COMPSCI 289:
Research Seminar in Computer Science

Research in Software Engineering

Ewan Tempero
School of Computer Science

How would the world change if. . .

Software products of the same quality could be produced 50% “cheaper‡” than they are now?

‡ cost can be financial, time, humans, . . .

How would the world change if. . .

Software products of the same quality could be produced 50% “cheaper‡” than they are now?

‡ cost can be financial, time, humans, . . .

- Those purchasing software products will not have to spend so much ⇒ more discretionary funds (e.g. for hiring new people)
- Those making software products will have the savings to:
  - make more profit
  - have more discretionary funds
  - make the product better
  - make new products they couldn’t afford to make before

How important is software to you?

- If software stopped working, how would that affect you?
- What software have you used today before the lecture (2020-09-28T0900)?
  - WeChat (on your phone)?
  - Facebook (on your phone)?
  - Twitter (on your phone)?
  - TikTok (on your phone)?
  - Instagram (on your phone)?
  - SSO (on your phone)?
  - Canvas (on your phone)?
  - eMail (on your phone)?
  - News website (on your phone)?
  - Zoom (on your phone)?
  - Auckland Transport App(‡—took the bus/ferry/train to campus)
  - Various automotive software (‡—drove/got a ride to campus)
  - Your phone (e.g. to listen to music, read today's lecture slides)
  - Your phone (to actually make a phone call!)
  - ...

  (‡ *if we were having this lecture on campus)*

We don’t need any more software

- “All the software we will ever need will be written by
  ~~1970s~~[1]
  ~~1981~~[2]
  ~~early 1980s~~[3]
  ~~Now~~[4]
  ~~Tomorrow~~[5]”

We don't need any more software

- "All the software we will ever need will be written by

  ~~1970s~~[1]

  ~~1981~~[2]

  ~~early 1980s~~[3]

  ~~Now~~[4]

  ~~Tomorrow~~[5]"
- The need for software is not going to go away any time soon

[1] Smalltalk-71

[2] The Last One

[3] Various 4th Generation Languages such as LINC

[4] Javascript rules!

[5] IEEE Spectrum—Programming Without Code: The Rise of No-Code Software Development

Software Engineering Research Goal

Figuring out how to make software products of the same quality C% cheaper than they are now. (for $C > 0$)

How hard is it to build good software?

- Therac-25 (1985-1987. Several patients severely injured or killed due to radiation overdose)
- Ariane 5 (4 June 1996. Code reused from Ariane 4 caused failure to launch)
- Mars Climate Orbiter (23 September 1999. Mixed measurement units caused lose of orbiter)
- Y2K (software assuming dates come from the years 1900-1999)
- INCIS (New Zealand Police — starting in early 90s, abandoned in 1999 with lawsuits)
- LASCAD (London Ambulance System 1992. Failed after deployment — a number of deaths attributed to delays)
- Novopay 2012
- . . .
- And an unknown (but likely very large) number of smaller projects that failed, either because of significant delays in delivery, large overruns in costs, or simply not delivered at all.

Is Software Engineering really that bad?

- Transmission Gully Motorway — originally planned to be opened April 2020, now could be as late as 2023 (maybe)

Where are the costs of building software

- Build the wrong thing (e.g. get the wrong requirements, interpret them incorrectly, they change too much)
- Insufficient testing, or testing against the wrong assumptions
- Product is difficult to use correctly, leading to many “user errors”
- Choose wrong architecture (e.g. does not scale to cope with more users, is vulnerable to security attacks, changes are expensive)
- Process to release new version too expensive (e.g. many manual steps in the process, approval process too slow/error prone)
- Takes too long to make changes, even simple ones (e.g. to fix faults, deal with changes in the environment, add new features)
- Takes too long for new developers to become productive
- Developers are not productive enough
- Developers make too many mistakes
- . . .

How is Software Research Done?

- code is analysed to find out what developers actually do
- source code repositories are analysed (e.g. Git) to find out how developers do it
- developers are surveyed
- developers are interviewed
- developers are required to perform various (usually benign) tasks and watched‡ while they do them
- developers are put in fMRI machines‡
- developers are wired with EEG, GSR, and other exotic sensors‡
- theories of how to create software are developed
- tools are developed (and developers are made to use them)
- ideas about what software quality means are discussed
- theories/tools/ideas/proposals/etc are evaluated and tested
- . . .

How to manage requirements

- Joachim Karlsson, Claes Wohlin, Bjorn Regnell “An evaluation of methods for prioritizing software requirements” *Information and Software Technology* Volume 39, Issues 1415, 1998, Pages 939-947 `https://doi.org/10.1016/S0950-5849(97)00053-0`

  Some requirements should be implemented before others. Techniques for deciding which to focus on can be more or less reliable, with higher or lower cost. What’s the tradeoff? Authors prioritised requirements for a telephony system using different methods and compared reliability.

- Yu-Cheng Tu, Ewan Tempero and Clark Thomborson “An experiment on the impact of transparency on the effectiveness of requirements documents” *Empirical Software Engineering*, 21:3 June 2016. pp. 1035-1066 `http://dx.doi.org/10.1007/s10664-015-9374-8`

  Does how we present requirements affect the efficiency and accuracy with which they are interpreted.

How to use mathematics

- G. J. Holzmann, “The model checker SPIN” *IEEE Transactions on Software Engineering* vol. 23, no. 5, pp. 279-295, May 1997, `http://doi.org/10.1109/32.588521`

  How can mathematics be packaged in a tool that will help prove software correct?

How to measure software design and its quality

- T. J. McCabe, “A Complexity Measure,” in *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, Dec. 1976, `http://doi.org/10.1109/TSE.1976.233837`

  Proposes *cyclomatic complexity number*, a metric for trying to capture how “complex” a piece of code is based on the number of control-flow paths through it.

- S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994, `http://doi.org/10.1109/32.295895`

  Proposes a set of 6 metrics intended to measure attributes of object-oriented designs, such as coupling, cohesion, and inheritance, the so-called “CK metrics”

- Neville I. Churcher and Martin J. Shepperd. 1995. “Comments on ’A Metrics Suite for Object Oriented Design’ ”. *IEEE Trans. Softw. Eng.* 21, 3 (March 1995), 263265. `https://doi.org/10.1109/32.372153`

  Notes a number of problems with the definitions of the CK metrics.

- Ewan Tempero and Paul Ralph “A Framework for Defining Coupling Metrics” *Science of Computer Programming* Volume 166 2018. pp. 214-230. `https://doi.org/10.1016/j.scico.2018.02.004`

  Proposes a framework for removing the some of the problems associated with defining coupling metrics (and defined 2^{23} such metrics, perhaps demonstrating the futility of it all!)

Are our beliefs about code quality valid

- D. Lawrie, C. Morrell, H. Feild and D. Binkley, "What's in a Name? A Study of Identifiers," *14th IEEE International Conference on Program Comprehension (ICPC'06)*, Athens, 2006, pp. 3-12, https://doi.org/10.1109/ICPC.2006.51

  Which is easier to understand—code where identifiers are full words joined together, abbreviations of the words, or just the initial letters? 80 alumni and undergraduates from a university were asked to describe 12 functions based on the code.

- Ewan Tempero, James Noble, Hayden Melton "How do Java Programs Use Inheritance? An Empirical Study of Inheritance in Java Software" *22nd European Conference on Object-Oriented Programming (ECOOP)* 2008 http://dx.doi.org/10.1007/978-3-540-70592-5_28

  If inheritance is so good, it must be used a lot in good software, right? How much is inheritance actually used? Examined use of inheritance in 93 open-source Java systems.

- Tony Gorschek, Ewan Tempero, and Lefteris Angelis. "A large-scale empirical study of practitioners' use of object-oriented concepts" *International Conference on Software Engineering (ICSE)*, 2010 https://doi.org/10.1145/1806799.1806820

  3,785 developers were asked about various beliefs on the value of various concepts associated with object-oriented design

How to test software effectively and efficiently

- V. R. Basili and R. W. Selby, ”Comparing the Effectiveness of Software Testing Strategies,” in IEEE Transactions on Software Engineering, vol. SE-13, no. 12, pp. 1278-1296, Dec. 1987, https://doi.org/10.1109/TSE.1987.232881

  Which is more effective at detecting faults—reading the code, testing based on functionality (“black-box testing”), or testing based on code structure (“white-box testing”)? 32 professional developers and 42 advanced level stuents.

- M. B. Cohen, P. B. Gibbons, W. B. Mugridge and C. J. Colbourn, “Constructing test suites for interaction testing” *25th International Conference on Software Engineering, 2003* 38-48, http://doi.org/10.1109/ICSE.2003.1201186

  Sometimes faults occur because of the interactions between replaceable components, rather than within the components. But the number of interactions grows exponentially with the number of components so how to test such cases efficiently.

What stops developers from working effectively

- N. B. Moe, T. Dingsøyr and T. Dybå, “Overcoming Barriers to Self-Management in Software Teams” in *IEEE Software*, vol. 26, no. 6, pp. 20-26, Nov.-Dec. 2009, `http://doi.org/10.1109/MS.2009.182`

  Much of the advice about managing software development teams is “self management”, but is it that easy, and if not, what makes it hard?

- Ewan Tempero, Tony Gorschek, and Lefteris Angelis. “Barriers to refactoring” *Communications of the ACM* 60, 10 (October 2017), 5461 `https://doi.org/10.1145/3131873`

  If refactoring is such a great idea, why isn’t it being done more?

What are developers *really* thinking

- Norman Peitek, Janet Siegmund, Chris Parnin, Sven Apel, Johannes C. Hofmeister, and Andr Brechmann. “Simultaneous measurement of program comprehension with fMRI and eye tracking: a case study” *12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM ’18)* 2018 pp110. `https://doi.org/10.1145/3239235.3240495`

  fMRI tells us what activity there is in the brain. Eye tracking tells us what the developer is looking at. Perhaps between the two we can confirm whether so-called “beacons” in program comprehension really exist.

- Minas R.K., Kazman R., Tempero E. (2017) “Neurophysiological Impact of Software Design Processes on Software Developers” *Augmented Cognition. Enhancing Cognition and Behavior in Complex Human Environments* (AC 2017) `https://doi.org/10.1007/978-3-319-58625-0_4`

  Some designs are considered better than others, in part because they take “less work” when performing tasks (such as modifications). Do EEG measurements, which can indicate cognitive load, show this?

How to do good software engineering research

- B. A. Kitchenham, T. Dyba and M. Jorgensen, “Evidence-based software engineering” *26th International Conference on Software Engineering* 2004, pp. 273-281, `http://doi.org/10.1109/ICSE.2004.1317449`

  What does it mean to “Evidence-based software engineering”?

- Paul Ralph and Ewan Tempero “’Construct Validity in Software Engineering Research and Software Metrics” 22nd International Conference on Evaluation and Assessment in Software Engineering, June 2018. `https://doi.org/10.1145/3210459.3210461`

  How do you demonstrate whether or not your proposed metric actually measures what you say it does?

Key Points

- Software enables the world
- $\Rightarrow$ improving any aspect of how software is created will have a significant impact on the world

COMPSCI 289:
Research Seminar in Computer Science

Research in Software Engineering

Ewan Tempero
School of Computer Science