)
 - Dropping likelihood and prior data one after the other into the Comparing application without grouping: (5/7)

These results indicate that the system allowed enough freedom for participants to explore data and problem in their own way. Additionally, all users were able to find the posterior distribution for our given scenario. This is not a definitive measure but an indicative measure of moving in the correct direction. Also, in the list above we have simplified and grouped user actions, but actually

participants performed multiple permutations across these tasks and action sets. A deeper analysis into this will reveal more insights. But, this was placed outside the scope of this thesis.

7.2.3 User Feedback

System was easy to use and understand. But, users were not sure if the system would understand their actions.

Most users found the system very simple to use as they were composed of generic features. On one side they found it easy and understandable, the other side they were un-sure if the system would understand their intentions. They were a bit hesitant in the beginning while performing task as they didn't know if their appropriation would work. A participant suggested, *"There should be a way for the system to continuously communicate what context inferences it is making."*

Users suggested the following generic applications which could be useful from their perspectives:

- Network building application. An application which could draw directed arrows from one object to another. Such an application could be used to show dependency or build organizational trees etc.
- Clustering application.
- A generic text editor applications.
- An application that could be used to browse the web and import data from any web source.

Users experience the following issues while using the grouping application:

- A few users didn't realize the presence of the focus+context menu for navigating to invisible groups.
- While hovering over the items inside the group they didn't realize that the items inside the groups were also draggable.

Users experience the following issues while using the visualizing application:

- Some users couldn't figure out immediately how to build a custom visualization from scratch.
- Some users couldn't figure out that variables were extracted when an item was dropped into the visualizing application. And in order to visualize the graph variables had to be dropped in the respective axes. They expected a random visualization to appear as soon as data was dropped on the container.
- Most users couldn't figure out x and y axis fields were dropzones.

There were no comments for planning, sourcing and comparing applications.

Chapter 8

Summary and future work

We now conclude with summarizing the thesis, and briefly discuss scope of future work.

8.1 Summary

Inspired by supporting user's emergent needs in problem solving mode, we designed and developed generic software support based on elastic design principles. We built domain specific support for performing basic Bayesian analysis with the system.

In order to do so we first delved into research on complex problem solving to understand what are complex problem? what uniquely characterizes them? and what are the strategies adopted by problems solver while working with such problem? From this insight we distilled core generic activities that problem solvers perform. Complex problems are characterized by uncertainty. User's goals keep evolving as and when new insights appear. Core generic activities that users perform in problem solving mode are wayfinding, sensemaking and managing data ordeals. To understand goal oriented activities better we turned to

Activity theory for better understanding how users interact with tools and perform activities to achieve their goals. Activity theory suggests that a singular reference to context is insufficient to support users in open ended queries. Users work in multiple contexts while in problem solving mode. We realize that interfaces and software support for such applications needs to be flexible to accommodate uncertainty. End user Development (EUD) is a paradigm that suggests for such flexibility and adaptability of systems at runtime. EUD suggests using a modular approach to achieving such flexibility. We adopt a few EUD activities like annotation, parameterization, composition and modification for adapting system functionalities at runtime. EUD's meta design theory suggests under specifying application so that they can be appropriated by the users the way they want in the context they want.

Transformative User Experience Design builds on the above mentioned lessons from EUD and meta design. It aims at supporting users in variety of spontaneously self defined task flows. It introduces two basic concepts, task containers and task object. Task objects are data items of interest to the user and task containers represent contexts which host these data items. Problem solvers operate in multiple such contexts. User should be able to move task objects from one context to the other. Containers should be underspecified so that users can impose their intentions and express a context of use. System should allow user to perform EUD activities in order to do so. Context can also be inferred by containers based on the task objects contained in it. Container should be able to adjust the posture of containing task objects to match the context of use. We discuss how TUX proposes to achieve elasticity as a generic quality of use.

Based on insights gained from complex problem solving, end user development, activity theory, meta design and TUX we narrow down concrete system requirements to build generic support for complex problem solving. We propose an interface design with 5 generic applications, grouping, sourcing, planning, visualizing and comparing. There are many other such generic applications that can be developed but we choose to implement only these 5

in the scope of this thesis. These are generic applications which are not built for nor reflect any specific use case placed on an infinitely extending canvas. These represent task containers described by TUX. We describe in detail interaction design for these applications and interaction potentials they offer.

To realize elasticity the software architecture, methods, features and functionality need to be flexible to support elasticity at the interface level. We use web-components specifications i.e. custom elements, shadow DOM and HTML imports to realize such flexibility. And, we build on a component based behavior framework powered by the polymer. Applications and objects are implemented as custom elements placed in a shadow DOM. This allows them to work independently without being affected by others.

We conduct a qualitative user study to evaluate the freedom offered by the system to allow exploration in Bayesian analysis and evaluate the usability of the system. Results of the study indicate high acceptability of the system. Participants in the study could understand and appropriate generic applications for performing Bayesian analysis. They were able to achieve their goals in a self informed self defined manner in multiple different ways.

8.2 **Future work**

As discussed earlier, in the scope of this thesis we do not focus on building a highly efficient zoom-able interface. Implementing a zoom-able interface with multiple strategies for organizing and managing applications on the canvas would be extremely useful.

We implemented support for performing basic Bayesian analysis. More support could be integrated for performing more complex statistical analysis.

Currently, task objects and containers are no not purely built out of behaviors. They are a mix of behaviors

and custom elements. More generic behaviors could be developed so that containers and task objects would be truly composable.

We did not built a tool-bar or any other element, from which applications could be spawn and brought onto the canvas. A tool-bar containing all applications allowing users to drag new application instances to the canvas would be very useful.

User study points out some usability flaws in the system which need to be corrected. A participant suggested during the study that the system should communicate the context it is constantly inferring from user actions. This would allow the users to correct their actions.

We implement only five generic containers, more containers like clustering, network building etc. could be added to provide more comprehensive support. There could be numerous generic and domain specific applications which could be useful.

Each application provides a generic functionality. And users use these applications independently. Generic applications could be merged by merging behaviors to create new applications at runtime.

The system is built to be used by a single user at a time. This could be further extended as a collaborative platform. Multiple users could share a common workspace and solve a larger problem together.

Task containers do not not know the identity of objects which get dropped onto it. It would be useful to build an intelligent context engine which could evaluate the canvas to infer context of use and identify object types by iterating over it properties.

We realized elastic design principles via Direct Manipulation Interfaces paradigm. But, TUX is not bound by any UI paradigm. It would be interesting to explore elasticity for other modes and modalities of interaction.

Appendix A

Appendix for User Study

User Study Form

User ID :

Name :

Age :

Gender :

Profession :

Highest degree :

1. Have you studied Statistics?

Yes No

2. Have you studied Bayesian Analysis?

Yes No

3. If yes, are you familiar with keywords like Prior, Posterior, Likelihood?

Yes No

4. Are you familiar with any software program for statistical analysis?

Yes No

5. How often do you work with such a software?

Daily Weekly Once in a while Never

6. Are you familiar with interfaces with drag and drop features?

Yes No

7. How often do you work with such interfaces?

Daily Weekly Once in a while Never

Figure A.1: Collecting user details and demographic data form

8. Tick the appropriate option

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1. I think that I would like to use such a system frequently.					
2. I found the system unnecessarily complex.					
3. I thought that the system was easy to use.					
4. I think that I would need the support of a technical person to be able to use this system.					
5. I found the various functions in the system were well integrated.					
6. I thought that there was too much inconsistency in the system.					
7. I would imagine that most people would learn to use this system quickly.					
8. I found the system very cumbersome to use.					
9. I felt very confident using the system.					
10. I need to learn a lot of things before I could get going with the system.					

Figure A.2: SUS form

9. Feedback:

1. Grouping Application

2. Spreadsheet Application

3. Visualizing Application

4. Comparing Application

5. General (suggestions/things you liked/disliked)

Figure A.3: Feedback

Bibliography

Australian Public Service Commission et al. Tackling wicked problems: A public policy perspective. 2012.

Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3): 114–123, 2009.

Gregory Z Bedny. *Application of Systemic-Structural Activity Theory to Design and Training*. CRC Press, 2014.

John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

C West Churchman. Guest editorial: Wicked problems. *Management Science* Vol. 14, 1967.

Maria Francesca Costabile, Daniela Fogli, Piero Mussio, and Antonio Piccinno. End-user development: The software shaping workshop approach. In *End user development*, pages 183–205. Springer, 2006.

Alan Dix. Designing for appropriation. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 2*, pages 27–30. British Computer Society, 2007.

Paul Dourish. The appropriation of interactive technologies: Some lessons from placeless documents. *Computer Supported Cooperative Work (CSCW)*, 12(4): 465–490, 2003.

Arthur Conan Doyle. *The sign of four*. Broadview Press, 2010.

- Gerhard Fischer and Elisa Giaccardi. Meta-design: A framework for the future of end-user development. In *End user development*, pages 427–457. Springer, 2006.
- Raymonde Guindon. *Cognitive science and its applications for human-computer interaction*. Psychology Press, 2013.
- Austin Henderson and Morten Kyng. There’s no place like home: Continuing design in use. In *Design at work*, pages 219–240. L. Erlbaum Associates Inc., 1992.
- Victor Kaptelinin. Activity theory. In *Encyclopedia of Human Computer Interaction*. Interaction Design Foundation, 2015.
- Victor Kaptelinin and Bonnie A Nardi. *Acting with technology: Activity theory and interaction design*. MIT press, 2006.
- John Kruschke. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press, 2014.
- Markus Latzina and Joerg Beringer. Transformative user experience: Beyond packaged design. *Interactions*, 19(2): 30–33, 2012.
- Markus Latzina and Joerg Beringer. Elastic workplace design. In *Designing Socially Embedded Technologies in the Real-World*, pages 19–33. Springer, 2015.
- William Lidwell, Kritina Holden, and Jill Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.
- Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. End-user development: An emerging paradigm. In *End user development*, pages 1–8. Springer, 2006.
- Nikolay Mehandjiev and Leonardo Bottaci. User-enhanceability for organisational information systems through visual programming. In *International Conference on Advanced Information Systems Engineering*, pages 432–456. Springer, 1996.

- Barbara Mirel. *Interaction Design for Complex Problem Solving: Developing Useful and Usable Software*. Morgan Kaufmann Publishers Inc., 2003.
- Bonnie A Nardi. *A small matter of programming: perspectives on end user computing*. MIT press, 1993.
- Donald A Norman. Human-centered design considered harmful. *Interactions*, 12(4):14–19, 2005.
- Alan J. Perlis. Epigrams on programming. *Sigppplan Notices*, 17(9):7–13, 1982.
- Jens Rasmussen and Kim J Vicente. The ecology of human-machine systems ii: Mediating 'direct perception' in complex work domains. *Ecological psychology*, 2(3):207–249, 1990.
- Horst Rittel. Second generation design methods. In *Developments in Design Methodology*. Wiley & Sons, 1984.
- Mary Shaw. Maybe your next programming language shouldn't be a programming language. 1989.
- Nathan Shedroff. Interfaces for understanding. In *More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure*. National Academic Press, 1997.
- Ben Shneiderman. 1.1 direct manipulation: a step beyond programming languages. *Sparks of innovation in human-computer interaction*, page 17, 1993.
- Markus Won, Oliver Stiemerling, and Volker Wulf. Component-based approaches to tailorable systems. In *End user development*, pages 115–141. Springer, 2006.

