CS 430 - Lecture 10 - Tools of the Trade II

Collin Roberts

October 17, 2023

Outline

Taxonomy of CASE

- Recall, CASE stands for **Computer Aided/Assisted Software Engineering**, not **Computer Automated Software Engineering**.
- At present, a computer is a tool of, and not a replacement for, a software professional.

Taxonomy of CASE

- CASE tools used during the
  1. earlier workflows (requirements, analysis, design) are called **front-end** or **upperCASE** tools, and
  2. later workflows (implementation, postdelivery maintenance) are called **back-end** or **lowerCASE** tools.

Examples

1. **data dictionary** - list of every data item defined in the software product. Some things to include:
   1. an English description of every item in the dictionary
   2. **Module names** ✓
   3. **Procedure names:** ✓
      1. parameters, and
      2. their types,
      3. locations where they are defined (i.e. which module), and
      4. description of purpose
   4. **Variable names:** ✓
      1. types, and
      2. locations (i.e. which module & procedure) where they are defined

Examples

2. **consistency checker** - to confirm that every data item in the specification document is reflected in the design, and vice versa.
3. **report generator**
4. **screen generator** - for creating **data capture** screens.

Taxonomy

1. Combining multiple tools creates a **workbench**.
2. Combining multiple workbenches creates an **environment**.
3. So our taxonomy is tools (task level) → workbenches (team level) → environments (organization level).

Scope of CASE

Primary motivations for implementing CASE:

1. Produce high-quality code.
2. Have up-to-date documentation at all times.
3. Automation makes maintenance easier.
4. Do everything more quickly, hence more cheaply.

Scope of CASE

- For example, if a specification is created by hand, there may not be any way to tell whether the document is current by reading it.
- On the other hand, if the specification is maintained within CASE software, then the latest version is the one the CASE software displays.

Scope of CASE

- Similarly, other documentation about the software is easier to maintain inside of CASE software.
- **Online documentation**, **word processors**, **spreadsheets**, **web browsers**, and **email** are CASE tools.

Scope of CASE

Coding tools of CASE include

1. **text editors** (including **structure editors** which are sensitive to syntax, including **online interface checking**), **debuggers**, **pretty printers** / **formatters**, etc.

Scope of CASE

- An **operating system front end** allows the programmer to issue operating system commands (e.g. compile, link, load) from within the editor.
- A **source-level debugger** automatically causes trace output to be produced. An **interactive source-level debugger** is what its name says.

Scope of CASE

- **Programming-in-the-small:** coding a single module.
- **Programming-in-the-large:** coding at the system level.
- **Programming-in-the-many:** software production by a team.

Revisions

Definition 1

A **revision** *is created when a change is made, e.g. to fix a fault.*

Revisions

Old revisions must be retained for reference, e.g.

1. if a fault is found at a site still running the old revision,
2. for auditing and
3. for other reasons.

Variations

Definition 2

A **variation** *is a slightly changed version that fulfills the same role in a slightly changed situation.*

Examples

1. two printer drivers, one for a laser printer and one for an inkjet printer, or
2. optimizing an application to run on different platforms, e.g. desktop vs. smart phone.

Remarks

1. Often the variation is also embedded into the file name.

Moral

1. A CASE tool is needed to effectively manage multiple revisions of multiple variations.

Configuration Control

Definition 3

A **configuration** *of a software product is a list, for every code artifact, of which version is included in the S/W product.*

Configuration Control

Definition 4

A **configuration control tool** *is a CASE tool for managing configurations (Definition 3).*

Configuration Control

1. **Motivation:** Fix S/W faults effectively.
2. The first step towards fixing a problem is to recreate it in a development environment.
3. If many configurations are possible, then configuration control will be needed in order to recreate a problem in a development environment.
4. Version control also facilitates ensuring that the correct versions get included when compiling / linking.
5. A common technique is to embed the version as part of the name.

Configuration Control

6. Adding details to a configuration yields a **derivation** of a S/W product:

Configuration Control

Definition 5

A **derivation** *is a detailed record of a version of the S/W product, including*

1. *the variation/revision of each code element (i.e. the* **configuration***),*
2. *the versions of the compilers/linkers used to assemble the product,*
3. *the date/time of assembly, plus*
4. *the name of the programmer who created the version.*

Configuration Control

7. A **version-control** tool is required to effectively track derivations.

Configuration Control During Postdelivery Maintenance

1. There is an obvious problem when a team maintains a software product.
2. Suppose that two different programmers receive two different fault reports. Suppose further that fixing both faults require changes to the same code artifact.
3. Without any new process in place, the programmer #2 will undo programmer #1's changes at deployment time.
4. See the next subsection for a possible solution to this problem, using **baselines**.

Baselines

1. When multiple programmers are working on fixing faults, a **baseline** is needed.
2. A **baseline** is a set of versions of all the code artifacts in a project (i.e. what versions are in production right now).
3. A programmer starts by copying the baseline files into a **private workspace**. Then he/she can freely change anything without affecting anything else.
4. The programmer **freezes** the version of the artifact to be changed to fix the fault. No other programmer can modify a frozen version.
5. After the fault is fixed, the new code artifact is promoted to production, modifying the baseline.
6. The old, frozen version is kept for future reference, and can never be changed.
7. This technique extends in the natural way to multiple programmers and multiple code artifacts.

Instructor Remark

1. In my experience, the strict technique described here is too slow. Instead developer #2 starts work right away, and incorporates developer #1's changes as soon as they are promoted to production. SQA needs to be kept informed in this situation!
2. One could argue that this technique is vulnerable to exponential growth of effort as the number of faults in a code artifact increases. The instructor counter-argues that if we achieve **separation of concerns** in our software products, then the probability of $>> 2$ simultaneous faults in one code artifact is low.

Student Question

What if #1 and #2 actually touch the same code?
Instructor Answer: I recommend using the same technique, being mindful that extra care will be needed when

1. incorporating #1's changes into #2's version, and
2. doing SQA (e.g. what should be the test cases and expected results for pass 0 and for pass 1?).

Configuration Control During Development

- During Development and Desk Checking, changes are too frequent for configuration control to be useful.
- We definitely want configuration control to be in force by the time we deploy to production.
- The text author recommends that configuration control should apply once the code artifact is passed off to the SQA group.
    1. In practice, we can decide when between the end of development and the time of deployment to begin enforcing configuration control.

Configuration Control During Development

- The same configuration control procedures as those for postdelivery maintenance should then apply.
- Proper version control permits management to take corrective action if project deadlines start to slip (as they are then aware of the status of every code artifact).

Build Tools

Definition 6

A **build tool** *selects the correct compiled-code artifact to be linked into a specific version of the S/W product.*

Build Tools

- Some organizations may not want to purchase a complete configuration-control solution.
- Then at least a version control tool must be used in conjunction with a **build tool** (Definition 6).

Issue

While a version control tool assists programmers in deciding which version of the source code is the latest, compiled code does not automatically get a version attached to it. Possible solutions (present in class only if time permits):

1. Automatically re-compile and re-link every night. Obviously this is expensive.
2. Use a tool like `make` to decide more intelligently, based on date and time stamps of compiled code. This idea has been incorporated into many different programming environments.

Student Question

What is the difference between a **build tool** (Definition 6) and a **configuration control tool** (Definition 4)?

Answer:

1. The purpose of a **build tool** is to make certain we have the correct compiled code artifacts linked in to a specific version of the S/W product. This can be effective for a small organization, managing one version of a S/W product at one location. This explains why auto-recompiling each night is a viable technique.
2. A **configuration control tool** is needed to manage multiple revisions of multiple variations. E.g. for a large organization which must manage multiple configurations running simultaneously across multiple locations.

Productivity Gains with CASE Technology

1. Research to date shows a modest gain in productivity following the introduction of CASE tools to an organization.
2. Other benefits of using CASE tools:
    1. faster development
    2. fewer faults
    3. better usability (e.g. from a screen generator)
    4. easier maintenance
    5. improved morale on the IT team

Productivity Gains with CASE Technology

This list of CASE tools is summarized in Figure 5.14 in the text.

Build tool (§5.11)	Coding tool (§5.8)
Configuration-control tool (§5.10)	Consistency checker (§5.7)
Data dictionary (§5.7)	E-mail (§5.8)
Interface checker (§5.8)	Online documentation (§5.8)
Operating system front end (§5.8)	Pretty printer (§5.8)
Report generator (§5.7)	Screen generator (§5.7)
Source-level debugger (§5.8)	Spreadsheet (§5.8)
Structure editor (§5.8)	Version-control tool (§5.9)
Word-processor (§5.8)	World Wide Web browser (§5.8)