Requirements Engineering Tools
Academic and Industrial Perspectives

Lund University, 21.4.2015
Many researchers create tools that have tremendous value and potential way beyond their initial setting.

- Spreading and maintaining these tools is difficult and time consuming.
- The CSG aims at providing services to all of DTU Compute to help them improve their tools and make them available to the world.
- See csg.compute.dtu.dk for more.
RE – a relevant topic?

- Let’s have a quick look at Google Trends for a first impression.
Requirements are a key factor

Reasons for failure:
- Insufficient user involvement: 12.4%
- Inadequate planning: 8.1%
- Unrealistic expectations: 9.9%
- Lack of management support: 9.3%
- Insufficient resources: 10.6%
- Others: 20.4%
- Obsolete features: 7.5%
- Requirements change: 8.7%
- Incomplete requirements: 13.1%

Reasons for success:
- Clearly set requirements: 13.0%
-Management support: 13.9%
- User involvement: 15.9%
- Others: 19.2%
- Well defined interfaces and responsibilities: 5.3%
-Qualified personnel: 7.2%
-Manageably sized project phases: 7.7%
-Realistic expectations: 8.2%
-Reasonable project planning: 9.6%

Requirements are a key factor

[Standish Group & Scientific American]
More (reliable) sources

- Requirements Engineering (or the lack thereof) is still the single most important reason for poor software quality
  - Lutz [1993] showed that 60% of errors in critical systems were the results of requirements errors.
  - Espiti [1996] conducted a survey of European companies and found that more than 60% of them considered requirements engineering problems as very significant.
  - Hall et al. [2002] carried out a case study of 12 companies at different CMM levels. They discovered that, out of a total of 268 development problems cited, almost 50% (128) were requirements problems.”

- „Nonetheless, requirements engineering is still performed in an intuitive and chaotic way.”

RE in the age of Agile

- Classic RE approaches are often associated with sequential development processes (“waterfall”), and sometimes frowned upon by proponents of lightweight (“agile”) methods.

- It is important to acknowledge, however, that the majority of concerns and techniques re-popularized in “agile” contexts are indeed concerned with requirements.
  - Test first
  - User stories
  - Customer on-site
  - Incremental releases
  - Backlog/grooming
  - Kanban-stages/buckets
Assoc. Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)
COMPUTE Software Group

Industrial Perspective on RE Tools
Tool Usage in RE

- Identify user requirements
- Test the software
- Model user requirements
- Document software systems
- Evaluate project feasibility
- Learn to use new tools
- Train staff
- Other

"I hate to be a cynic, but there are hardly any worthwhile tools. The overhead in learning to use them is too great for the payoff."

Luisa Mich, Mariangela Franch, Pierluigi Novi Inverardi: Market research for requirements analysis using linguistic tools
**Tools**

Do you use any tool supporting requirements analysis and top-level design?

<table>
<thead>
<tr>
<th></th>
<th>1–5</th>
<th>6–20</th>
<th>21–50</th>
<th>51–100</th>
<th>More than 100</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yes</strong></td>
<td>16%</td>
<td>18%</td>
<td>33%</td>
<td>33%</td>
<td>51%</td>
</tr>
<tr>
<td><strong>No</strong></td>
<td>84%</td>
<td>82%</td>
<td>67%</td>
<td>67%</td>
<td>49%</td>
</tr>
</tbody>
</table>

Luisa Mich, Mariangela Franch, Pierluigi Novi Inverardi: *Market research for requirements analysis using linguistic tools*
State of RE in Practice is Poor

“There is a lot of information available on solid RE practices but anecdotal evidence still indicates poor practices.”

Improving RE Process Maturity is easy

- Here are some examples of the practices by maturity level defined by the REAIMS RE process maturity framework.

- Basic
  - 3.1 Define a standard document structure
  - 4.3 Identify and consult system stakeholders
  - 6.2 Use language simply, consistently and concisely
  - 8.2 Organize formal requirements inspections

- Intermediate
  - 4.10 Prototype poorly understood requirements
  - 9.6 Define change management policies

- Advanced
  - 10.6 Specify systems using formal specifications
  - 10.8 Collect incident experience

Prose for Requirements Engineering

- Alternatives exist, that can (mostly) replace NL, as various case studies have demonstrated.

**Language Type Usage**

- 5% Controlled
- 16% Structured
- 79% Plain

- Natural Language Processing (NLP) and Information Retrieval (IR) technology can do amazing things:
  - generating sequence diagrams from natural language use case descriptions;
  - generating class diagrams from NL requirements specifications.

- However, if the performance is less than perfect, using tools is often worse than not using them.
Concurrent Requirements Stores

- In typical industrial settings, five to eight different media are used to store requirements.

Stefan Winkler: *Information Flow Between Requirement Artifacts. Results of an Empirical Study*  
Requirements Flow

Stefan Winkler: Information Flow Between Requirement Artifacts. Results of an Empirical Study
Industrial Perspective

- Plain text prevails as the major RE “formalism”.

- Existing tools are expensive and poor.
  - They are used out of despair or regulatory torture.
  - Problems include ease of use, cost, and lack of (obvious) benefit.

- Many long standing problems are still open.
  - Transition to design phase, tracing
  - Integration in “lightweight” approaches
  - Team collaboration, version control
  - Effort/Cost estimation
  - Requirements validation

- Academia is sitting in an ivory tower.
Assoc. Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)
COMPUTE Software Group

Academic Perspective on RE Tools
Should Academia Create Tools?

- **Should we as scientists really create tools?**
  - After all, we’re academics, and our job is research and teaching, and creating products is really the job of industry, isn’t it?
  - Also, we don’t have the resources to create tools with the degree of polishing required.

- **I believe that there are just three ways we as academics can impact the industrial practice of software development.**
  - Educate students better – but we’re doing that already as best as we can.
  - Conduct targeted research through industrial co-operations.
  - Create and publish tools that address practical needs.

- **Actually, tools are instrumental to SE research.**
  - Conceptual research without validation is not any longer *de rigueur*.
  - Tools are essential for running case studies.
  - Tools that implement novel concepts are *embodied hypotheses*. 
Academia Perspective on RE Tools

- Academia cannot hope to create “a better DOORS”.
  - We should not attempt to, either—we should leap-jump industry.

- There are many clever ideas & algorithms. Here are just few examples:
  - Natural Language processing (checking of style/grammar, document outline)
  - Scenario enactment for validation
  - Effort estimation based on Function Points
  - Model Version Control to support group collaboration
  - Trace-preserving transition to design

- These contributions are used neither in industry nor commercial tools.

- In order to achieve any kind of adoption, academic ideas will have to satisfy three conditions.
  - They must be nicely wrapped – people are spoilt rotten by visual bling.
  - They must provide overkill benefit – acceptance must be a no-brainer.
  - They must address bread and butter features – no matter how booooring.
  - Advanced features must be fully automatic – no training/knowledge needed.
Enactment

- Scenarios for Use Cases and Persona descriptions can be validated through enactment.
  - With a formal scenario structure, text to speech processing can create an interesting effect.
  - Enactment can be done without tool support, as a “design game”.
## Function Point Estimation

- **Function Point Analysis (FPA)** allows cost estimation of use cases.
- FPA is not routinely used in practice, despite solid evidence in their favor.
- There is very little literature linking them to RE.
- The topic is usually not taught in academic courses, and the tools don’t support it.
Design Inspections

- Inspections are a rather old QA technique that is particularly suitable for early phases.
  - There is solid evidence to support the cost-effectiveness of code inspections.
  - There is very little literature on or guidelines for the application of inspections to requirements, or analysis-level models.
  - The topic is usually not taught in academic courses, and the tools don’t support it.
Visual Editors

- Many people like to complement their textual requirements by some visualization, with the aim to
  - provide better overview (over a set of goals and their relationships, say);
  - illustrate their otherwise dreary and boring texts; or
  - provide an alternative view to better explain what they mean.

- Observe that these drawings have a different status than both UML diagrams and fully informal doodles.
Group Collaboration

- **Requirements specification is typically a team activity**
  - Classical problems of distribution/replication, locking/version control arise.

- **Unfortunately, existing tools do not fit the case of RE very well.**
  - Collaborative editing in Word, Google Docs, CMS, and Wikis support prose-like textual data spread out over a number of files.
  - Collaborative programming projects use VCS’s like CVS, Continuous, SVN, GIT, etc. for many small text files in a fairly static overall structure.
  - In collaborative modeling, a single large DB (e.g., XMI-file) is created that captures a graph-like structure.

- **Requirements have unique characteristics, though:**
  - More than one person, but not that many either.
  - Requirements exhibit characteristics of text and graphs.
  - Conventional VC methods for code are not suitable
The collaboration support in RED is made up of two feature sets.

- On the Client, RED offers multi-file projects with diff/merge by files/element.
- On the Server, RED offers a visual version history that focuses on major development activities rather than (small) individual commits of data files.

<table>
<thead>
<tr>
<th>Context &amp; Challenges</th>
<th>Realization &amp; Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case</strong></td>
<td><strong>Files</strong></td>
</tr>
<tr>
<td>Coding</td>
<td>$10^k$</td>
</tr>
<tr>
<td>Editing</td>
<td>1 (k?)</td>
</tr>
<tr>
<td>Modeling</td>
<td>1</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>1...k =Editing</td>
</tr>
<tr>
<td></td>
<td>1? =Modeling</td>
</tr>
<tr>
<td></td>
<td>1...k Editing + Modeling</td>
</tr>
</tbody>
</table>

where $k$ is a small constant (maybe around 3...8)
Trace-respecting A/D-Transition

- Translating a (big) textual requirements specification into a design model is difficult.
  - The translation as such is difficult and requires guidance and expertise.
  - There are bound to be many decisions, some are genuine design decisions, others result from weaknesses of the specification (ambiguity, omissions, ...)
  - Domain experts and clients typically do not understand the design level language (e.g., UML), so they cannot validate the translation outcome.
  - Trace links must be established manually.

- Idea: Translating individual requirement is much easier.
  - Each requirement is translated into a small model fragment (lenient syntax).
  - The translation as such uncovers errors by change-of-perspective.
  - The resulting fragments are then woven automatically.
  - Weaving diagnostics and manual inspection of the result uncover errors.
  - Traces are generated automatically.
**Requirement MLC9a**
Guest readers may inspect suggestions in the wishlist system.

**Requirement MLC9b**
Guest readers may inspect suggestions.

**Requirement MLC9c**
Book suggestions may be inspected.
Here is an excerpt from the LMS requirements specification, and how the features described may be captured as models.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Model Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLC 2</td>
<td>Librarians may add, update, and delete corpus items manually.</td>
<td><img src="Corpus.png" alt="" /></td>
</tr>
<tr>
<td>MLC 4</td>
<td>Librarians and Readers may post and inspect media they think should be acquired by the library to a public “wish list” indicating the status of the wish and the originator.</td>
<td><img src="Acquisitions.png" alt="" /></td>
</tr>
<tr>
<td>MLC 10</td>
<td>A librarian can do all a reader can do; a reader can do all a guest reader can do.</td>
<td><img src="wistlist.png" alt="" /></td>
</tr>
</tbody>
</table>
Weaving Fragments Establishes Traces

**DERIVE & CLARIFY**

- $R_1$ → $F_1$
- $R_2$ → $F_{2.1}$
- $R_3$ → $F_{2.2}$
- $R_4$ → $F_3$

**WEAVE**

- $F_1$ → $F_{2.1}$
- $F_{2.1}$ → $F_{2.2}$
- $F_{2.2}$ → $F_3$

**CONSOLIDATE**

- $F_{2.2}$ → $F_3$
- $F_3$ → $F_{2.2}$

**FORWARD TRACING**

- $R_1$ → $F_1$
- $R_2$ → $F_{2.1}$
- $R_3$ → $F_{2.2}$
- $R_4$ → $F_3$

**BACKWARD TRACING**

- $F_3$ → $F_{2.2}$
- $F_{2.2}$ → $F_{2.1}$

**Eastern Philosophy**

- $F_{2.1}$ → $F_{2.2}$
- $F_{2.2}$ → $F_3$
Assoc. Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)
COMPUTE Software Group

The Requirements Editor RED
The Requirements Editor RED

- **RED is a stand-alone tool for requirements engineering**
  - RED is based on Eclipse RCP, and was developed mostly by students as their final thesis project (MSc, some BSc) plus some paid hands.
  - Development has started in September 2011, a major re-engineering took place in 2013. Currently (04/15), we are preparing version 3.0.

- **RED is intended as a tool to support teaching**
  - We aspire to maximize conceptual clarity and coherence, while offering a comprehensive and practical toolbox with some cutting-edge features.
  - The tool aspires to be conceptually consistent, in itself (UI, meta-model) and with regards to the course material (slides, case studies, samples, guidelines).

- **Development goes on, a first public release is scheduled for 09/15.**
  - RED now consists of over 1,860/650 classes (hand-written/generated) and over 114,000 Lines of Code (Java).
  - The last major components (Collaboration Server) are close to completion.
  - The main focus has shifted to quality rather than adding new features.
  - One of the next steps is Bootstrapping, i.e. documenting RED in RED.
RED Features

- Features in RED 3.0 (3/2015)
  - Goals, stakeholders, visions
  - Textual & multimedia requirements
  - Informal requirements, assumptions
  - Use cases, test cases
  - External document integration
  - Personas, storyboards
  - Scenarios, enactment, Text2Speech
  - Use case points effort estimation
  - Cost/benefit annotation & analysis
  - Full cross-referencing glossary
  - UML Model Fragments
  - Browsing, searching, and sorting
  - Reporting, exporting, importing
  - Multi-file projects, Merging
  - Inspection support, locking
  - Traceability, manual change history
  - Visual modeling (Use Cases, Goals)
  - Model fragments weaving

- 3.1 (9/2015)
  - Online collaboration server
  - Dynamic web service extensions
  - More visual modeling (all of UML)
  - Dynamic view filtering

- 3.2 (3/2016)
  - Quantitative risk management
  - Features, Issues, Bugs
  - Releases, release planning
  - AHP prioritization

- Future Work (Options)
  - More file formats (ReqIF, XLSX,...)
  - CNL/Pattern checker
  - semi-automatic text-to-model translation
  - formal methods for checking
  - Mobile elicitation device
Release Plan

0  1  2  3  4

3/2011
Tool Infrastructure
Stakeholders, Vision,
Goals, Requirements

3/2014
Multi-file projects
Function Points
Scenario Enactment

3/2015
Refactoring
Project/File Merging
Basic Visual Editors

3/2017
Tech. & Conceptual
Consolidation

3/2013
Tech. Consolidation
- Re-Engineering
- Building, Versioning
- Releases, Issues

3/2014
Concept Consolidation
- User Interface
- Metamodel

9/2015
Issues, Bugs, Ideas
Release Planning,
Risks & Priorities

9/2016
Visual Editors for
Process & Structure
Models

3/2016
Collaboration Server
Weaving & Tracing
More Visual Editors

RED Release History and Plan
Status 2015-03-10

http://www.compute.dtu.dk/~hsto/tools/red.html
https://hsto@bitbucket.org/hsto/red.git
Conclusion
Academic Tool Development

- Developing practical tools is possible.

- Developing practical tools is not easy.
  - Considerable effort and time for polishing is needed – it’s a long shot.
  - But IT is a people industry – and we have the most valuable resource.

- Developing practical tools is useful.
  - Such a tool can serve as a proof-of-concept platform for individual ideas, it can be the basis for case studies, provide students with a realistic project environment...

- Creating large scale software is the topic of Software Engineering.
  - First we should get it right ourselves. Then, we should help scientists outside of SE to get their large scale developments right.
  - That is the aspiration of the COMPUTE Software Group.

- Therefore, in SE, developing tools should be accepted as a scientific contribution per se – not just for proof-of-concept.
Upcoming Elections for ACM Europe Council

The ACM Europe Council aims to increase the level and visibility of ACM activities across Europe, e.g.

- fostering the visibility and relevance of ACM in Europe, and
- encouraging greater participation of Europeans in all dimensions of ACM.

Vote for me to represent Germany & Scandinavia!

http://europe.acm.org/