iUCP: Estimating Interactive-Software Project Size with Enhanced Use-Case Points

Nuno Jardim Nunes and Larry Constantine, University of Madeira

Rick Kazman, University of Hawaii

// An empirical study shows that estimations based on a modified use-case-point method exhibit less interestimator variance than those based on the original method. //

IN RECENT YEARS, the software engineering (SE) and human-computer interaction (HCI) communities have tried to combine their methods and techniques. Such cross-fertilization is difficult because each community works independently. Although practitioners often participate in multidisciplinary teams, a lack of communication still exists. Software developers often fail to recognize mature, successful user-centered design techniques from the HCI community—for instance, user roles and personas, human-activity modeling, and contextual inquiry and design.[1] Although these techniques tackle major SE issues (requirements and user involvement), too few practitioners understand them, and they're still far from experiencing large-scale adoption.

We've been exploring one area of cross-fertilization for both disciplines: how to produce more consistent software project size estimations based on *use-case points* (UCPs) by exploiting *usage-centered design* (usageCD). Usage-centered design differs from user-centered design in that it puts uses rather than users at the center of design and changes the prime objective from enhancing user experience to enhancing user performance.[2] In particular, we've modified the UCP method to make it appropriate for agile development of interactive software; we call our version Interactive UCP (iUCP).

UCPs

Researchers have proposed several functional size measurement methods and cost estimation models, notably function point analysis (FPA)[3] and COCOMO.[4] Both assume that developers can derive size measurements and estimates from historical project data and current project characteristics.

With object orientation, use cases emerged as a dominant technique for structuring requirements. This technique was integrated into the Unified Modeling Language (UML) and Unified Process and became the de facto standard for SE requirements modeling. Consequently, Gustav Karner created the UCP method, which estimates project size by assigning points to use cases in much the same way that FPA assigns points to functions.[5] The UCP model gained popularity because of its simplicity and abstraction, which make it good for early estimations. Sergey Diev[6] and Edward Carroll[7] comprehensively discuss UCPs.

Defining Actors and Use Cases

The UCP model's starting point is the standard UML definitions of actor and use case (www.omg.org/gettingstarted/what_is_uml.htm) because the UCP method focuses mainly on estimating actor and use case complexity. Furthermore, the model takes into account technical, environmental, and productivity factors.

We seek to create more consistent size estimations based on revised actor and use-case concepts. Our modifications to the UCP method result in more convergent estimates of unadjusted complexity because they rely on

0740-7459/11/$26.00 © 2011 IEEE

CALCULATING USE-CASE POINTS

Use-case point (UCP) calculations aren't complicated; the central problem is defining the UCP model's elements (actors and use cases) and assigning weights to them. Sergey Diev discussed issues related to weighting actors and use cases; he argued that to obtain reasonably accurate estimates, we must clarify these concepts across and within projects.[1]

Estimation relies on the quality of the underlying use-case model.[2] The estimates' quality depends on consistent application of the heuristics across and within projects. Edward Carroll described how a multiteam organization used UCP to accurately estimate project cost early during software development.[3] He also explained how the organization evaluated metrics to ensure the UCP model's accuracy. Ayman Issa and his colleagues discussed how use-case representations of requirements don't directly map to the structures that project managers use.[4] This failure leads to ongoing comparisons of individual costs that are subjective and often don't represent final project expenditures.

References

1. S. Diev, "Use Cases Modeling and Software Estimation: Applying Use Case Points," *ACM Software Eng. Notes*, vol. 31, no. 6, 2006, pp. 1–4.
2. K. Vinsen, D. Jamieson, and G. Callender, "Use Case Estimation: The Devil Is in the Detail," *Proc. 12th IEEE Int'l Requirements Eng. Conf.*, IEEE CS Press, 2004, pp. 10–15.
3. E.R. Carroll, "Estimating Software Based on Use Case Points," *Proc. 2005 Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 05), ACM Press, 2005, pp. 257–265.
4. A. Issa, M. Odeh, and D. Coward, "Software Cost Estimation Using Use-Case Models: A Critical Evaluation," *Proc. 3rd Int'l Conf. Information and Comm. Technologies: From Theory to Applications* (ICTTA 06), vol. 2, IEEE Press, 2006, pp. 2766–2771.

well-defined, less ambiguous definitions of actors and use cases that we obtain through the usageCD method. The technical, environmental, and—particularly—productivity factors that adjust the complexity still depend on historical data from past projects but don't depend on the actor and use case models.

Estimating UCPs

The UCP method first determines the *unadjusted actor weight* (UAW). For each actor in the use-case model, the method attributes a weight factor:

- *Simple actors* (a weight factor of 1) are system actors that communicate through an API.
- *Average actors* (a factor of 2) are system actors that communicate through a protocol or data store.
- *Complex actors* (a factor of 3) are human actors that interact normally through a GUI or other human interface.

The total UAW is the weighted sum of all the actors.

The UCP method also attributes a weight factor for each use case with reference to the scenario that leads to the state originally anticipated by the user (success scenario):

- *Simple use cases* (a factor of 5) involve a simple UI or simple processing and only one database entity. The success scenario involves three or fewer transactions and five or fewer class implementations.
- *Average use cases* (a factor of 10) involve moderately complex UIs and two or three database entities. The success scenario involves four to seven transactions and five to 10 classes.
- *Complex use cases* (a factor of 15) involve complex UIs or processing and three or more database entities. The success scenario involves eight or more transactions and 11 or more classes.

The total *unadjusted use-case weight* (UUCW) is the weighted sum of all the use cases.

We further modify unadjusted UCPs (UUCPs) to reflect a project's complexity and its developers' experience. To do this, we weight *technical-complexity factors* (TCFs) and *environment-complexity factors* (ECFs) on the basis of the team's experience, the development platform, and other criteria depending on the context.

After estimating the UCPs, we estimate the number of project hours by multiplying the UCPs by a *productivity factor* (PF) defining the ratio of development person-hours per UCP. We base PF on past project statistics and fine-tune it through historical data. A value between 15 and 30 is typical, depending on the team's experience. The complete formula is

$$\text{UCPs} = (\text{UUCW} + \text{UAW}) \times \text{TCFs} \times \text{ECFs}.$$

For example, for a project with a UUCW of 50, a UAW of 10, 1.02 TCFs, and 1.04 ECFs,

$$\text{UCPs} = (50 + 10) \times 1.02 \times 1.04 = 63.648.$$

So, applying a PF of 20 would yield an estimate of 1,272.96 person-hours for the project.

For more on calculating UCPs, see the related sidebar.

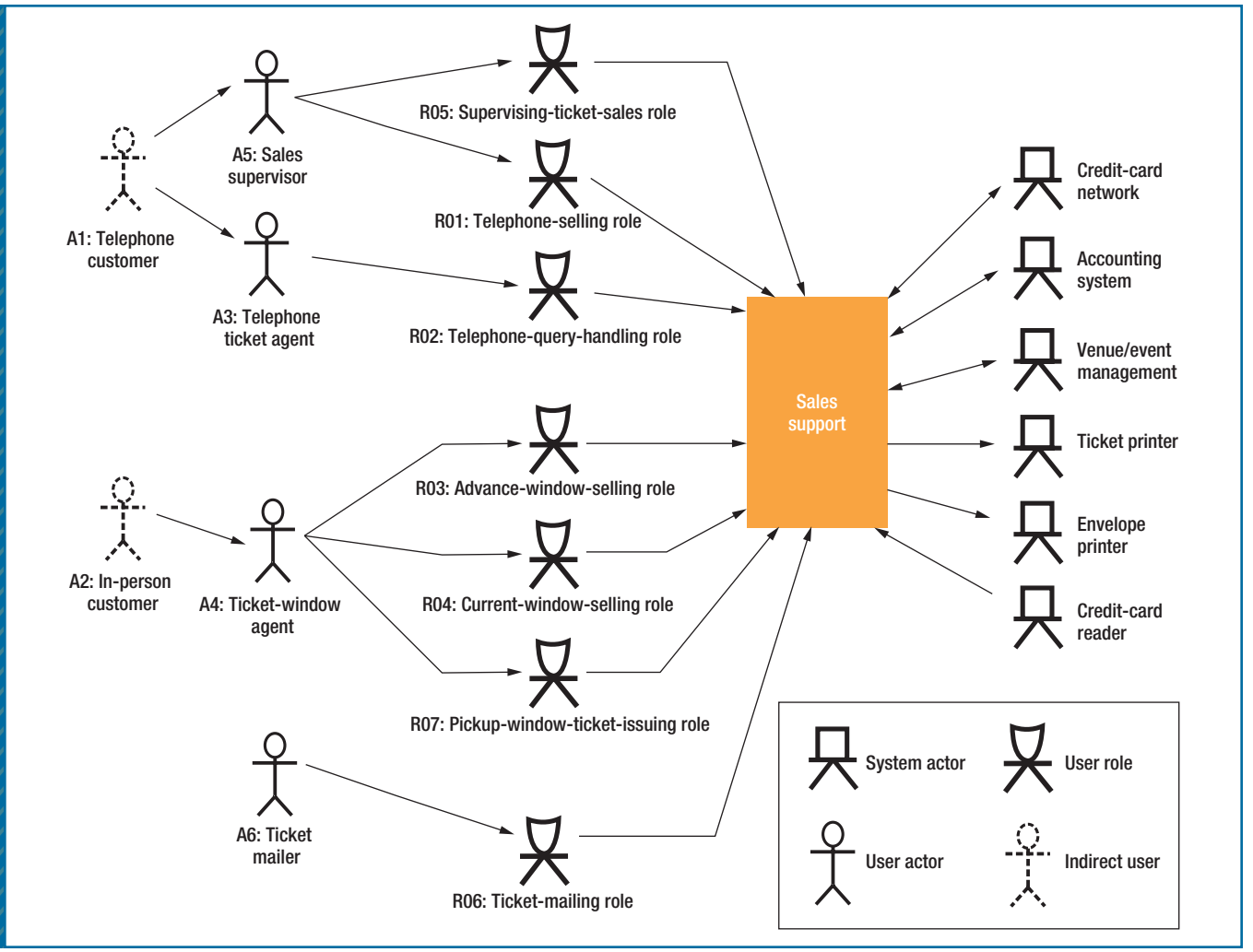

FIGURE 1. A context map for a ticketing application. From left to right, indirect and direct human actors interact through roles with the reference system (sales support). On the right, several system actors also interface with the reference system.

iUCP: Estimation in Interaction Design Projects

To explain how interaction design can influence UCP estimation,[5] we consider the model-based techniques that Larry Constantine and Lucy Lockwood pioneered[8] and others further expanded.[7,8] UsageCD provides the methodological scaffolding for applying activity theory, particularly for interactive software development. In the broader context of human-activity modeling, it provides a systematic approach to organize and represent the contextual aspects of human use of tools and artifacts.

Weighting Actors

Effective interaction design involves understanding users and their needs. As with UML, we call users who interact with a system *actors*. However, unlike UML, we expand the actor concept through user roles, representing relationships between users and a system. We can describe a role by the context in which it's performed, the characteristic manner in which it's performed, and the design criteria for the role's supporting performance.

The difference between a usageCD context map and a conventional use-case model is the richness of the information conveyed about each actor. For example, a conventional UML model would represent the ticketing problem in Figure 1 with 12 actors: six human actors (A1–6 in Figure 1) and six system actors). With the UCP method, the estimation would be four simple actors (the credit card reader, envelope printer, ticket printer, and venue or event manager), two average actors (the credit card network and accounting system), and six complex actors (A1–6).

Table 1 illustrates the weighting of the 12 ticketing application use cases

TABLE 1

Use-case point (UCP) and interactive UCP (iUCP) estimations.

Method	Actor type	Description	No. of actors	Weight factor	Weight
UCP	Simple	Defined API	4	1	4
	Average	Interactive or protocol-driven	2	2	4
	Complex	GUI	4	3	12
					20 total
iUCP	Simple system	Defined API	4	1	4
	Average system	Interactive or protocol-driven	2	2	4
	Simple human/complex system	Supports one user role	1	3	3
	Average human	Supports two to three user roles or one focal role	2	4	8
	Complex human	Supports more than three user roles or more than one focal role	1	5	5
					24 total

for the model in Figure 1. Analyzing the model, we can verify that the complexity weighting of actors discards all the role information provided by the usageCD method. Conventional UML assumptions probably wouldn't consider A1 and A2 as actors, and they would have zero weighting under the assumptions discussed previously. So in either case, only four complex actors exist, and the total estimated weight is 20.

iUCP considers the user roles and additional information that usageCD provides to inform early estimation by weighting the actors. The number of roles each actor supports provides an important way to infer the use case's complexity. Additionally, usageCD suggests the concept of *focal role* together with several relationships that constitute a model designated a *user-role map*.[11] Focal roles are central to the rest of the design process.

Our experience working and consulting on many projects that have applied usageCD suggests revised heuristics for actor weighting:

- *Simple system actors* (a factor of 1) communicate through an API.
- *Average system actors* (a factor of 2) communicate through a protocol or data store.
- *Simple human actors* (a factor of 3) are supported by one user role.
- *Complex system actors* (also a factor of 3) communicate through a complex protocol or data store.
- *Average human actors* (a factor of 4) are supported by two or three user roles or one focal role.
- *Complex human actors* (a factor of 5) are supported by more than three user roles or more than one focal role.

We can conclude from Table 1 that there's a total difference of four UCPs (from 20 to 24) on the basis of revised actor weighting (assuming both approaches zero-weight the indirect actors). Although this difference might look minor, in a real-world project with three times as many actors and roles, the impact is substantial.

Weighting Use Cases

Use cases have become ubiquitous in software development.[11] We can attribute part of their success to the concept's simplicity, but some of that success is probably also due to their imprecise definition. Entire books have discussed the definition of a use case, and we find many instantiations of use cases that vary in scope, detail, format, and style. Any estimation method relying on weighting use cases will suffer from the same uncertainty. UsageCD clearly defines use cases through the concept of *essential use cases*.[11]

Essential use cases are more abstract, generalized, and technology-free descriptions of the essence of a given problem. But they're also described in a systematic sequence of steps divided between user intentions and system responsibilities. These steps provide a systematic way to identify transactions (that we can depict using narratives or sequence or activity diagrams), which are key to classifying use cases in the

TABLE 2

A use case for withdrawing cash from an ATM.

User intention	System responsibility
Identify self	Check identity
Specify amount	Provide cash

UCP method. For example, Table 2 shows a use case for withdrawing cash from an ATM.

A conventional use-case scenario for this example (for example, from the Eclipse Process Framework wiki at http://epf.eclipse.org) would typically be classified as complex because it involves more than seven transactions. However, the essential use case would count just two essential steps (system responsibilities), making this a simple use case.

The discrepancy underlies the problems in applying UCPs across companies, teams, and projects. Interestingly, the heuristics for assigning weight factors to use cases depend on assumptions about the UI. In the UCP method, a simple use case corresponds to a simple UI, an average use case to a moderate UI, and a complex use case to a complex UI. However, conventional use cases don't reflect the division between user intentions and system responsibilities that conveys the notion of interaction (that is, interaction happens when a user specifies an intention to the system).

Essential use cases provide a systematic way to express transactions as steps in a dialogue. Originally, this type of description was intended to get at a task's essence from a user's perspective, avoiding unintended or premature assumptions about the UI. When applied to estimating use cases, it becomes an important way to retain scope and prevent the granularity problems we previously described.

Estimating transactions isn't the only concern when assigning weight factors. The heuristics specifically mention two additional criteria depending on the conceptual architecture:

- the number of entities manipulated in the use case's context and
- the number of classes implementing the use case.

The relationship between use cases and implementation classes is accomplished in UML using the entity/control/boundary pattern. However, this pattern doesn't reflect the separation of concerns that interactive system development requires. Boundary classes encapsulate interfaces to both human actors and system actors, so no clear distinction exists between human and system interaction. So, the implementation classes extracted from the use cases won't reflect the UI's complexity, which is key to assigning weight factors to use cases.

iUCP extends this original framework to include two concepts reflecting the user intentions that form the basis of usageCD: tasks and interaction spaces.[9] Task classes model the structure of the dialogue between the user and the system; they also manage task-level sequencing, multiple-interaction-space consistency, and mapping between entities and the interface. Interaction-space classes represent the space in a system's UI where the user interacts with all the functions, containers, and information needed to carry out a particular task or set of interrelated tasks. Together, the concepts of task and interaction-space classes extend the UML entity/control/boundary pattern, providing enhanced separation of concerns and enabling more consistent estimation of use-case complexity, particularly regarding the underlying human interaction.

Elsewhere, we've described how to extract software architecture from essential use cases.[12] Figure 2 highlights the process, in which task classes originate from user intentions, control and entity classes from system responsibilities, and interaction spaces from the crossing of both. This process increases traceability and is central in identifying the entities and classes required to implement a use case.

The ATM example in Figure 2 illustrates how we can use usage-centered architecture to inform the classification of use cases in iUCP. Transactions are the number of system responsibilities in an essential use case. Implementation classes are the total number of classes originating from an essential use case, as shown by the dashed lines connecting use-case descriptions to the conceptual architecture.

Table 3 shows how to apply the heuristics to the example in Figure 2. We count the number of system responsibilities and user intentions per use case and the number of originating implementation classes. This contrasts with the uncertainty surrounding a conventional use case: not only is isolating transactions more difficult, but there's also little guidance regarding the number of implementation classes corresponding to each use case.

Empirical Evaluation

To evaluate the impact of using usageCD for use-case estimation, we developed an empirical experiment with master's students taking the University of Madeira's human-centered software engineering course. The course exposed students to the concepts of usageCD and had them develop a group project in teams of four. On average, approximately 30 students participated (20 SE and 10 HCI students).

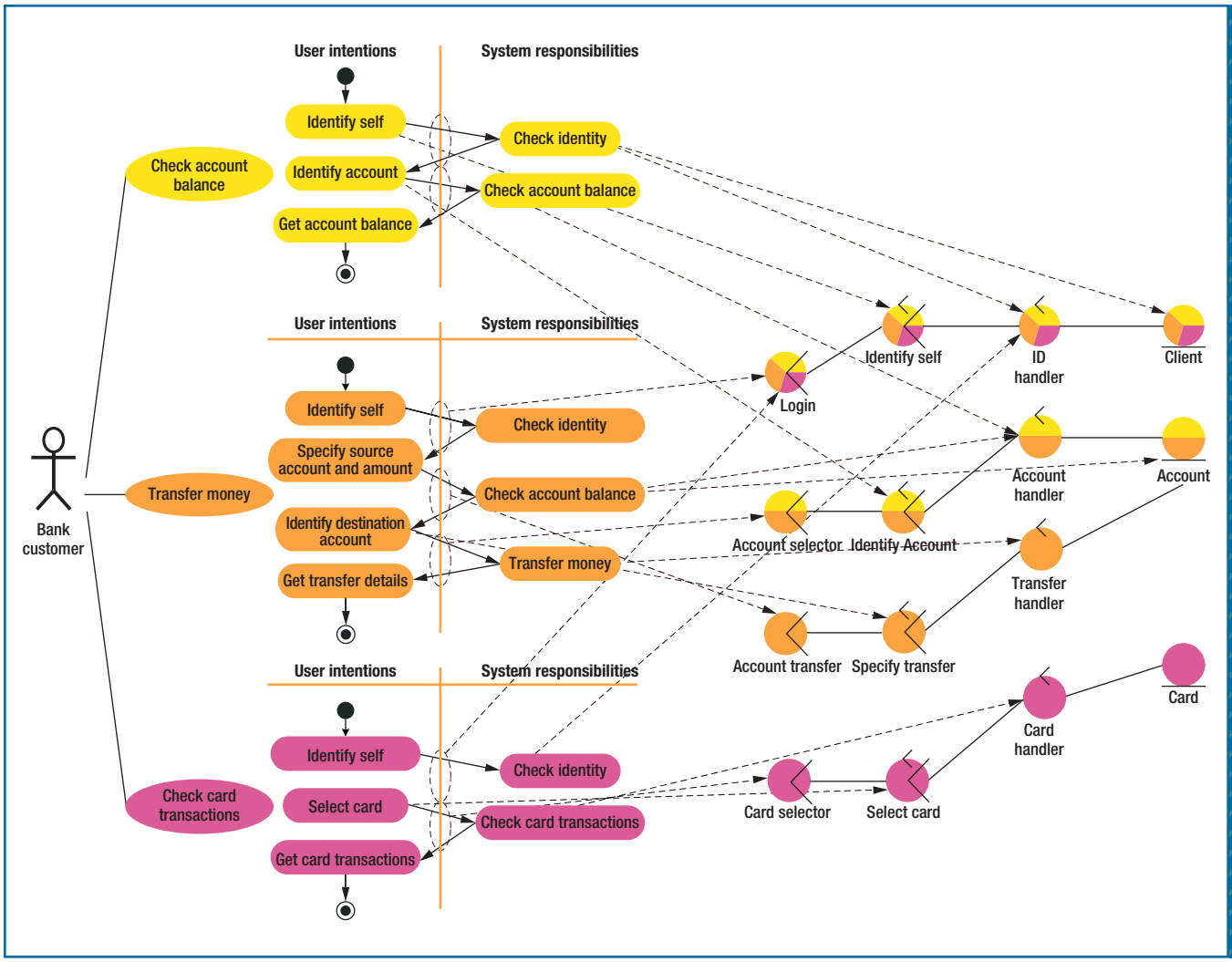

FIGURE 2. A conceptual architecture extracted from essential use cases.[12] This simple model refers to an ATM system involving one actor and three use cases. Each use case is detailed with an essential task flow described in terms of user intentions and system responsibilities. The right side depicts the architecture extracted from the use cases.

Most students had a computer science background and experience developing moderate-sized software systems. Some HCI students had backgrounds in design or the social sciences but were always grouped with CS students.

Over two consecutive years, the students worked on the same project. They had to model and prototype a computer-based controller for managing a videoconferencing facility. Here are two short excerpts from the project brief:

TABLE 3

Estimation based on iUCP for the ATM example.

Use-case type	Description	No. of use cases	Weight factor	Weight
Simple	Simple UI, 1 entity, ≤3 transactions	0	5	0
Average	Average UI, 2-3 entities, 4-7 transactions	1	10	10
Complex	Complex UI, >3 entities, >7 transactions	2	15	30

TABLE 4

Modeling estimates from each group of students.

Method	Group	No. of activities	No. of actors	No. of user roles	No. of artifacts	No. of use cases	No. of interaction spaces	No. of tasks	No. of controls	No. of entities
UCP	1	8	7	4	4	10	7	11	5	4
	2	10	5	5	5	23	17	23	8	8
	3	6	2	4	6	14	11	12	9	13
	4	14	8	7	4	9	8	9	6	5
	5	16	8	7	5	38	16	45	13	5
	6	10	7	9	7	19	13	13	8	8
	7	6	3	7	5	7	10	11	6	5
	Average	10.0	5.7	6.1	5.1	17.1	11.7	17.7	7.9	6.9
	Standard deviation	3.8	2.4	1.9	1.1	10.8	3.8	12.9	2.7	3.1
	Variance	14.7	5.9	3.5	1.1	117.1	14.6	165.6	7.1	9.8
iUCP	1	7	12	4	3	14	11	13	5	3
	2	9	8	2	2	13	7	16	6	6
	3	8	6	7	7	6	8	8	8	6
	4	9	6	3	6	17	6	27	17	10
	5	8	5	4	3	18	10	11	8	8
	6	5	4	2	7	8	8	18	7	7
	7	3	7	3	4	16	6	6	6	4
	Average	7.0	6.9	3.6	4.6	13.1	8.0	14.1	8.1	6.3
	Standard deviation	2.2	2.6	1.7	2.1	4.6	1.9	7.1	4.1	2.4
	Variance	5.0	6.8	3.0	4.3	20.8	3.7	49.8	16.5	5.6
	F-test		0.867	0.848		0.054		0.169		

A presentation area at the front of the room faces several rows of seats behind desks. For each pair of seats there is a press-to-talk/press-to-release microphone with an LED that indicates when it is active. Loudspeakers are located at the front and sides of the room.

There are two video cameras, one at the front of the room, the A camera, facing toward the audience, and one at the back, the B camera, facing front toward the presentation area. The cameras are mounted on motorized gimbals and are equipped with motorized zoom lenses.

The project aimed to provide a simple, efficient video controller system interface. Students had to design

TABLE 5

Calculated actor and use-case weights for the UCP method and iUCP.

Method	Group	Unadjusted actor weight	Unadjusted use-case weight	Unadjusted UCPs
UCP	1	6	60	66
	2	12	195	207
	3	34	125	159
	4	14	110	124
	5	8	235	243
	6	15	140	155
	7	13	55	68
	Average	14.6	131.4	146.0
	Standard deviation	9.2	66.2	66.3
	Variance	84.0	4,381.0	4,391.3
iUCP	1	20	75	95
	2	10	130	140
	3	28	55	83
	4	15	110	125
	5	12	118	130
	6	11	75	86
	7	14	85	99
	Average	15.7	92.6	108.3
	Standard deviation	6.3	27.2	22.9
	Variance	40.2	739.6	525.9
	F-test	0.393	0.048	0.021

something practical using existing technology and the available programming resources. The students had two months to complete the project and had to present all the models prescribed in usageCD and estimate their project using the UCP method or iUCP.

For empirical evaluation, we gave 14 distinct groups of five students the same project (seven groups in one academic year and an additional, distinct, seven groups in the following academic year). The first seven groups modeled the system using usageCD and produced a UUCP estimation on the basis of the UCP method. The second seven groups modeled the same system using the same models but generated an iUCP estimation. We hypothesized that the students' unadjusted complexity estimates of actors and use cases would have less variance using iUCP than using the UCP method.

During the project, the students developed several models independently. At the end, we inspected all their models. Table 4 summarizes the information we collected from the models (that is, the table lists the factors that students used to generate their estimations).

To verify whether the data from both groups followed a normal distribution, we performed the Shapiro-

ABOUT THE AUTHORS

NUNO JARDIM NUNES is an associate professor of computer science and the president of the Madeira Interactive Technologies Institute (Madeira-ITI) at the University of Madeira. His research interests include service design, bridging software engineering (SE) and human-computer interaction (HCI), and methods and tools for agile software development. Nunes has a PhD in HCI and SE. He's a member of the ACM and SIGCHI. Contact him at njn@uma.pt.

LARRY CONSTANTINE is a professor at the University of Madeira's Department of Mathematics and Engineering and an Institute Fellow with Madeira-ITI. His research interests include safety-critical interaction and model-driven design. Constantine has an SB in management from MIT. He's an ACM Fellow, a member of the Usability Professionals' Association and the IEEE Computer Society, and 2009 winner of the Stevens Award. Contact him at lconstantine@uma.pt.

RICK KAZMAN is a professor at the University of Hawaii's Department of Information Technology Management and a visiting scientist at the Software Engineering Institute. His research interests include software architecture, design and analysis tools, software visualization, software engineering economics, and human-computer interaction. Kazman has a PhD in computational linguistics from Carnegie Mellon University. Contact him at kazman@sei.cmu.edu.

Wilk test. For group 1, $p = 0.636$; for group 2, $p = 0.291$. In addition, both groups' box plots showed no outliers. To determine the quality of the variances between the two samples, we applied the F-test. Because both the UCP method and iUCP depend primarily on the number of actors and use cases, we compared the numbers in both groups. Apart from actors and roles, the variance in use cases differed significantly ($p = 0.054$).

Table 5 summarizes the calculated UAW and UUCW for the UCP method and iUCP. Comparing the variances of the calculated UAW and UUCW shows improved results. As we expected, the F-test for the UAW isn't statistically significant ($p = 0.40$) but the differences for UUCW ($p = 0.048$) and UUCP ($p = 0.021$) are statistically significant.

Our results show that using iUCP produces size estimations more consistent in their estimation of use-case complexity and overall UCP unadjusted complexity. This supports our hypothesis that by using iUCP, the students' unadjusted complexity estimates of actors and use cases would have less variance.

Early estimation of software size is critical. Our approach not only helps bridge the gap between SE and HCI but also provides software developers with systematic guidance to produce quality early estimates for software. It's increasingly important to find ways to enable both HCI and SE experts to collaborate early in the life cycle. By employing usageCD techniques such as user roles, essential use cases, and interactive conceptual architectural models, we not only bridge the gap but—what's more important—also illustrate how HCI techniques can improve software estimates and models.

Combining SE and HCI provides new opportunities for collaboration between interaction designers and software developers. This helps developers see the advantage of using HCI techniques early on. Conversely, interaction designers can better understand their models' impact and recognize UI elements' impact at the architecture level, building common ground for other activities such as prioritizing development and planning releases.

We built the iUCP on statistical data from usageCD projects collected over several years. However, a systematic evaluation would require more extensive data collection and analysis over a longer period of time. But our purpose here isn't to prove the estimation method's validity; other researchers (for example, Edward Carroll[7]) have covered this topic. Our modifications of the UCP method are minimal, letting us preserve the original model's integrity. Our goal with iUCP is to help software developers and interaction designers apply heuristics that are suitable for interactive applications and that work consistently across and within projects.

References

1. A. Seffah and E. Metzker, "The Obstacles and Myths of Usability and Software Engineering," *Comm. ACM*, vol. 47, no. 12, 2004, pp. 71–76.
2. L. Constantine, "Beyond User-Centered Design and User Experience," *Cutter IT J.*, vol. 17, no. 2, 2004.
3. A.J. Albrecht, "Measuring Application Development Productivity," *Proc. Joint Share, Guide, and IBM Application Development Symp.*, IBM, 1979, pp. 83–92.
4. B.W. Boehm et al., *Software Cost Estimation with COCOMO II*, Prentice-Hall, 2000.
5. G. Karner, "Resource Estimation for Objectory Projects," Rational Software, 1993.
6. S. Diev, "Use Cases Modeling and Software Estimation: Applying Use Case Points," *ACM Software Eng. Notes*, vol. 31, no. 6, 2006, pp. 1–4.
7. E.R. Carroll, "Estimating Software Based on Use Case Points," *Proc. 2005 Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 05), ACM Press, 2005, pp. 257–265.

8. LL. Constantine and L.A.D. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Longman, 1999.
9. N.J. Nunes and J.F. Cunha, "A Software Engineering Method for Small Software Development Companies," *IEEE Software*, vol. 17, no. 5, 2000, pp. 113–119.
10. N.J. Nunes and J.F. Cunha, "Wisdom—Whitewater Interactive System Development with Object Models," *Object-Oriented User Interface Design*, M. van Harmelen, ed., Addison-Wesley, 2001, pp. 197–243.
11. L. Constantine and L. Lockwood, "Structure and Style in Use Cases for User Interface Design," *Object-Oriented User Interface Design*, M. van Harmelen, ed., Addison-Wesley, 2001, pp. 245–279.
12. N. Nunes, "What Drives Software Development: Bridging the Gap between Software and Usability Engineering," *Human-Centered Software Engineering*, Springer, 2009, pp. 9–25.