THE UNIVERSITY of EDINBURGH
informatics

Coursework Feedback - Part 1

Björn Franke
Embedded Software

Overview

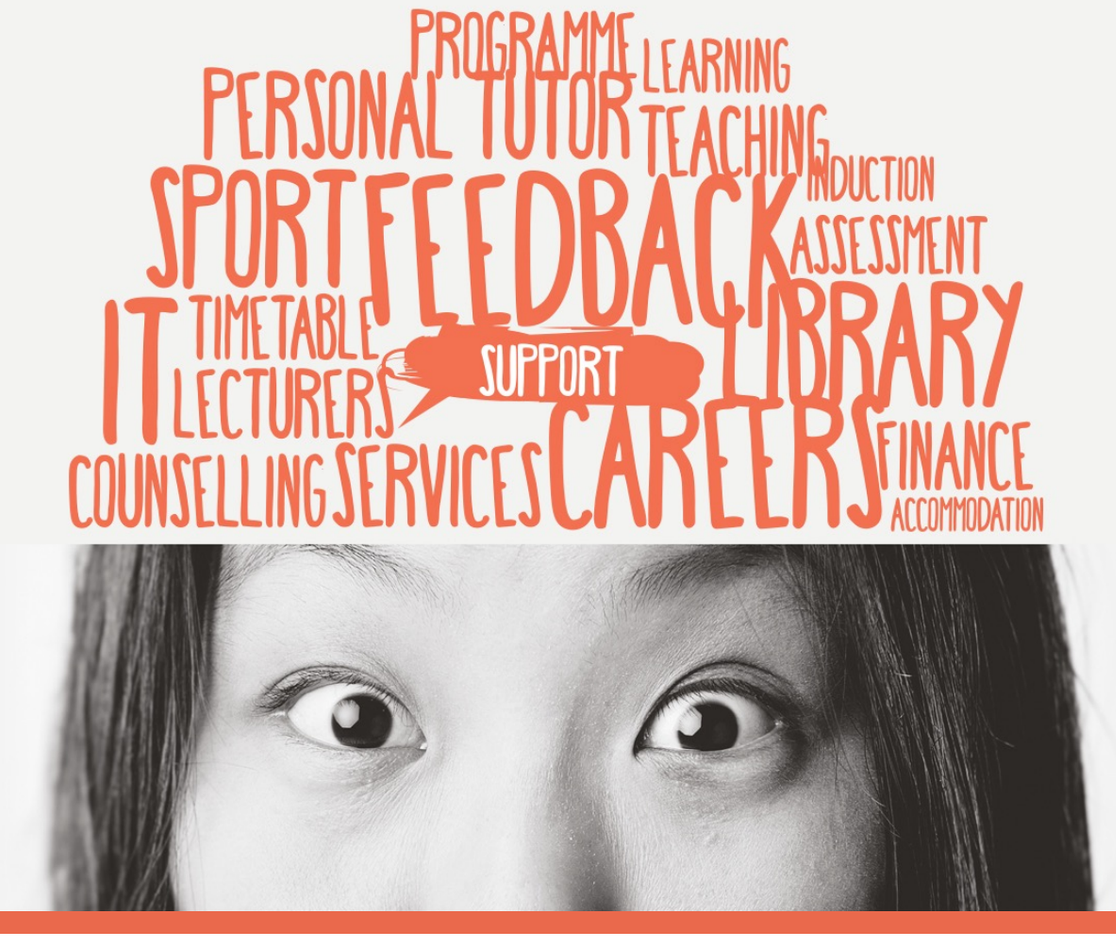
- Surveys
- Coursework Feedback
  - General Considerations
  - General Observations
  - Common Problems
- PhD Positions

Surveys

Various Surveys

- **National Student Survey (NSS), closes April 30, 2015**
- **Edinburgh Student Experience Survey (ESES), closes March 1, 2015**
- Postgraduate Taught Experience Survey (PTES), closes June 15, 2015
- Postgraduate Research Experience Survey (PRES), closes May 14, 2015

Undergraduate Surveys 2015



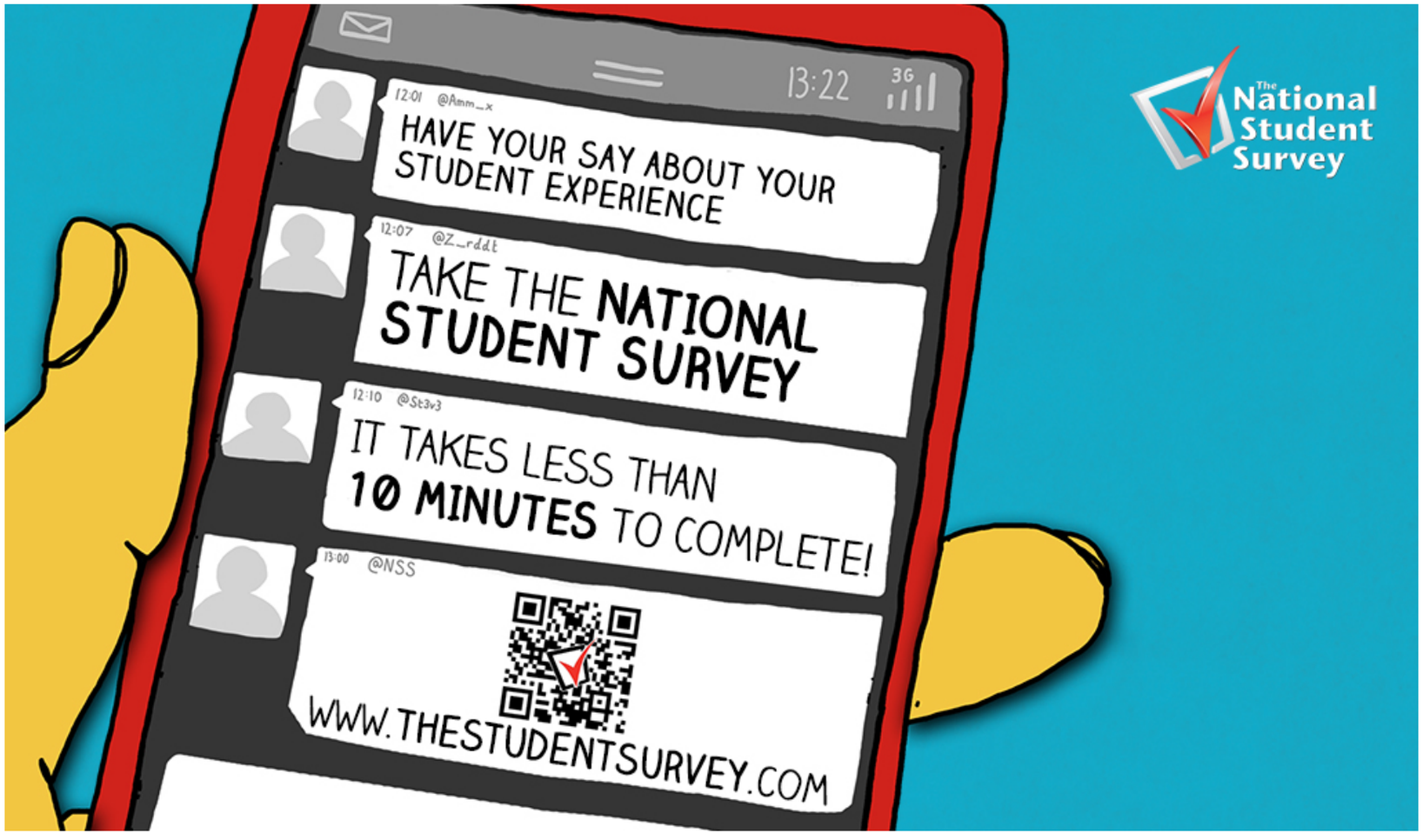
THE UNIVERSITY of EDINBURGH

The National Student Survey
www.thestudentsurvey.com

ON THE SAME PAGE AS YOU
www.ed.ac.uk/same-page

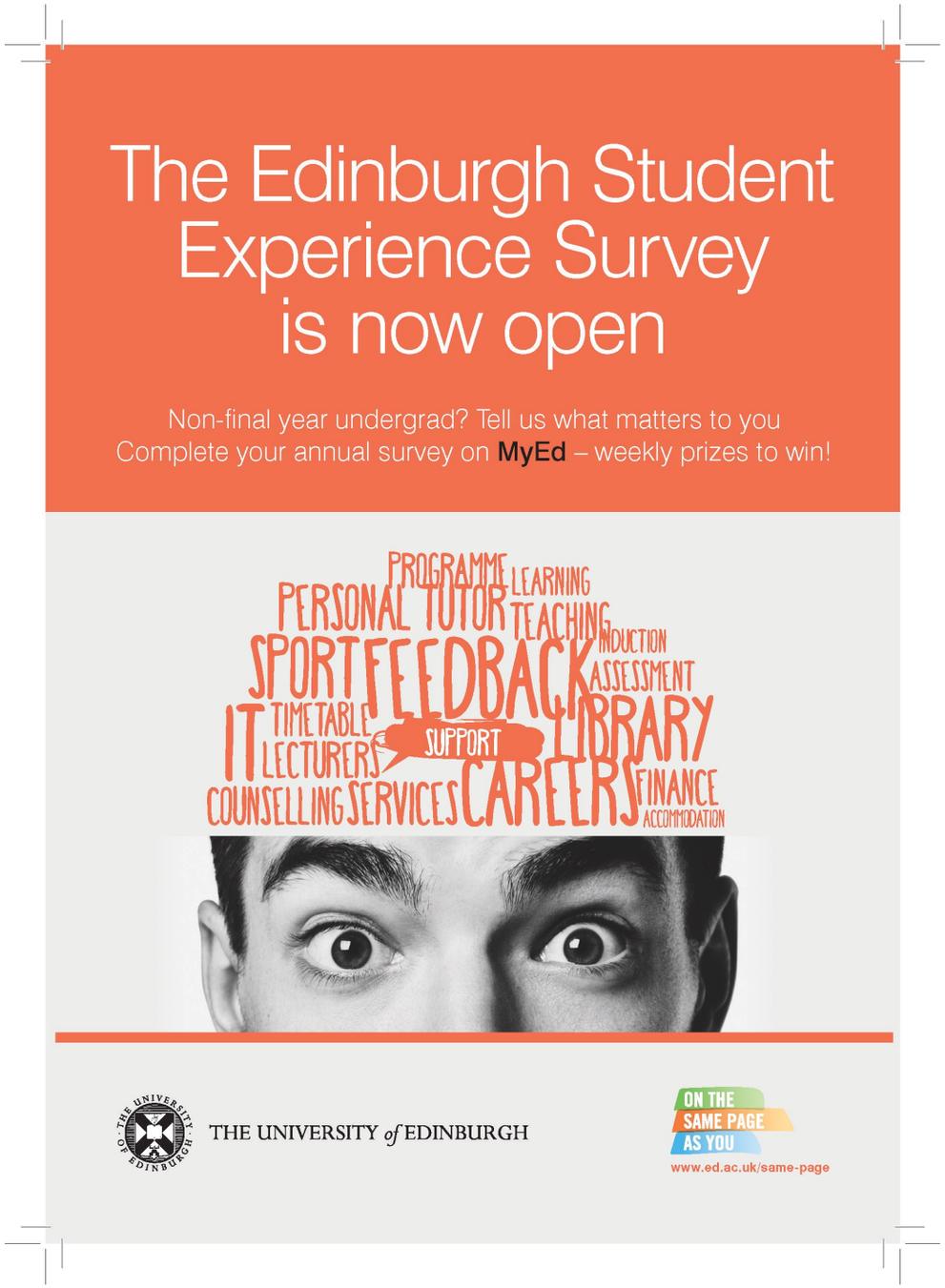
Two surveys for undergraduate students now open

- National Student Survey (NSS)
  - If you are an undergraduate in your FINAL year



Two surveys for undergraduate students now open

- Edinburgh Student Experience Survey (ESES)
  - If you are an undergraduate not yet in your final year



What do the surveys cover?

- Teaching on programme
- Academic support
- Organisation & management
- Learning resources
- Assessment & feedback
- Personal development
- Overall experience

How are the findings used?

Your responses are vital – and we are listening!

- Previous findings have resulted in:
  - Improvements to libraries, study and social spaces
  - New assessment and feedback processes
  - Student Information Points
  - Peer Support initiatives
  - Personal Tutors

Will my responses remain private?

- Yes, Ipsos MORI analyses the NSS data and the University analyses the Edinburgh Student Experience data
- All responses are confidential
- Individuals are not identified when results are reported

How do I complete my survey?

- Check your University email – a personalised URL has been sent to you
- Log in to MyEd – where you will see an announcement and a link to your survey
- Final year students can also complete the NSS survey on smartphones (go to www.thestudentsurvey.com in your web browser)

Incentives

By completing the survey you:

- Will be entered into a prize draw to win £100 Tesco vouchers, £20 Amazon vouchers
- Could win print credit at central promotional events by completing your survey then and there
- And most importantly – you'll help shape the student experience!

What matters to you?

Visit www.myed.ac.uk to complete your survey today

THE UNIVERSITY of EDINBURGH

The National Student Survey
www.thestudentsurvey.com

ON THE SAME PAGE AS YOU

www.ed.ac.uk/same-page

Coursework Feedback

General Considerations

- Networked home security system (Internet of Things…)
  - Security is paramount!
    - Intruder might attack the security system
    - Have you considered this?
  - Reliability
    - False alarm is annoying.
    - Missing an alarm is catastrophic.
      - How do you deal with a system crash?
- Maintainability/Extensibility
  - Currently four sensors
    - Could extend in the future. Have you considered this?

General Observations

- Poor documentation (comments)
  - Try maintaining a large, undocumented SW project, or add some new functionality to it!
- Poor C programming standards
  - You will have to improve your C programming skills if you really want to work in this area!
- Not particularly well tested systems
  - Testing often takes as much time as implementation!

Defensive Programming

- Have you used static analysis tools, e.g. Lint?
- Pairwise programming?
- Code review?
- Designed test cases before implementation? Followed thorough test regime?

Comments

- Poorly commented code
  - Lack of comments
  - Quality of comments
- Better:
  - Comment each function: what does it do? Is there anything special about it (callback, interrupt handler)? What is the meaning of its parameters (any assumptions)? What is its return value? Does it have side-effects/modify state?
  - Comment each data structure: what is it used for? Separate payload/ auxiliary fields (e.g. in linked list)
  - Comment each variable if use is not immediately obvious and local: what is used for?
  - Comment all unusual functionality!

Insecure Code

- The basic rule of thumb is: "I'm not aware of all types of security exploits. I must protect against those I do know of and then I must be proactive!”
- Apply this to your CW:
  - Main vulnerability is web interface, in particular, CGI string parsing
  - User might submit malicious request

External Attacks

- Attacker might submit invalid CGI query/parameters
  - Too long
  - Invalid time
  - Not formatted according to expectations
- Denial of Service Attack
  - Send requests at high frequency

Insecure Code Example

Bad:

```
int risky_programming(char *input){
  char str[1000+1];     // one more for the null character
  // ...
  strcpy(str, input);   // copy input
  // ...
}
```

Good:

```
int secure_programming(char *input){
  char str[1000];
  // ...
  strncpy(str, input, sizeof(str)); // copy input without exceeding the length of the destina
  str[sizeof(str) - 1] = '\0'; // if strlen(input) == sizeof(str) then strncpy won't NUL term
  // ...
}
```

More Insecure Code

Generally, check length of string before accessing it:

```
if (result[30] == '-') …
```

What if string contains less than 31 characters?

Code Duplication

- Copy&paste of time conversion code
  - Possible errors multiplied. Might be missed if errors are detected and fixed
  - Code size expansion! Waste of resources
- Better:
  - Outline common code
  - Write a function for time conversion and call whenever needed

Scalability Issues

```
int alarm1_activated;
int alarm2_activated;
int alarm3_activated;
int alarm4_activated;
...
```

This does not scale!

Better:

```
#define MAX_SENSORS
int alarm_activated[MAX_SENSORS]
```

More Scalability Issues

Horrible:

```
if (alarm_state1) {
  // Do something
}
if (alarm_state2) {
  // Do something
}
if (alarm_state3) {
  // Do something
}
if (alarm_state4) {
  // Do something
}
```

Better:

```
for(i = 0; i < MAX_SENSORS; i++)
{
  if (alarm_state[i]) {
    // Do something
  }
}
```

More Code Duplication

- Copy&Paste of “press button callback”
  - button1_callback, button2_callback, …
  - All contain the same code
  - Same problem as before, plus not scalable (think of 100’s of sensors/buttons)

- Better:
  - Combine functionality into a single callback function
    - Deal with same functionality in a single block
    - Grouping together (e.g. buttons 1-4 in a single switch/case block)

Scalability Issues/Code Duplication

```
void button1_pushed();

void button2_pushed();

void button3_pushed();

void button4_pushed();
```

Better:

```
void button_pushed(int button){
  switch (button) {
    case 1: …
    case 2: …

    default:…
  }
}
```

Hardcoded Constants

```
if (action == 0) {
 // do this
}
else if (action == 1) {
 // do something different
}
```

Better:

```
typedef enum {GET_TIME, SET_TIME} action_t;
action_t action = …
if (action == GET_TIME) {
 // do this
}
else if (action == SET_TIME) {
 // do something different
}
```

Reading of RTC

```
time_T get_time() {
  return t;
}
while (1) {
  t = read_RTC(); // Read RTC periodically
  // Do stuff
}
```

Better:

```
time_T get_time() {
  return read_RTC(); // Read RTC on demand
}
```

PhD Positions

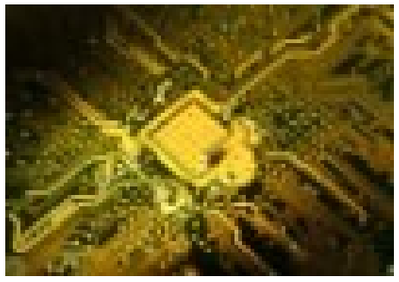
lfcs | Laboratory for Foundations of Computer Science

cisa | Centre for Intelligent Systems and their Applications

```
lemma resPres:
  fixes P :: pi
  and   Q :: pi
  and   x :: name

  assumes PBiSimQ: "P ~ Q"

  shows "<νx>P ~ <νx>Q"
proof -
  let ?X = "{x. ∃P Q. P ~ Q ∧ (∃a. x = (<νa>P, <νa>Q))}"
  from PBiSimQ have "(<νx>P, <νx>Q) ∈ ?X" by blast
  moreover have "⋀P Q a. P ↝[bisim] Q ⟹ <νa>P ↝[(?X ∪ bisim)] <νa>Q"
  proof -
    fix P Q a
    assume PSimQ: "P ↝[bisim] Q"
    moreover have "⋀P Q a. P ~ Q ⟹ (<νa>P, <νa>Q) ∈ ?X ∪ bisim" by blast
    moreover have "bisim ⊆ ?X ∪ bisim" by blast
    moreover have "eqvt bisim" by(rule eqvt)
    moreover have "eqvt (?X ∪ bisim)"
      by(auto simp add: eqvt_def dest: eqvtI)+
    ultimately show "<νa>P ↝[(?X ∪ bisim)] <νa>Q"
      by(rule stronglateSimPres.resPres)
  qed

  ultimately show ?thesis using PBiSimQ
    by(coinduct, blast dest: unfoldE)
qed
```

icsa | Institute for Computing Systems Architecture

epcc



Program

Specify

Design

Verify

PhD in Pervasive Parallelism

Optimize

Apply

http://pervasiveparallelism.inf.ed.ac.uk

intel

Agilent Technologies

Microsoft

ALTRAN

criticalblue
Accelerating Embedded Software

QUALCOMM

IBM

codeplay

ACE

SAMSUNG

ARM

freescale™

ORACLE

wolfson microelectronics

SYNOPSYS

amazon

ARM iCASE Studentship

- “Profile-Directed Parallelisation of Sequential Legacy Applications”
- In collaboration with ARM Ltd., Cambridge
- ARM mentor, internships, access to technology
- Student eligibility requirements for EPSRC Industrial CASE funding are:
  - A relevant connection with the UK, usually established by residence, and
  - an upper second class honours degree, or a combination of qualifications and/or experience equivalent to that level.
- This Studentship will cover tuition fees at the UK/EU rate and provide a tax-free stipend at the EPSRC rate. Students receive funding for a full EPSRC studentship for 3.5 years (currently ~£68,648).
- Contact: Björn Franke (bfranke@inf.ed.ac.uk)