Software Engineering Meets Program Verification: Incremental Development & Continuous Checking

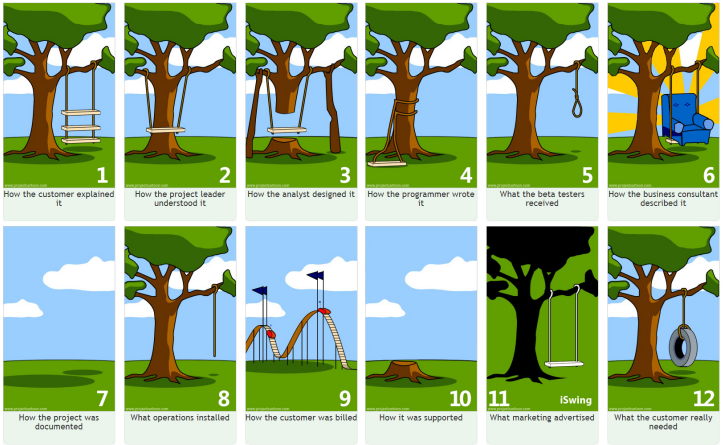
Manuel Bentele

University of Freiburg

Hahn-Schickard

16th Alpine Verification Meeting (AVM'24)
September 04, 2024

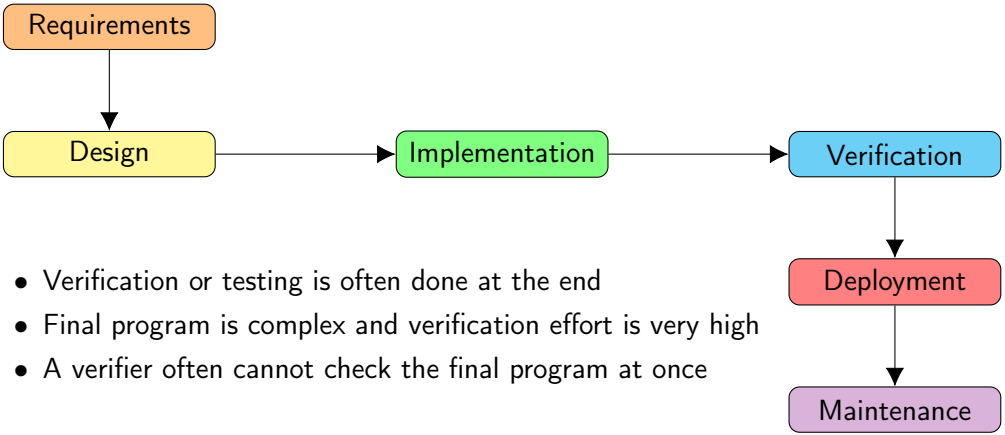
How to (not) develop new products?



https://pmac-agpc.ca/project-management-tree-swing-story

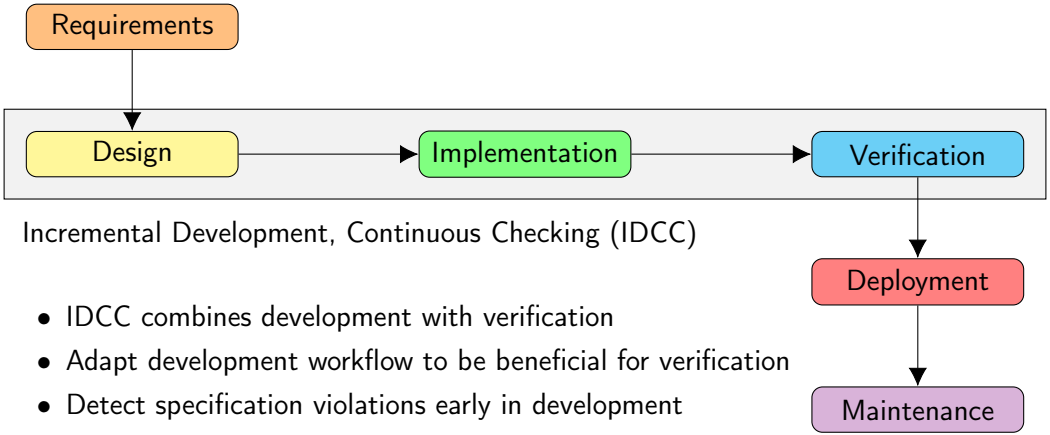
- Product engineering can be complicated
- Steps can go wrong during the development process
- Quality assurance (verification & validation) is important

How does a software development process look like?



- Verification or testing is often done at the end
- Final program is complex and verification effort is very high
- A verifier often cannot check the final program at once

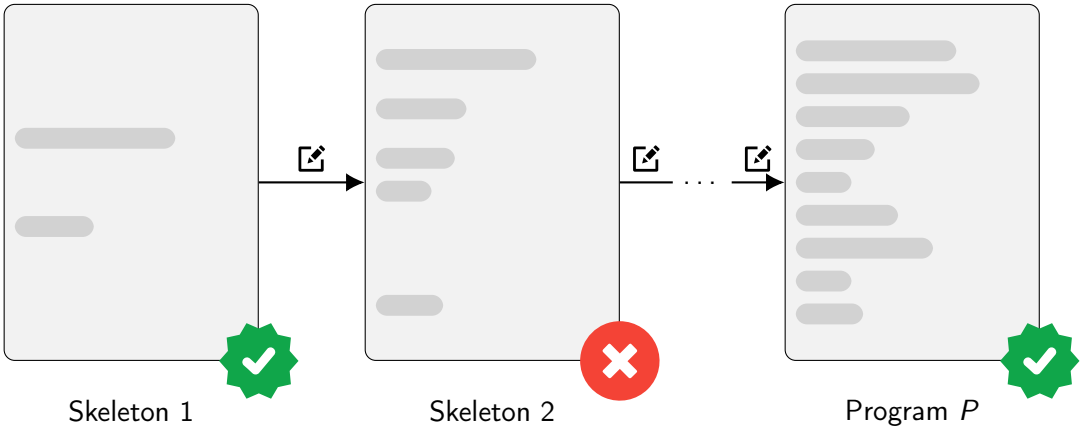
How is software engineering combined with program verification?



Incremental Development, Continuous Checking (IDCC)

- IDCC combines development with verification
- Adapt development workflow to be beneficial for verification
- Detect specification violations early in development

How does the incremental development workflow look like?



How does the incremental development workflow look like? (cont'd)

- Develop program increment by increment:
  - Start with a program skeleton and
  - refine skeleton (until final program P implements all functionality) while
  - checking each program revision after an increment is created

- Development adaptions for continuous checking (program verification):
  - Initial program skeleton should contain as much control flow as possible
  - More data flow and calculations are added while refining the skeleton

What software do we develop with IDCC and what specification is checked?

- IDCC is applied in the context of embedded systems to develop embedded software written in the language C
- Temporal dependencies are used as specification:
  - Interface specification for Hardware Abstraction Layers (HALs)
  - "Does each program revision uses hardware access functions correctly?"
  - E.g., call of `HAL_Serial_Send()` must be preceded by a call of `HAL_Init()`
- Violations of temporal dependencies are crucial and could lead to system failures
- Temporal dependencies can be checked early in development (e.g., in a skeleton)

How to apply IDCC using an example?

```
void main() {
  int avg = 0;
  HAL_Init();
  while (*) {
    HAL_Read_Analog();
    while (*) {
      HAL_Delay_Ns();

    }



    HAL_Serial_Send(avg);

  }
}
```

Skeleton

```
void main() {
  int avg = 0, cnt = 0, val = 0;
  HAL_Init();
  while (1) {
    val = HAL_Read_Analog();
    while (cnt < 10000) {
      HAL_Delay_Ns();
      cnt++;
    }
    if (val >= 0) {
     avg = (avg + val) / 2;
    }
    HAL_Serial_Send(avg);
    cnt = 0;
  }
}
```

Program

How to apply IDCC using an example? (cont'd)

- Skeleton:
  - Sketches the control flow for final program
  - Contains required HAL function calls
  - Introduces non-determinism for all unknown choices[1] (if possible)
- Increment:
  - Refines skeleton by adding data flow (changes marked in green)
  - Restricts non-determinism by making expressions more concrete
- Program:
  - Final program cannot be verified by Ultimate Automizer with defaults at once in a reasonable time
  - But incremental verification with Ultimate Automizer succeeds (reuses computation results of the verified skeleton)

[1] Non-deterministic choices are denoted in the code with the symbol *

Conclusion

What has been done?

- Development process to combine software engineering with program verification
- Program verification is carried out continuously during development
- Verify complex programs that cannot be checked by a verifier at once

What is currently being done?

- Redevelopment of two real-world programs for a sensor system via IDCC

What are the next steps?

- Include concurrent programs with interrupts in the development process
- Implement witness-guided verification in Ultimate to (re)use invariants from witnesses for incremental verification