End-User Development

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor

Context

- In software development, there has always been a tension between those who know **what** must be developed (**subject-matter experts**, **end users**, etc.) and those who know **how** to develop it (the **software developers**).
- **End-user development (EUD)** aims to solve this problem by empowering end users to **design** and/or **customize** the **user interface** and **functionality of software**.
- EUD “scales out” software development activities by **enabling a much larger pool of people** (the **end users**) to participate.

Challenges of EUD

- EUD is inherently **different** from traditional software development, and trying to support EUD by simply **mimicking traditional SE approaches** is often **insufficient** to produce successful results.
  - ❖ End users usually **do not have training** in professionals' programming languages, formal development processes, or modeling and diagramming notations.
  - ❖ End users often **lack the time or motivation to learn** these traditional techniques.
  - ❖ End users usually write code in order to achieve a **short- or medium-term goal** rather than to create a durable software asset that will produce a continuing revenue stream.

Objectives of EUD

Providing appropriate **tools**, **social structures**, and **development processes** that are **highly usable**, **quickly learned**, and **easily integrated into domain practice** (Burnett & Scaffidi).

*"EUD is 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)

Lieberman, H., Paterno, F., and Wulf, V. (eds.) 2006. End User Development, Springer.

The birth of EUD

- **Spreadsheets** were **the first major EUD programming environment**, beginning with VisiCalc, then continuing with Lotus 1-2-3 and Excel.

VisiCalc circa 1980. *(Courtesy of Dave Winer)*

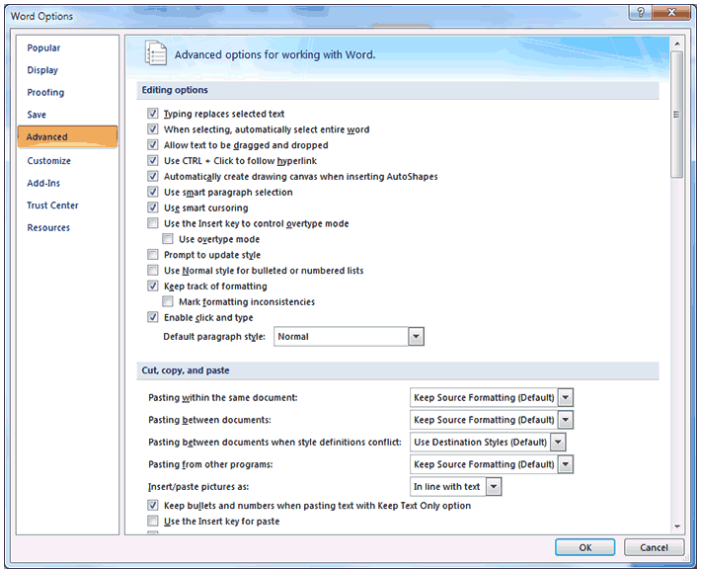
The birth of EUD

- Users of spreadsheets **may not think** of themselves as "**doing programming**".
- **Spreadsheets**, however, are indeed **programming environments** because their **formulas** are **first-order functional programs** (Jones et al., 2003).
  - The **formulas** can refer to **input "variables"** (cell names) and the results of the formulas are **computed output values**.

Tailoring

- **Tailoring** is any “activity to **modify** a computer application **within its context of use**” (Won et al., 2006).
- At the most basic level, **tailoring** encompasses **specifying/adjusting parameters** to an existing application in a way that **changes its behavior** at a **high level of granularity**.
- Once tailoring begins to involve **creating full-fledged programs** in order to **extend the functionality** of an application, the activity seamlessly encompasses **end-user programming**.

Tailoring



Basic-level tailoring

```
Sub PasteSpecial()
Selection.PasteSpecial DataType:=wdPasteText
End Sub
Sub SmallCaps()
'
' SmallCaps Macro
'
'
    With Selection.Font

        .SmallCaps = True

    End With
End Sub
Sub Subscript()
'
' Subscript Macro
'
'
    With Selection.Font
        If (.Superscript) Then
        .Superscript = False
        .Subscript = False
        ElseIf (.Subscript) Then
        .Subscript = False
        .Superscript = True
        Else
        .Subscript = True
        .Superscript = False
        End If


    End With
End Sub
```

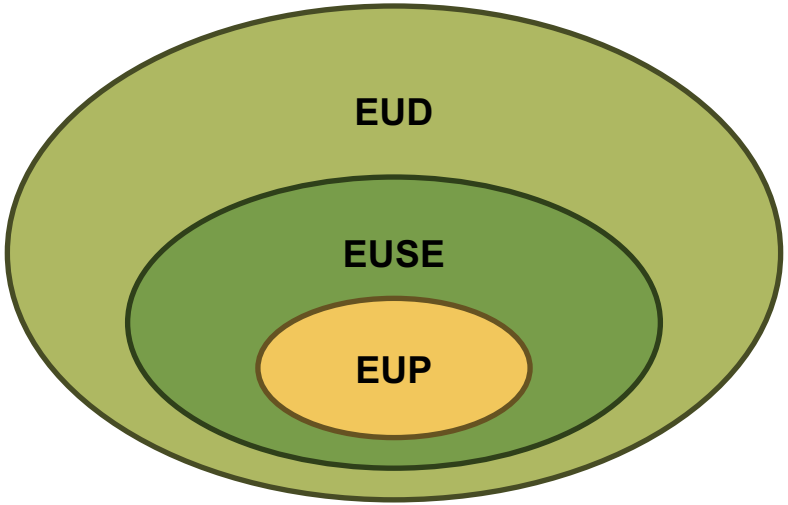
Full-fledged program: Macros in MS Word

EUD, EUP, and EUSE

- **EUD** overlaps with two similar concepts: **end-user programming (EUP)** and **end-user software engineering (EUSE)**.
- **EUP** enables end users to **create their own programs** (Ko et al., 2011).
- The difference between EUP and EUD is that **EUD** methods, techniques, and tools **span the entire software development lifecycle**, including **modifying** and **extending** software, not just the “create” phase.
- **EUP** is **the most mature** from a research and practice perspective.

EUD, EUP, and EUSE

- **EUSE** is a **more recent** area and emphasizes on the ***quality*** of the software end users create, modify, or extend; thus its research **focuses on methods, techniques, and tools that promote the quality** of such software.

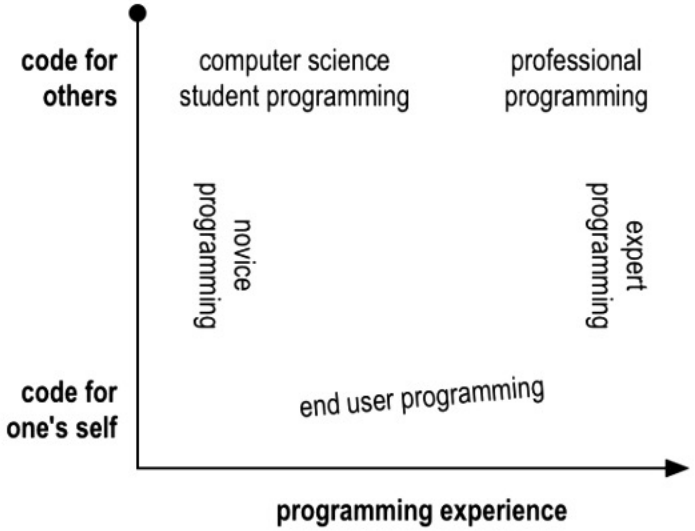


End-user programming (EUP)

End-user programming (EUP)

- **EUP** is defined as *"programming to achieve the **result of a program** primarily for **personal, rather than public use**"* (Ko et al., 2011).
- In **EUP**, the (end-user) developer's goal is to actually ***use*** the program.
  - ❖ This contrasts with **professional programming**, where the goal is to **create a program for other people to use**, often in exchange for monetary compensation.
- The programs created through **EUP** can be **extensions of existing applications** (e.g., macros), or **new applications that run separately** from existing applications.
- End users can perform **EUP** through a wide range of **interaction styles**.

End-user programming (EUP)



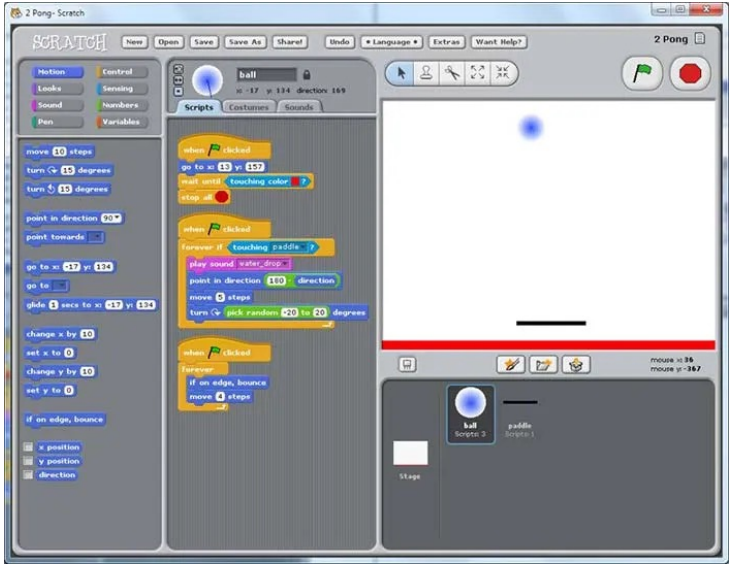
Source: Ko et al. (2011).

End-user programming (EUP): **Interaction Styles**

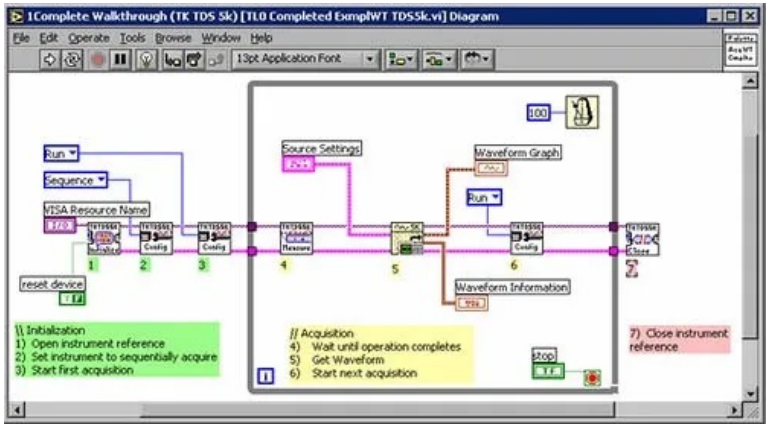
Programming using visual attributes

- At least some of a program's **semantics** is expressed through the **visual layout** of the program.
- For example, the **grid-like arrangement of cells in a spreadsheet** carries a certain **semantics**:
  - ❖ Cells that are vertically or horizontally aligned with one another are part of a composite object **defined solely based on the visual layout of cells**.
- In a **visual language**, **semantics** can be encoded in many **attributes of a visual representation**, such as position, color, size, and intersection with other shapes.

Programming using visual attributes



Scratch (Resnick et al., 2009)

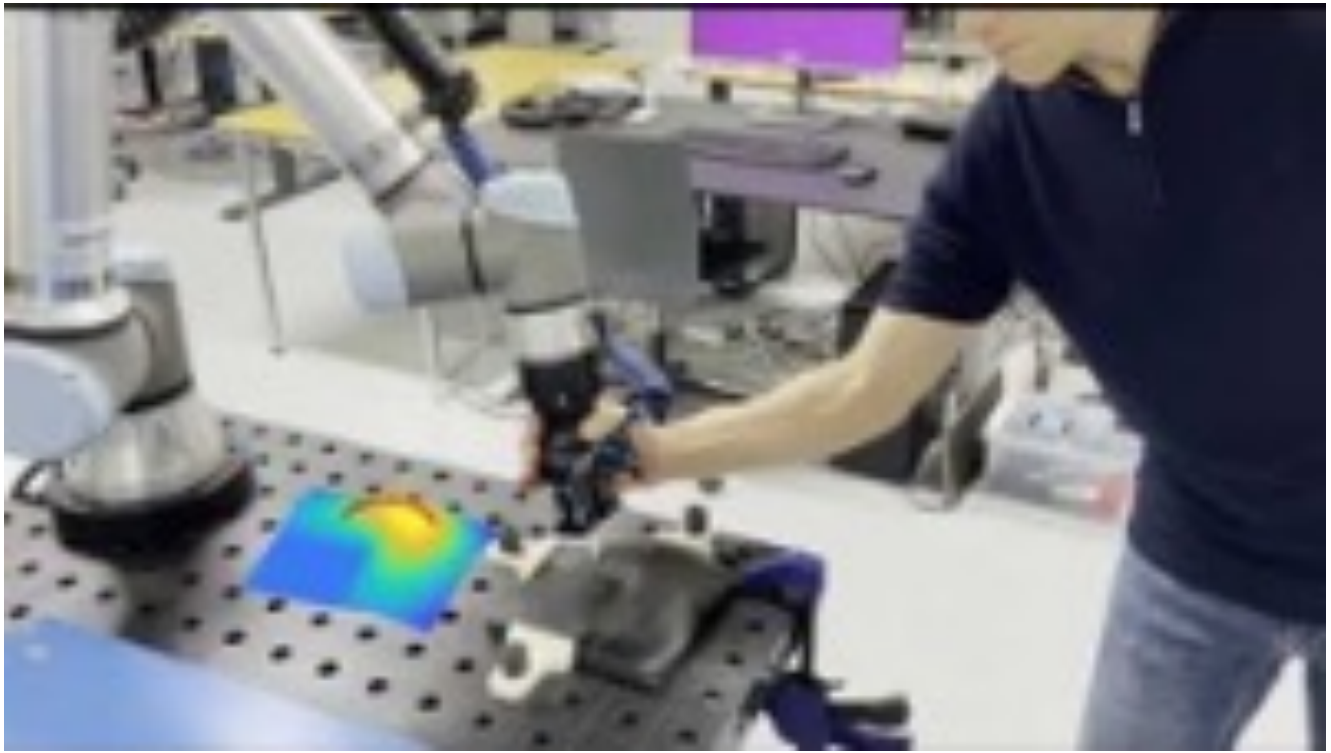


LabVIEW programming language for creating circuit simulations and other programs.

Programming-by-demonstration (PBD)

- Programming-by-demonstration (PBD), sometimes called *programming-by-example*, is a programming technique whereby **the user demonstrates** the new **program's logic** (or provides some examples), from which the programming environment **infers** a program representing that logic.
- Some PBD systems are able to **deductively infer** the **entire program**, while **others deduce what they can** and **ask the user for help** for the rest.
- A typical problem is to represent the final program in a form useful to the user to enable the end-user developer to **review, test, and debug the program**.
  - **PBD** is often used **in combination with visual or textual languages**.

Programming-by-demonstration (PBD)



Programming-by-example (PBE)

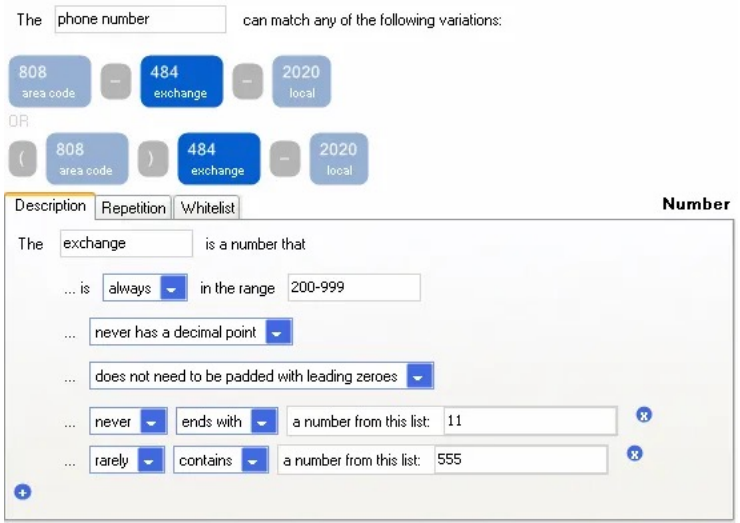
	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

FlashFill in MS Excel 2013 (Gulwani, 2011)

Programming-by-specification

- In programming-by-specification, the user **describes** a desired program, and a tool then **generates** the program for the user.
  - For example, natural language → Python (Liu and Lieberman, 2005).
- A key limitation of this approach, as with inference-based PBD approaches, is that it is difficult for a user to **predict what program will be generated** from any particular input.
- Domain-specific languages (**DSLs**) and **forms-based visual interfaces** are useful to make the **bounds** of a tool's input language more obvious to users.
  - Users' specifications are **restricted** to only those that can actually be handled by the tool.

Programming-by-specification

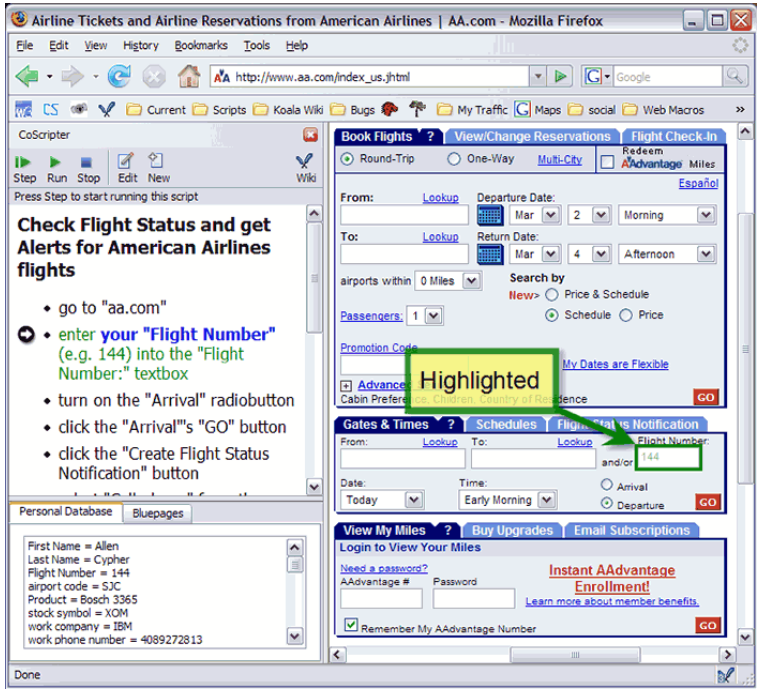


Visual specification of what a phone number looks like; from this specification, **the tool generates code** that can check whether a particular string matches the specification (Scaffidi 2009).

Programming with text

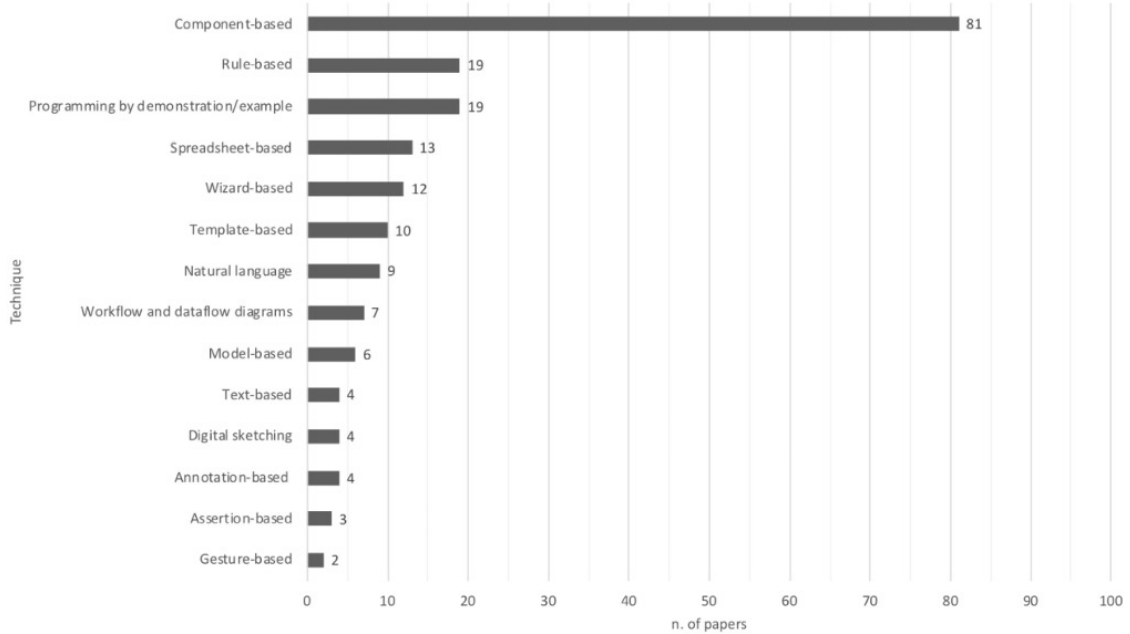
- It's **the most traditional** interaction technique for programming.
  - ❖ For a time, some believed that this style of programming would not be appropriate for EUP.
- Most programming environments that support other interaction styles **also include text to some extent**.
- Despite the proliferation of alternative interaction styles, **text remains widely used** because of its **conciseness** and **effectiveness** for communicating abstract concepts.

Programming with text



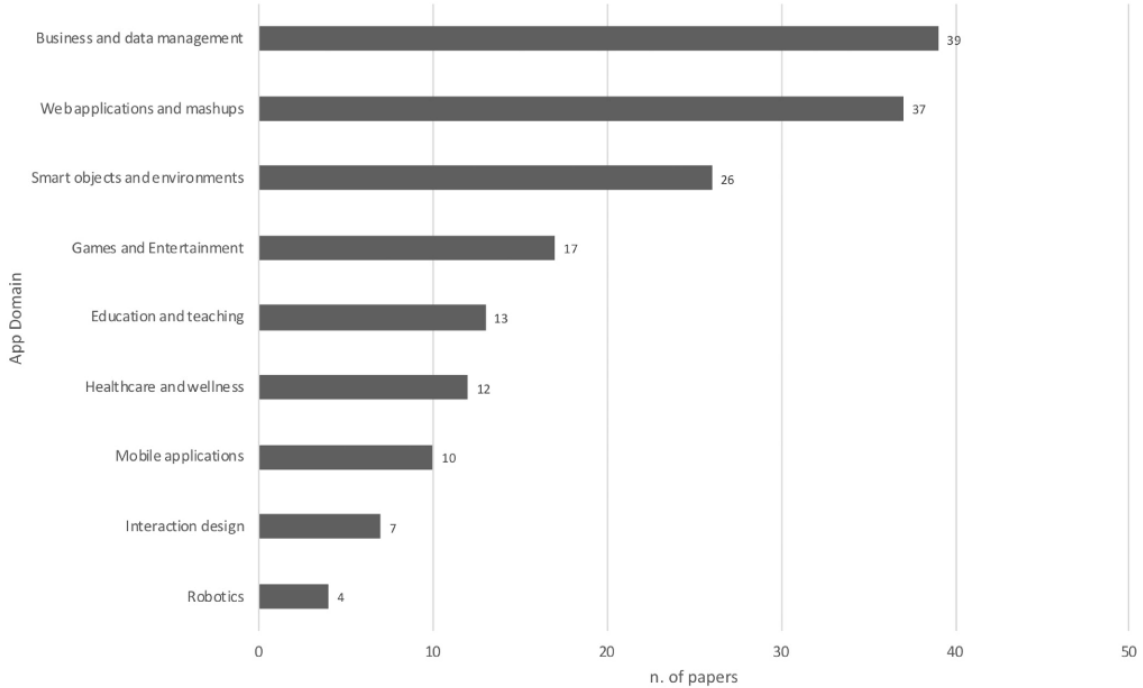
CoScripter programming tool to edit a web macro (Scaffidi et al., 2010).

Frequency of proposed techniques



Source: Barricellia et al. (2019)

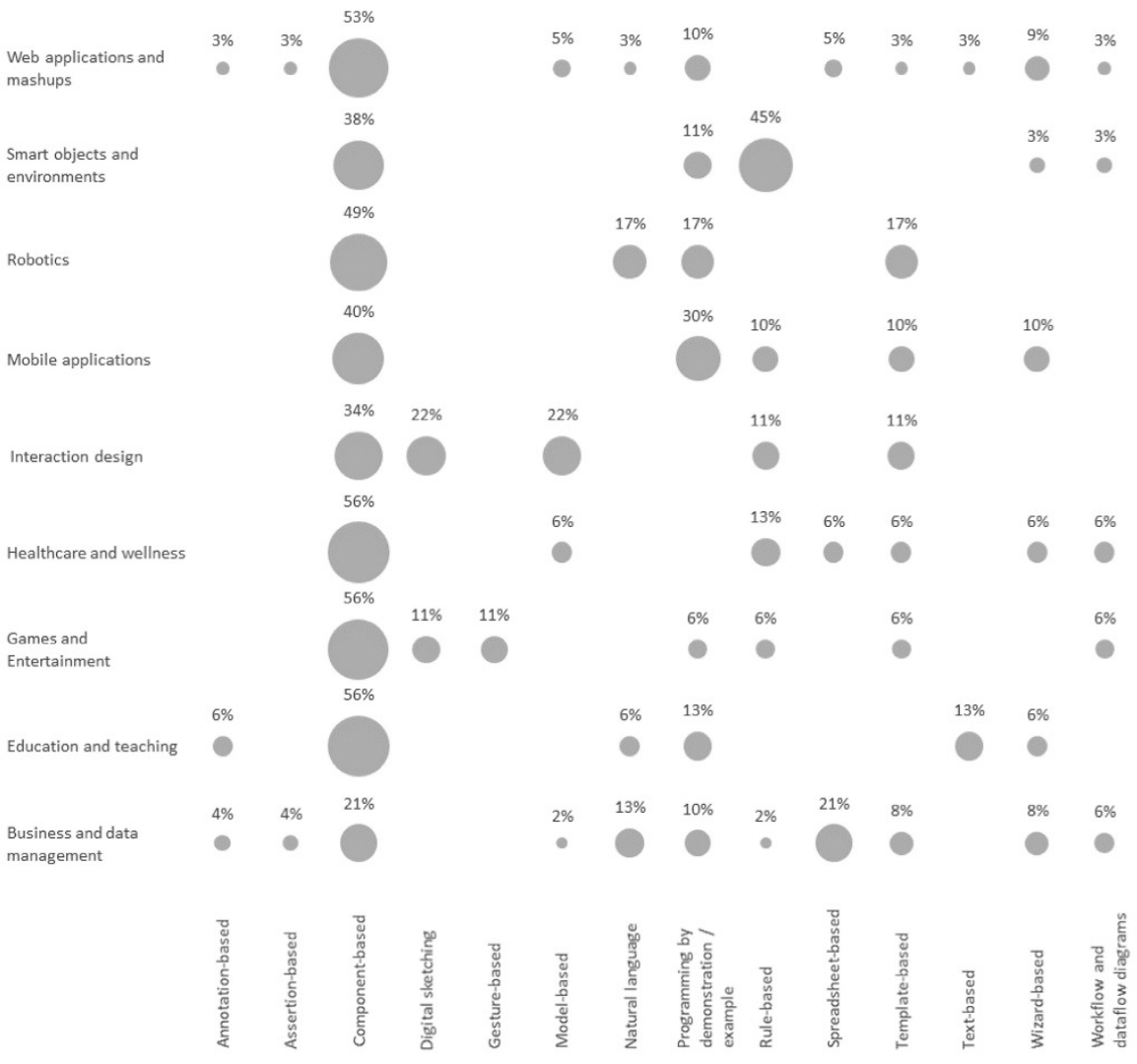
Typical application domains



Source: Barricellia et al. (2019)

Mapping of techniques to application domains

Source: Barricellia et al. (2019)



SDU

End-user software engineering (EUSE)

End-user software engineering (EUSE)

- **EUSE** is defined as *"end-user programming involving **systematic** and **disciplined activities** that address **software quality** issues"* (Ko et al., 2011).
- Attention to quality is important because **poorly-written software** can cause **data loss**, **security breaches**, **financial loss**, or even **physical harm**.
  - ❖ **Even when** the software is created by **end-user developers**.
- The software qualities relevant to **EUSE** are **the same** as those of interest to professional developers who sell their products.
  - ❖ Reliability, performance, maintainability, reusability, privacy, security, etc.

End-user software engineering (EUSE)

Table II. Qualitative Differences Between Professional and End-User Software Engineering

Software Engineering Activity	Professional SE	End-user SE
Requirements	explicit	implicit
Specifications	explicit	implicit
Reuse	planned	unplanned
Testing and Verification	cautious	overconfident
Debugging	systematic	opportunistic

Source: Ko et al. (2011).

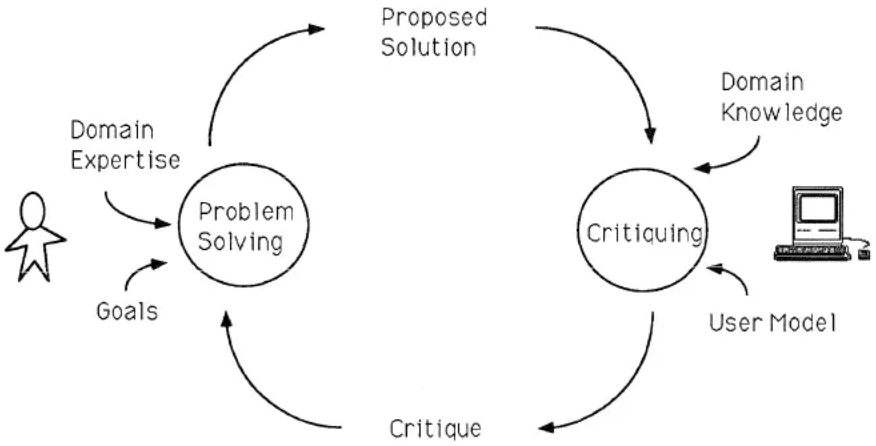
Requirements and design

- Examples of requirements (goals) in EUD include (Ko et al., 2011; Blackwell, 2004; Blackwell, 2006; Rosson et al., 2002):
  - **Personalizing** the way that an application or computer behaves
  - **Automating** time-consuming tasks
  - **Performing computations** that are **hard to do** accurately **by hand**
  - **Communicating** information

Requirements and design

- Professional developers are expected to investigate, document, and refine requirements before they start to design or code an application.
- In contrast, end users often **live in their domain** every day and **know it very well**, so they often already have an idea about the requirements, and **do not do any extra work** to arrive at them, document them, or check them.
  - ❖ This makes end-user developers to jump directly into coding **without taking the time** to **document** their requirements or **look for inconsistencies**.
  - ❖ This also makes end-user programmers' requirements to be rather **emergent** and **tightly intertwined with design**.
  - ❖ Adapted **design partners** have been investigated for EUD (Diaz et al., 2008).

Requirements and design

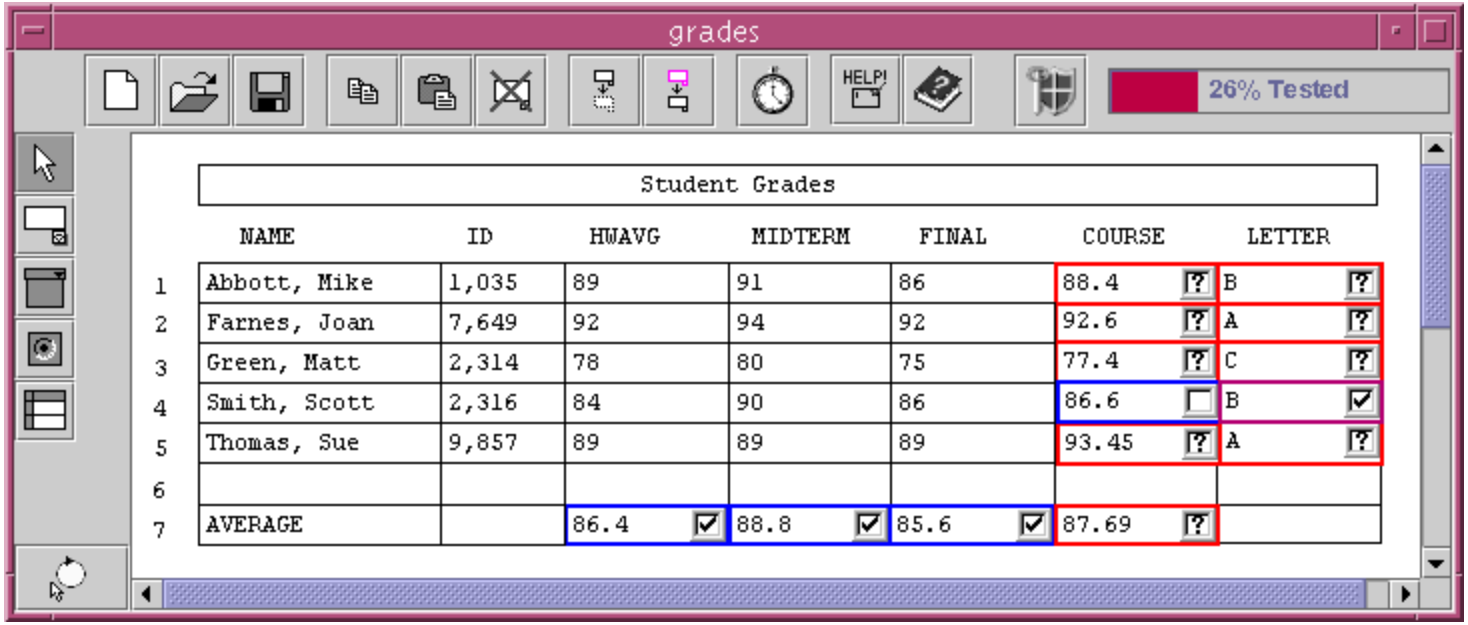


The design critic process (Fischer et al., 1990).

Verification and validation

- **Testing** is the most common approach for **verification and validation (V&V)**, even among professional developers.
- The most developed end-user testing approach is “What You See Is What You Test” (**WYSIWYT**) for **systematically testing spreadsheets** (Fisher et al., 2006).
  - ❖ It employs a “Surprise-Explain-Reward” strategy (Wilson et al., 2003): ***surprises*** such as **colored borders attract users’ attention** to areas of the spreadsheet that need testing, and **tool tips explain the colors’ meaning** and the **potential reward** in using the testing devices.

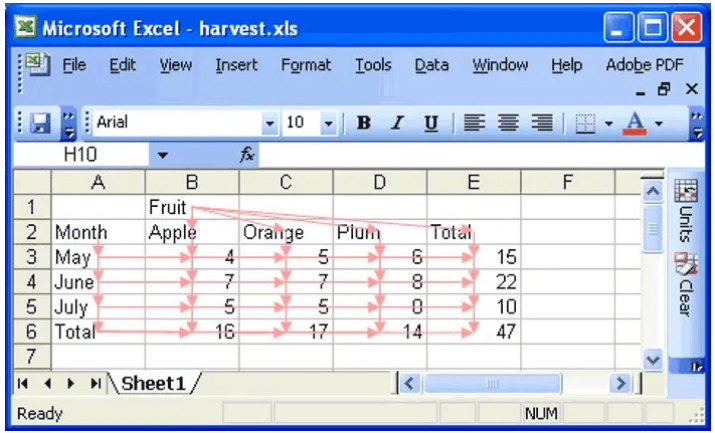
Verification and validation



WYSIWYT approach, where **checkmarks** indicate testedness, **question marks** indicate that a cell needs testing, and **coloured borders** indicate correctness.

Verification and validation

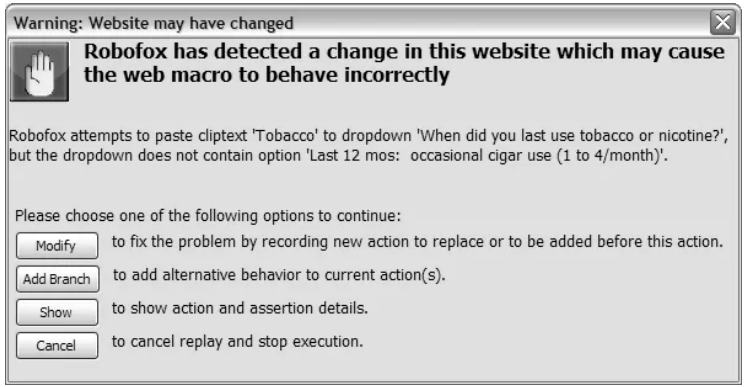
- Another common approach is to automatically **look for errors on the basis of types, dimensions, or units** (Erwig and Burnett, 2002; Abraham and Erwig, 2007; Coblenz et al., 2005; Chambers and Erwig, 2009).
- This approach can be regarded as specific kinds of **assertions**.



UCheck

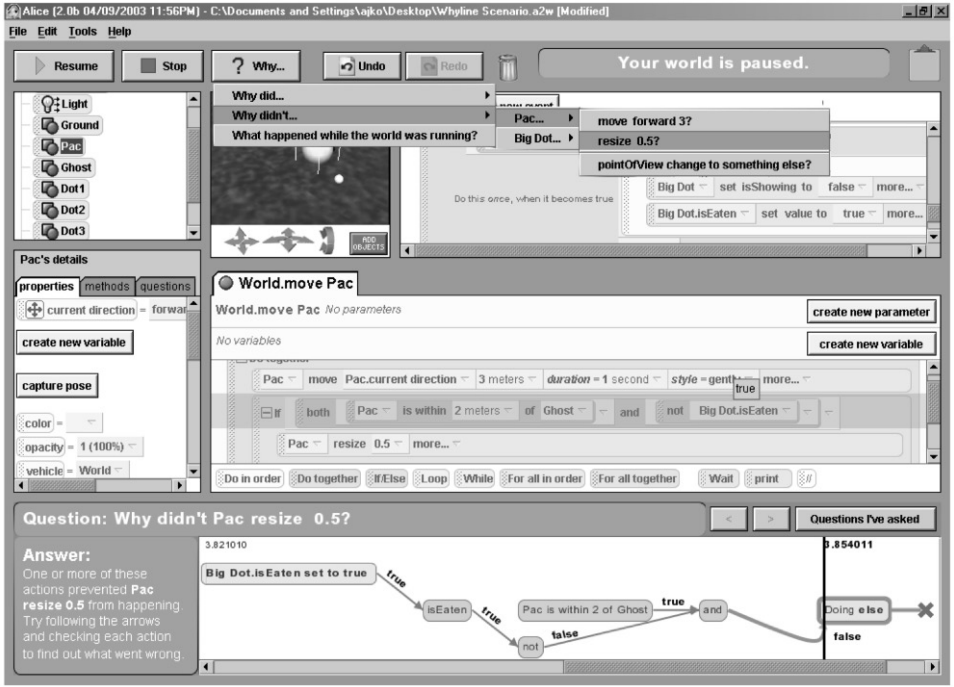
Debugging

- Some of the **debugging techniques** used by professional developers have been **adapted for use in EUP tools**.
- Several **EUP** tools provide tight integration between **testing and debugging**.
  - ❖ For example, **assertions** can be inserted **proactively** when a program is created, in order to perform automatic **tests** and initiation of **debugging** if an assertion fails.



Popup window asking user to indicate whether and how a **Robofox** web macro should be modified due to a **violated assertion**.

Debugging



The **Whyline** for Alice. The user has asked **why** Pac Man **failed to resize** and the answer shows a **visualization of the events** that prevented the resize statement from executing (Ko and Myers, 2004).

Reuse

- Supporting reuse of end-user programs is challenging because **end-user developers rarely have** the **opportunity** or **training** required to **design highly reusable programs**.
  - The **reuse** of end-user developers' programs can **propagate errors** across an organization (Mackay, 1990).
- In **EUD**, finding, reusing, and even sharing code becomes **more opportunistic**, as the goals of reuse are more to save time and **less to achieve other software qualities**.

Reuse

- Like in professional SE, prior work on **reuse for EUD** has largely focused on **components** and **APIs**.
  - ❖ Consequently, many of the **challenges** that professionals face, **end users face as well**.
- However, while APIs designed for professional use often focus on optimizing flexibility, **end users** often **need much more focused support** for achieving their domain-specific goals.

Future and implications of EUD

- As **users** continue to **grow in number and diversity**, **EUD** is likely to play **an increasingly central role in shaping software** to meet the broad, varied, rapidly changing needs of the world.
  - ❖ Emergence of **low-code development platforms**.
- With the continually broadening scope and power of **EUP**, substantial additional **attention to quality will become increasingly crucial**.
- As a result, the **fit** between **software's form** and **individual users' needs** might be **closer** than has been possible before, **vastly increasing the usefulness of software in peoples' lives**.

End-User Development

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor