RW344: Software Design

Bernd Fischer
bfischer@cs.sun.ac.za
Software Process Models
Topics

- software process
- software process phases
- waterfall models
- V-models
How do you build software?

I'll go up and find out what they need and the rest of you start coding!
Software processes

A software process is a **structured set of activities** followed to develop a software system.
A software process is a **structured set of activities** followed to develop a software system.
Software process
also called software life cycle

A software process is a **structured set of activities** followed to develop a software system.

There are four generic activities or **phases**:

- **specification**: identifying the required services and the constraints on system operation & development
- **development**: converting the system specification into an executable system
- **validation**: showing that a system conforms to its specification and meets the user requirements
- **evolution**: adapting software to changes in circumstances
A software process is a structured set of activities followed to develop a software system.

There are four generic or phases:

- **specification**: identifying the required services and the constraints on system operation & development
- **development**: converting the system specification into an executable system
- **validation**: showing that a system conforms to its specification and meets the user requirements
- **evolution**: adapting software to changes in circumstances

Also called software life cycle.
Software process

A software process is a **structured set of activities** followed to develop a software system.

There are four generic or **phases**:

- **specification**: identifying the required services and the constraints on system operation & development
- **development**: converting the system specification into an executable system
- **validation**: showing that a system conforms to its specification and meets the user requirements
- **evolution**: adapting software to changes in circumstances

also called **software life cycle**
A software process is a structured set of activities followed to develop a software system.

There are four generic activities or phases:

- **specification**: identifying the required services and the constraints on system operation & development
- **development**: converting the system specification into an executable system
- **validation**: showing that a system conforms to its specification and meets the user requirements
- **evolution**: adapting software to changes in circumstances

Also called software life cycle
There are four generic activities or phases:

- **Specification**: identifying the required services and the constraints on system operation & development
- **Development**: converting the system specification into an executable system
- **Validation**: showing that a system conforms to its specification and meets the user requirements
- **Evolution**: adapting software to changes in circumstances

A software process is a **structured set of activities** followed to develop a software system. Also called **software life cycle**

- **What the system does**: what the system does
- **How the system works**: how the system works
- **That the system works**: that the system works
- **Keep the system working**: keep the system working
Software processes

Different software processes are possible that differ in
• how the phases are arranged
• how much attention is paid to each phases
  – whether and how the phases are broken down
• what must be done in each phase
• when and how to transition between phase
  – what must be delivered by each phase
Software processes

Different software processes are possible that differ in
• how the phases are arranged
• how much attention is paid to each phases
  – whether and how the phases are broken down
• what must be done in each phase
• when and how to transition between phase
  – what must be delivered by each phase

“Null process:”

devolution validation
Software processes

Different software processes are possible that differ in:

• how the phases are arranged
• how much attention is paid to each phases
  – whether and how the phases are broken down
• what must be done in each phase
• when and how to transition between phases
  – what must be delivered by each phase

“Null process:”

```
development  validation
```

“code and fix”
Specification

**specification**: identifying the required services and the constraints on system operation & development

Main concern: **what** should the system do?

Main result: **requirements** == description of desired system functionality

- **user requirements**: abstract, natural-language
  - aimed at users, customer, and contractor

- **system requirements**: specific, semi-formal
  - aimed at developers and contractor
Specification

Main concern: what should the system do?

Main result: requirements == description of desired system functionality

• **user requirements**: abstract, natural-language
  – aimed at users, customer, and contractor

• **system requirements**: specific, semi-formal
  – aimed at developers and contractor
Requirements engineering tasks

- **feasibility study**: check whether a new system is needed, and whether it can be built within the given time and budget

- **requirements elicitation**: gather requirements by
  - user interviews and questionnaires
  - user observation (ethnographic studies)
  - brainstorming and role playing
  - use cases and prototyping.

- **requirements analysis**: classify, organize, and prioritize requirements
  - can prompt more elicitation
Requirements engineering tasks

- **requirements specification**: translate chosen requirements into coherent document(s)
  - different specification methods

- **requirements validation**: check requirements for realism, consistency, and completeness
  - interleaved with analysis and specification
  - can prompt more elicitation
  - will inevitably lead to the discovery of errors that must be fixed in the user and system requirements
Requirements engineering is hard...

I'll need to know your requirements before I start to design the software.

First of all, what are you trying to accomplish?

I'm trying to make you design my software.

I mean what are you trying to accomplish with the software?

I won't know what I can accomplish until you tell me what the software can do.

Try to get this concept through your thick skull: the software can do whatever I design it to do!

Can you design it to tell you my requirements?
Development

development: converting the system specification into an executable system

Main concern: how should the system work?
Development = Design + Implementation

development: converting the system specification into an executable system

Traditionally broken down into several stages:

- architectural design
- interface design
- abstract specification
- coding

- component design
- data structure design
- algorithm design
- debugging
Development = Design + Implementation

development: converting the system specification into an executable system

Traditionally broken down into several stages:

- architectural design
- interface design
- abstract specification
- coding
- component design
- data structure design
- algorithm design
- debugging

development is an iterative process with feedback between the stages

design and implementation are typically interleaved
Validation

**validation**: showing that a system conforms to its specification and meets the user requirements

Main concern: **does** the system work?
Verification and Validation

**validation**: showing that a system conforms to its specification and meets the user requirements

Traditionally broken down into two separate activities:

- **verification**: do we build the system right?
- **validation**: do we build the right system?
Verification and Validation

**validation**: showing that a system conforms to its specification and meets the user requirements

Traditionally broken down into two separate activities:
- **verification**: do we build the system right?
- **validation**: do we build the right system?

V&V techniques:
- reviews (code and documents)
- prototyping and simulation (documents)
- **testing**
- static analysis (e.g., model checking, proof)
Verification and Validation

Testing works at different levels:

- **unit testing**: individual components
  - methods; test functional behavior (pre/post)
- **module testing**: groups of related components
  - classes; test class invariants
- **system testing**: whole (sub-) system(s)
  - test emergent properties (e.g., security, reliability,...)
- **acceptance testing**: use customer data
  - test that system meets user expectations
Verification and Validation

Testing works at different levels:

- **unit testing**: individual components
  - methods; test functional behavior (pre/post)
- **module testing**: groups of related components
  - classes; test class invariants
- **system testing**: whole (sub-) system(s)
  - test emergent properties (e.g., security, reliability,...)
- **acceptance testing**: use customer data
  - test that system meets user expectations

Each development phase corresponds to a V&V phase!
Evolution

**evolution**: adapting software to changes in circumstances

Main concern: **keep the system working**
Evolution

Main concern: keep the system working

Evolution: adapting software to changes in circumstances

also called maintenance
Evolution: 

Main concern: keep the system working

Traditionally broken down into several categories:

- **corrective**: fixing bugs in the field
- **adaptive**: respond to changes in environment
- **perfective**: respond to changes in user requirements
- **preventive**: improve or refactor system to prevent future problems

More information:

also called maintenance
Evolution

Main concern: **keep** the system working

Traditionally broken down into several categories:

- **corrective**: fixing bugs in the field
- **adaptive**: respond to changes in environment
- **perfective**: respond to changes in user requirements
- **preventive**: improve or refactor system to prevent future problems

More information:

also called **maintenance**

**evolution**: adapting software to changes in circumstances
Distribution of efforts

- Design: 15%
- Requirements engineering: 10%
- Specification: 10%
- Coding: 20%
- Testing: 45%
Distribution of efforts

observed project numbers (not target numbers!)
Distribution of efforts

Verification effort dominates both specification and development efforts!

observed project numbers (not target numbers!)
Distribution of efforts

observed project numbers (not target numbers!)

- maintenance: 75%
- testing
- coding
- design
- requirements engineering
- specification
Distribution of efforts

observed project numbers (not target numbers!)

... but maintenance (50-75%) dominates the total life cycle effort!
Distribution of maintenance efforts

Corrective: 21%

Adaptive: 25%

Preventive: 4%

Perfective: 50%

Source:
Distribution of maintenance efforts

Requirements-driven changes dominate maintenance efforts!

Source:
Software process modeling

A software process is a **structured set of activities** followed to develop a software system.
A software process model is a generic description of a class of software processes.
A software process model is a generic description of a class of software processes.

- typically described informally
  - “manifestos”, books, standards
  - case studies
Software process modeling

A software process model is a generic description of a class of software processes.

- typically described informally
  - “manifestos”, books, standards
  - case studies
- can also be formalized / implemented
  - foundation for project management systems

More information:
A software process model is a generic description of a class of software processes.

- typically described informally
  - “manifestos”, books, standards
  - case studies
- can also be formalized / implemented
  - foundation for project management systems
- we will identify software process and software process model

More information:
Dimensions of software processes
Dimensions of software processes

- **linear vs. iterative**
  - phases are *executed in order*
  - phases are *repeated regularly*
Dimensions of software processes

- **linear vs. iterative**
  - phases are *executed in order*
  - phases are *repeated regularly*

- **monolithic vs. incremental**
  - single system is delivered *at end*
  - system is delivered in *several working versions* with increasing functionality
Dimensions of software processes

- **linear vs. iterative**
  - phases are *executed in order*
  - phases are *repeated regularly*

- **monolithic vs. incremental**
  - single system is delivered *at end*
  - system is delivered in *several working versions* with increasing functionality

- **document-based vs. code-based**
  - most *deliverables* (except final system) are *text*
  - almost all *deliverables are code* (prototypes, test cases, ...)
Dimensions of software processes

• **heavy-weight** vs. **light-weight**
  – strict rules and detailed plans, formal models and methods
  – flexible and reactive, prototyping
  – also called **plan-driven** and **agile**
Dimensions of software processes

• **heavy-weight** vs. **light-weight**
  – strict rules and detailed plans, formal models and methods
  – flexible and reactive, prototyping
  – also called plan-driven and agile

• **component-driven** vs. **model-driven** vs. **test-driven**
Dimensions of software processes

• **heavy-weight** vs. **light-weight**
  – strict rules and detailed plans, formal models and methods
  – flexible and reactive, prototyping
  – also called **plan-driven** and **agile**

• **component-driven** vs. **model-driven** vs. **test-driven**
  – requirements are modified to fit existing components
Dimensions of software processes

- **heavy-weight vs. light-weight**
  - strict rules and detailed plans, formal models and methods
  - flexible and reactive, prototyping
  - also called plan-driven and agile

- **component-driven vs. model-driven vs. test-driven**
  - requirements are modified to fit existing components
  - requirements are formulated as (formal) models
Dimensions of software processes

- **heavy-weight vs. light-weight**
  - strict rules and detailed plans, formal models and methods
  - flexible and reactive, prototyping
  - also called plan-driven and agile

- **component-driven vs. model-driven vs. test-driven**
  - requirements are modified to fit existing components
  - requirements are formulated as (formal) models
  - requirements are formulated as test cases
Dimensions of software processes

- **heavy-weight** vs. **light-weight**
  - strict rules and detailed plans, formal models and methods
  - flexible and reactive, prototyping
  - also called plan-driven and agile

- **component-driven** vs. **model-driven** vs. **test-driven**
  - requirements are modified to fit existing components
  - requirements are formulated as (formal) models
  - requirements are formulated as test cases

Note: not all combinations possible or sensible...
Code-and-fix

- No specification, no design
- Just dive in and starting coding
Code-and-fix

- No specification, no design
- Just dive in and starting coding

Advantages
Code-and-fix

- No specification, no design
- Just dive in and starting coding

Advantages
- no overhead
- see progress quickly
- suitable for small, short-lived programs (<200 lines)
Code-and-fix

- No specification, no design
- Just dive in and starting coding

Advantages
- no overhead
- see progress quickly
- suitable for small, short-lived programs (<200 lines)

Disadvantages
Code-and-fix

- No specification, no design
- Just dive in and starting coding

**Advantages**
- no overhead
- see progress quickly
- suitable for small, short-lived programs (<200 lines)

**Disadvantages**
- no way to assess progress, quality, or risks
- changes likely to require major design overhaul
Code-and-fix

- No specification, no design
- Just dive in and starting coding

**Advantages**
- No overhead
- See progress quickly
- Suitable for small, short-lived programs (<200 lines)

**Disadvantages**
- No way to assess progress, quality, or risks
- Changes likely to require major design overhaul

Dangerous for all projects but homeworks!!
Classic waterfall model

requirements

design

implementation

validation

evolution
Classic waterfall model

- requirements
- design
- implementation
- validation
- evolution

• each phase **must** be completed before next is started
Classic waterfall model

- each phase **must** be completed before next is started
- each phase transition is marked by delivery of corresponding documents

- requirements
- design
- implementation
- validation
- evolution

• doc.
• doc.
• code
• doc.
Classic waterfall model

- Each phase **must** be completed before next is started.
- Each phase transition is marked by delivery of corresponding documents.

More information:
Classic waterfall model: Advantages

- phase distinction and order encourages discipline
  - can provide support for inexperienced development teams
- emphasis on communication through documents
  - promotes documentation
  - support for off-shoring
  - allows automation (if documents are formal models)
- implementation postponed until project objectives are well-understood
  - minimizes risk of waste
Classic waterfall model: Disadvantages

• inflexible partitioning of project into distinct stages
  – no feedback, but specification problems often show only during coding or integration
  – validation too late

• first working version available only very late
  – little feedback from users until then
  – increases risk of obsolescence

• requires accurate and complete requirements right from the start
  – difficult to respond to changing requirements

• over-reliance on documents, heavy-weight process
  – documents get out-of-sync
The classic waterfall model is a strawman.

Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial

Source:
The classic waterfall model is a strawman.

"... the implementation described above is risky and invites failure."

Source:
The classic waterfall model is a strawman.

“... the implementation described above is risky and invites failure.”

Figure 2. Implementation steps to develop a computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial

Source:
Modified waterfall model

- previous phase can be revisited when problems occur
Modified waterfall model

- previous phase can be revisited when problems occur
- increases flexibility
Modified waterfall model

- previous phase can be revisited when problems occur
- increases flexibility
- ... but requires more synchronization of documents
Modified waterfall model

- previous phase can be revisited when problems occur
- increases flexibility
- ... but requires more synchronization of documents
- ... and often more than one earlier phase is affected
Modified waterfall model

- previous phase can be revisited when problems occur
  - increases flexibility
  - ... but requires more synchronization of documents
  - ... and often more than one earlier phase is affected
  - ... and validation is still late
V-model: mirror all specification and development steps by validation steps.

Source:
• D. Firesmith: Using V models for testing, http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315
V-model: mirror all specification and development steps by validation steps.

basically still a waterfall model!

Source:
Double-Vee: use and test models.

Source:
• D. Firesmith: Using V models for testing, http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315
Triple-Vee: verify tests.

Source:
Big Design Up-Front models (BDUF)

**BDUF** denotes any model where the design is completed and perfected before the implementation is started.

- derived from waterfall model
- suitable for
  - well-understood domains with stable requirements
  - high-reliability systems
- unsuitable for
  - highly complex systems in ill-understood domains
  - changing requirements
Iterative methods

- spiral model
- rapid application development (RAD) / prototyping
- rational unified process (RUP) / modelling
- agile / lean / scrum development
- eXtreme Programming
Iterative methods

- spiral model
- rapid application development (RAD) / prototyping
- rational unified process (RUP) / modelling
- agile / lean / scrum development
- eXtreme Programming

We will start with a waterfall model and look at these later...
Summary

- Software development process is split into phases:
  - Specification
  - Development
  - Validation
  - Maintenance

- Maintenance dominates total life cycle effort.

- Software process models describe how the phases are arranged and what must be done in each phase.

- Main distinction: Linear vs. Iterative.

- The linear waterfall model is the traditional model:
  - Good for stable requirements, bad for change.